A Comparison of Repositioning and Scheduling Algorithms for the Ride-Hailing Problem

Lauren Paul[1,2], Sebastian Hübner[1], Christina Plump[1,2], Bernhard J. Berger[3], and Rolf Drechsler[1,2]

[1] Faculty of Mathematics and Computer Science, University of Bremen, Germany
[2] German Research Center of Artificial Intelligence – CPS, Bremen, Germany
[3] Institute of Embedded Systems, Hamburg University of Technology, Germany

Abstract. The *Ride-Hailing Problem* is an online optimization challenge that coordinates a fleet of vehicles to serve requests revealed over time. The problem is NP-hard, and many solution approaches use stochastic or machine-learning based algorithms, which perform well but are difficult to interpret. We study the trade-offs between performance and explainability. Our results show that greedy assignment consistently outperforms more complex methods, indicating that explainability need not come at the cost of efficiency. We also find that the effectiveness of repositioning strategies strongly depends on demand patterns, highlighting the value of context-aware policies.

Keywords: ride-hailing problem, dynamic optimisation, repositioning, explainability, scheduling

1 Introduction

Online optimization is central to real-world logistical challenges such as resource allocation and scheduling, where information about incoming jobs is not known ahead of time [25]. The ride-hailing problem is one such online optimization problem with real-world applications. On-demand ride hailing services like Uber and Lyft must match dynamically arising customer requests to drivers in real-time [18, 24]. One primary optimization goal is the maximization of profit; however, customer satisfaction also plays an important role, incentivizing reduced waiting times [4].

Due to the inherent complexity of the ride-hailing problem (RHP) [30], many existing solution approaches rely on stochastic metaheuristics [7, 10, 11, 16, 31] or machine-learning (ML) techniques to optimize scheduling and repositioning strategies [22, 23, 26, 27, 32]. While these methods achieve strong performance in practice, their decision-making processes are often difficult to explain and remain opaque to customers and operators alike. This has fueled growing interest in explainability as a research direction [17]. Explainable optimization algorithms provide justifications for their choices, thereby increasing trust and potentially improving user satisfaction [12, 17].

In this work, we extend existing research on the RHP with a focus on explainability. Specfically, we address the following research questions: (RQ1) How do request frequency and distribution affect the performance of different repositioning strategies? (RQ2) Is there a trade-off between performance and explainability in the investigated RHP algorithms, and how does demand shape their performance?

To answer these questions, we evaluate a variety of approaches, including a transparent greedy heuristic, batching [1] and load-reduction-based approaches [2, 9] adapted from scheduling literature, as well as Xu et al.'s LEARNING-PLANNING algorithm [28], which incorporates machine learning to optimize assignments. Our findings show that the simple greedy strategy consistently outperforms more complex approaches, highlighting that explainable algorithms can achieve strong performance even when compared to more sophisticated ML-approaches. Additionally, we observe that different repositioning strategies perform best under different demand conditions, suggesting that adaptively choosing repositioning strategies based on request patterns could yield additional benefits.

2 Methodology

In this section, we introduce the ride-hailing problem and describe the scheduling and repositioning algorithms developed and evaluated in the remainder of this paper.

2.1 Problem Description

An RHP instance is defined on a directed weighted graph $G = (V, E, w)$, where $w : V \times V \to \mathbb{R}_+$ denotes driving time, with a set $\mathcal{M}$ of m vehicles and a sequence $\mathcal{J}$ of n requests over time horizon $T \in \mathbb{N}_+$. Vehicle locations at time t are given by $\mathcal{X}_t = (x_1^t, \ldots, x_m^t)$, with $\mathcal{X}_0$ specifying starting positions.

Each request $j = (r, s, d)$ has release time r, pick-up location $s \in V$, and drop-off location $d \in V$. A vehicle v serves j upon reaching s at some $t \geq r$ and traveling to d. The completion time of j, C_j, is $r + w(s, d)$. Pick-ups and drop-offs are assumed to be instantaneous.

A feasible solution assigns sequences of requests to vehicles such that no request is served before its release time and no vehicle serves two requests simultaneously.

Solutions are evaluated with respect to **customer satisfaction**, measured by the total *flow time*, i.e., $\sum_{j \in \mathcal{J}} (C_j - r_j)$ and **operating costs**, measured by the cumulative *driving time*, i.e., the total time vehicles spend driving rather than idle: $\sum_{i=1}^{T} \sum_{j=1}^{m} w(x_j^{i-1}, x_j^i)$.

2.2 Scheduling Algorithms

Scheduling algorithms define strategies of assigning requests to vehicles. The following algorithms are used and extended in our application.

Greedy The GREEDY algorithm is an intuitive solution approach well described in the literature [14, 15, 20]. At any time t, let $J_t = \{j = (r', s, d) \mid r' \leq t \wedge j \text{ not yet served}\}$ denote the set of released but unserved requests. If a vehicle finishes its current request at time t and is located at d, it is assigned a request $j \in \arg\min_{j \in J_t} w(d, s_j)$, and $J_t \leftarrow J_t \setminus \{j\}$. If $J_t = \emptyset$, the vehicle stays idle until a request is released. This is repeated until all requests have been served.

Least-Expected-Load The *load* of a vehicle v, denoted L_v, is defined as the time needed for v to serve all requests currently assigned to it. Let d_f^v denote the drop-off location of the final request assigned to v. Then, upon receiving a new request $j = (r, s, d)$, the LEAST-EXPECTED-LOAD algorithm assigns j to some $v \in \arg\min_{v \in \mathcal{V}}(L_v + w(d_f^v, s) + w(s, d))$, extending the greedy list-scheduling algorithm for load-balancing on related machines [2, 9]. We also incorporate a background swap-based local search that periodically exchanges two jobs in a randomly selected vehicle's schedule until no improving swaps remain.

Batching Batching is a well-studied paradigm in online matching [1]. This algorithm collects requests into a batch $\mathcal{J}_b$ and assigns them via a min-cost matching when any of the following conditions hold ($t_c :=$ current time point):

1) **Age:** $x.r + t_{max} \geq t_c$ for some $x \in \mathcal{J}_{t_c,b}$.
2) **Batch Size:** Number of requests $n_b = |\mathcal{J}_{t_c,b}|$ surpasses a given threshold l_g, i.e. $n_b > l_g$.
3) **Request Frequency:** In a certain time frame t_w the number of requests $n_{t_w} = |\{x | x \in \mathcal{J}_{t_c,b} \wedge x.r + \alpha_w \cdot t_w \geq t_c\}$ surpasses a given threshold l_t, i.e. $n_{t_w} > l_t$.

The threshold values l_g and l_t are chosen to be proportional to the number of vehicles m, such that $l_g = \alpha_g \cdot m$ and $l_t = \alpha_t \cdot m$, where α_g and α_t are configurable (we use $\alpha_g = 3.5$, $\alpha_t = 2.5$). The time frame t_w used in condition (3) is based on the average of all previous consecutive requests at time point t_c: $\frac{1}{|\mathcal{J}'|-1} \sum_{i=1}^{j} x_i.r - x_{i-1}.r$, with $\mathcal{J}' = \{x_0, ..., x_j\}$. The maximum waiting time for a request is then proportional to the obtained time frame, i.e. $t_{max} = \beta_{max} \cdot t_w$, where $\beta_{max} = 2$ is also configurable.

Repositioning is incorporated via the k-Centers algorithm (see Table 1). A vehicle is repositioned if sufficient time has passed since its last repositioning ($t_{rt} = \beta_{rt} \cdot t_w$, $\beta_{rt} = 4$) *and* it has served at least one request since then, provided no batched request is currently pending.

Learning-Planning Following Xu et al. [28], this approach operates in two phases. In the offline phase, historical data is used to estimate spatio-temporal value patterns via incremental state-value updates. In the online phase, these learned values are combined with immediate rewards as edge weights in a bipartite matching, solved via the Hungarian algorithm [13, 21, 29]. We refer to [28] for full implementation details; our parameter choices are $R = 10D$ and $\gamma = 0.9$.

2.3 Repositioning Algorithms

Repositioning strategies find new locations for vehicles which are not currently serving any request. These locations should be selected to optimize a vehicle's

Table 1: Summary of repositioning strategies. For clustering-based strategies, centroids/centers are selected from received requests.

Algorithm	**Zoning Strategy**	**Scheduling Algorithm(s)**
District	Fixed-size districts.	Greedy
District-Swap	Fixed-size districts.	Greedy
k-Means [3, 6]	k request centroids, arbitrary-sized clusters.	Greedy
Balanced k-Means [19]	k request centroids, prioritises evenly sized clusters.	Greedy
k-Centers [5, 8]	k centers chosen to minimise the maximum distance between a request and a center.	Batching, Learning-Planning

ability to serve future requests, for example, by placing it closer to an area from which future requests are expected to arise. The repositioning strategies considered in this work are summarized in Table 1.

All strategies except the District strategy, which has fixed vehicle-district assignments, assign vehicles to districts or centroids or centers via a min-cost matching calculated via the Hungarian algorithm [13, 21, 29].

3 Evaluation

Setup For evaluation, eight test instances were generated, modeling multiple realistic scenarios. We varied three factors: request frequency (busy (HF) and slow hours (LF)), the presence (HS) or absence (U) of a *hot-spot* (an area with elevated request frequency, e.g., a stadium or airport), and service duration (6 or 12 hours) which defines the time horizon of the instance. To provide a lower-bound on algorithm performance w.r.t. total flow-time, the offline optimum was computed using a MILP formulation, solved with CPLEX 22.1.2.0 on an Intel Core i7-10700 CPU.

Repositioning Strategies In order to assess the impact of different repositioning strategies on algorithmic performance, we evaluated combinations of the Greedy algorithm with the repositioning strategies described in Section 2.3 across ten runs for each of the eight instances described above. Using Greedy without repositioning as a baseline, we compared combinations of the Greedy algorithm with k-Means (km), Balanced k-Means (bkm), District (Dist), and District-Swap (D-swap) repositioning strategies. Results are summarized in Table 2 and Table 3, instances encode the used setup[4].

Assignment Strategies We next compared ride-hailing algorithms that pursue strategies beyond the basic Greedy approach. Some of these algorithms, including Learning-Planning and Batching, incorporate their own custom

[4] The necessary information for reproducing the results can be found with the following Zenodo package https://doi.org/10.5281/zenodo.20457001

Table 2: Results for Total Flow Time (values divided by 10,000)

Instance	Gr.	km	bkm	Dist.	D-Swap
HFHS6	26.1	**25.8**	26.0	26.0	26.1
HFHS12	56.0	57.2	56.7	**55.9**	56.3
HFU6	18.7	18.0	18.7	**17.8**	17.8
HFU12	48.3	47.2	48.8	47.3	**47.2**
LFHS6	1.7	2.0	**1.7**	2.6	2.8
LFHS12	2.7	**2.6**	2.7	3.8	4.1
LFU6	5.6	**4.9**	5.6	5.2	5.3
LFU12	9.6	**8.2**	9.0	8.4	8.6

Table 3: Results for Sum of Driving Times (values divided by 1,000)

Instance	Gr.	km	bkm	Dist.	D-Swap
HFHS6	**107.7**	109.0	107.6	111.9	112.3
HFHS12	**217.5**	219.2	217.9	223.2	222.5
HFU6	**126.5**	132.8	126.8	135.0	132.9
HFU12	**231.1**	243.4	231.6	242.4	239.0
LFHS6	**14.1**	48.8	**14.1**	39.9	41.3
LFHS12	**22.7**	76.3	22.8	59.8	60.9
LFU6	**53.6**	91.0	53.8	76.5	71.2
LFU12	**91.4**	160.2	108.8	124.3	117.9

assignment and repositioning strategies to improve performance. Others, such as Least-Expected-Load, do not exploit repositioning at all. For reference, we also included Greedy as a baseline, as well as Greedy with k-Means repositioning. Each algorithm was executed ten times on the eight instances described above. Figure 1 presents flow time and driving time for four selected instances, illustrating the relative differences across algorithms under different demand conditions.

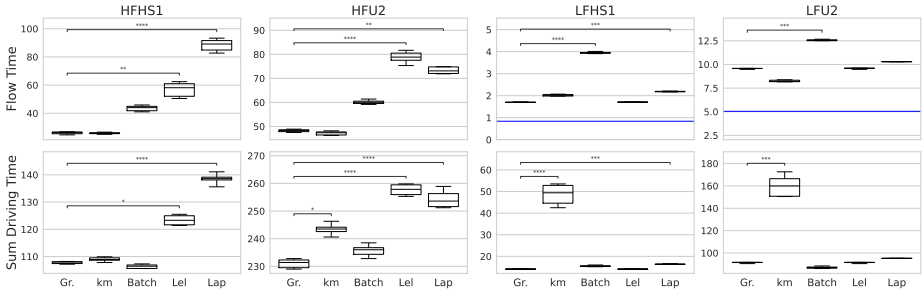

Fig. 1: Boxplots of total flow time (above) and total driving time (below) for selected instances and ride-hailing algorithms (Left to right: Greedy, Greedy + k-Means repositioning, Batching, Least Expected Load, Learning-Planning). The blue line in the low-frequency instances represents the optimal offline solution. Bars indicate significant difference from the greedy baseline, determined by Kruskal–Wallis tests followed by post-hoc Dunn's tests (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$, **** $p < 0.0001$).

Greedy-based strategies consistently outperform all other algorithms across request frequencies and distributions for total flow time. In low-frequency instances, both Batching exhibits significantly higher flow times, whereas Least-Expected-Load performs similarly to Greedy without repositioning. For high-

frequency instances, Learning-Planning and Least-Expected-Load show the worst performance.

Results for Reseach Questions For **RQ 1**, we investigate the effectiveness of repositioning strategies under different demand conditions. Our results indicate that district-based repositioning strategies which aim to evenly distribute vehicles across the service area outperform request-density based repositioning strategies. However, for non-uniform request distributions, district-based strategies increase total driving distance without reducing total flow-time. When request frequency is high, vehicles are frequently interrupted before completing repositioning, resulting in similar performance across all strategies.

RQ 2 investigates trade-offs between performance and explainability. Explainability decreases from Greedy (distance-based, fully transparent) through Least-Expected-Load (mostly interpretable) and Batching (opaque parameters) to Learning-Planning (learned rewards, difficult to explain in real-time). Yet Greedy and its variants consistently outperform all others in flow time and driving time, suggesting that *more complex algorithms do not necessarily yield better performance*. The best Greedy variants achieve flow times roughly double the offline optimal, with a smaller gap on uniform instances, indicating concentrated demand exposes a limitation of greedy assignment.

Limitations This study has several limitations. First, explainability is assessed qualitatively based on the authors' judgment, without formal quantitative measures. Second, all experimental instances were synthetically generated, which may limit the external validity of the findings. Finally, several modeling simplifications were made, e.g., indefinitely waiting users.

4 Conclusion & Future Work

We compared the performance of scheduling and repositioning algorithms for the RHP based on total flow and driving time. We implemented extensions on existing algorithms as well as the ability to combine a scheduling algorithm with different repositioning strategies. Our results show that the greedy algorithm outperforms the more complex approaches and that repositioning can be beneficial. In future work, we want to extend the scenario with time windows and the ability to share a vehicle among customers. Future work could also benefit from the development of explainability metrics and/or user surveys. Finally, we plan to evaluate our algorithms on real datasets to adapt to real-world scenarios.

Acknowledgments

GenAI has been used to support with polishing the final texts and building scripts for generating figures. All output has been reviewed by the authors.

We thank the students of the ElO4Future project at the University of Bremen for supporting with providing the overall framework for evaluation. This work has been carried out with in the project CAUSE, which has been funded by Deutsche Forschungsgemeinschaft (DFG) GRK 2972 - Project Number: 513623283.

References

1. Ackermann, C., Rieck, J.: A novel repositioning approach and analysis for dynamic ride-hailing problems. EURO Journal on Transportation and Logistics **12**, 100109 (2023)
2. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (May 1997)
3. Dehariya, V.K., Shrivastava, S.K., Jain, R.: Clustering of image data set using k-means and fuzzy k-means algorithms. In: 2010 International Conference on Computational Intelligence and Communication Networks. pp. 386–391 (2010)
4. DIMACS: Ride-hailing problem (2022)
5. Dyer, M., Frieze, A.: A simple heuristic for the p-centre problem. Operations Research Letters **3**(6), 285–288 (1985)
6. El Morr, C., Jammal, M., Ali-Hassan, H., El-Hallak, W.: K-Means, pp. 361–384. Springer International Publishing, Cham (2022)
7. Furuhata, M., Dessouky, M., Ordóñez, F., Brunet, M.E., Wang, X., Koenig, S.: Ridesharing: The state-of-the-art and future directions. Transportation Research Part B: Methodological **57**, 28–46 (2013)
8. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science **38**, 293–306 (1985)
9. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell system technical journal **45**(9), 1563–1581 (1966)
10. Herbawi, W., Weber, M.: The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. In: 2012 IEEE congress on evolutionary computation. pp. 1–8. IEEE (2012)
11. Jung, J., Jayakrishnan, R., Park, J.Y.: Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing. Computer-Aided Civil and Infrastructure Engineering **31**(4), 275–291 (2016)
12. Kamath, U., Liu, J.: Introduction to interpretability and explainability. In: Explainable artificial intelligence: An introduction to interpretable machine learning, pp. 1–26. Springer (2021)
13. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly **2**(1-2), 83–97 (1955)
14. Lee, D.H., Wang, H., Cheu, R.L., Teo, S.H.: Taxi dispatch system based on current demands and real-time traffic conditions. Transportation Research Record **1882**(1), 193–200 (2004)
15. Liao, Z.: Real-time taxi dispatching using global positioning systems. Commun. ACM **46**(5), 81–83 (May 2003)
16. Liu, Z., Zhou, C., Li, J., Wang, C., Zhang, P.: Improved genetic algorithm-based path planning for multi-vehicle pickup in smart transportation. Smart Cities **8**(4) (2025)
17. Lumbreras, S., Ciller, P.: Interpretable optimization: Why and how we should explain optimization models. Applied Sciences **15**(10) (2025)
18. Lyft: 2024 sustainability & impact report (2024)
19. de Maeyer, R., Sieranoja, S., Fränti, P.: Balanced k-means revisited. Applied Computing and Intelligence **3**(2), 145–179 (2023)
20. Manasse, M., McGeoch, L., Sleator, D.: Competitive algorithms for on-line problems. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 322–333. STOC '88, Association for Computing Machinery, New York, NY, USA (1988)

21. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics **5**(1), 32–38 (1957)
22. Parimala, M., Lopez, D., Senthilkumar, N.: A survey on density based clustering algorithms for mining large spatial databases. International Journal of Advanced Science and Technology **31**(1), 59–66 (2011)
23. Qin, Z.T., Zhu, H., Ye, J.: Reinforcement learning for ridesharing: A survey. In: 2021 IEEE international intelligent transportation systems conference (ITSC). pp. 2447–2454. IEEE (2021)
24. Tinucci, R.: Update zur elektrifizierung von uber (2025)
25. Vaze, R.: Preface, pp. xiii–xviii. Cambridge University Press (2023)
26. Veres, M., Moussa, M.: Deep learning for intelligent transportation systems: A survey of emerging trends. IEEE Transactions on Intelligent transportation systems **21**(8), 3152–3168 (2019)
27. Wen, D., Li, Y., Lau, F.C.M.: A survey of machine learning-based ride-hailing planning. IEEE Transactions on Intelligent Transportation Systems **25**(6), 4734–4753 (2024)
28. Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., Ye, J.: Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 905–913. KDD '18, Association for Computing Machinery, New York, NY, USA (2018)
29. Yadav, S.S., Lopes, P.A.C., Ilic, A., Patra, S.K.: Hungarian algorithm for subcarrier assignment problem using gpu and cuda. International Journal of Communication Systems **32**(4) (2019), e3884 dac.3884
30. Yalaoui, F., Quy Nguyen, N.: Identical machine scheduling problem with sequence-dependent setup times: Milp formulations computational study **11**(1), 15–34 (January 2021)
31. Zhan, X., Szeto, W., Shui, C., Chen, X.M.: A modified artificial bee colony algorithm for the dynamic ride-hailing sharing problem. Transportation Research Part E: Logistics and Transportation Review **150** (2021)
32. Zheng, L., Chen, L., Ye, J.: Order dispatch in price-aware ridesharing. Proceedings of the VLDB Endowment **11**(8), 853–865 (2018)