Staggered Routing in Autonomous Mobility-on-Demand Systems

Antonio Coppola[1], Gerhard Hiermann[1], Dario Paccagnan[2], and Maximilian Schiffer[3]

[1]School of Management, Technical University of Munich, Germany
antonio.coppola@tum.de, gerhard.hiermann@tum.de

[2] Department of Computing, Imperial College London, U.K.
d.paccagnan@imperial.ac.uk

[3]School of Management & Munich Data Science Institute,

Technical University of Munich, Germany
schiffer@tum.de

Abstract

In autonomous mobility-on-demand systems, effectively managing vehicle flows to mitigate induced congestion and ensure efficient operations is imperative for system performance and positive customer experience. Against this background, we study the potential of staggered routing, i.e., purposely delaying trip departures from a system perspective, in order to reduce congestion and ensure efficient operations while still meeting customer time windows. We formalize the underlying planning problem and show how to efficiently model it as a mixed integer linear program. Moreover, we present a matheuristic that allows us to efficiently solve large-scale real-world instances both in an offline full-information setting and its online rolling horizon counterpart. We conduct a numerical study for Manhattan, New York City, focusing on low- and highly-congested scenarios. Our results show that in low-congestion scenarios, staggering trip departures allows mitigating, on average, 94% of the induced congestion in a full information setting. In a rolling horizon setting, our algorithm allows us to reduce 90% of the induced congestion. In high-congestion scenarios, we observe an average reduction of 66% as the full information bound and an average reduction of 56% in our online setting. Surprisingly, we show that these reductions can be reached by shifting trip departures by a maximum of six minutes in both the low and high-congestion scenarios.

Keywords: autonomous mobility-on-demand; trip-staggering

1. Introduction

Congestion in cities is soaring. In 2022, the average US driver spent 51 hours stuck in traffic, resulting in a country-wide loss of $81 billion when accounting solely for lost working hours (Pishue 2023). Beyond financial implications, congestion is responsible for emissions of noxious gases, including carbon dioxide, which fuel climate change and cause significant health issues (Levy et al. 2010). Against this backdrop, new mobility paradigms such as autonomous mobility-on-demand (AMoD) hold the promise to mitigate the negative externalities of congestion by allowing for more efficient transport solutions. Such an AMoD system consists of a centrally controlled fleet of self-driving vehicles that provide ride-hailing services to passengers (Pavone 2015). By observing the system's complete state in real time and by leveraging centralized vehicle dispatching and rebalancing algorithms, a fleet operator may provide transportation solutions that are more efficient than passengers traveling either in their own cars or via uncoordinated ride-hailing services. However, no consensus exists yet as to whether AMoD systems will ultimately reduce congestion. Indeed, while some experts argue for reduced congestion via improved fleet control, others claim that AMoD systems may lead to an increase in demand, possibly resulting in higher transport volumes during peak hours (Oh et al. 2020).

To effectively reduce congestion, particularly in light of potentially increased demand, a number of approaches have been proposed, including ride-sharing (Ruch et al. 2020), allocation of parking space (Zhang & Guhathakurta 2017), and strategic coordination with public transport (Salazar et al. 2019), to name a few. Amongst them, *congestion pricing* and *fleet-optimal routing* have emerged as the most prominent ones both within academia and practice. To this end, congestion pricing assumes a system-wide authority that decides on a congestion fee for certain districts or roads to influence driver behavior and, consequently, the traffic flow (Pigou 1920). Depending on the tolling scheme, individuals may be incentivized to choose an alternative route (Roughgarden 2005, Paccagnan & Gairing 2023) or shift their travel to a different time (Arnott et al. 1990). Accordingly, congestion pricing approaches aim to balance traffic flow across two dimensions, *space* and *time*. Fleet-optimal routing, instead, assumes control over a fleet of vehicles and aims at reducing congestion by coordinating their routing decisions (Jahn et al. 2005, Patriksson 2015). This approach can be applied locally by a fleet operator or via central control, i.e., traffic guidance by a municipality (Levin 2017, Houshmand et al. 2019). Notably, a significant bulk of the existing literature in this domain has focused on balancing traffic flow across the *space* dimension (Jalota et al. 2023). On the contrary, fleet-optimal approaches to balance traffic across the *time* dimension remain unexplored. Nonetheless, with the advent of AMoD systems, orchestrating vehicle flows over time provides an additional degree of freedom, which may prove valuable in taming congestion. Indeed, similar approaches have already provided factual benefits in related fields, e.g., in the context of ground delay programs for air traffic control (cf. Jacquillat 2022), thus further motivating the pursuit of this direction.

Against these backdrops, we introduce the concept of *staggered routing*, where an AMoD operator delays the departure of trips over time to reduce local congestion bottlenecks so long as all trips'

drop-off deadlines are met. Specifically, we take a first step to unravel the *potential* of staggered routing in AMoD systems.

1.1. Contribution

Our work aims to inform on the impact of staggering trip departures in reducing congestion. Towards this goal, we present three key contributions. First, we introduce the problem of *staggered routing*, i.e., the problem of staggering the fleet departure times so as to minimize the resulting congestion while meeting drop-off deadlines. We then formulate this problem as a mixed integer linear program (MILP) and show that it is NP-hard. Second, we provide an effective algorithmic framework to solve the staggered routing problem at scale. Specifically, we i) show how a large number of the resulting big-M constraints can be dropped without altering the optimal solutions, and ii) develop a matheuristic that allows us to solve large-scale instances. This matheuristic bases on a construction heuristic, a problem-specific local search for intensification, while relying on our MILP to escape local optima and verify optimality. Finally, we apply our methodology to a real-world case study for the Manhattan area in New York City. Specifically, we study the impact that staggered routing has in both an offline and an online decision-making setting.

Our results show that in low-congestion (LC) scenarios, i.e., scenarios in which only the AMoD fleet induces congestion but the system without the fleet would remain uncongested, staggering trip departures allow to mitigate on average 94% of the induced congestion in a full-information setting. While not entirely reaching this upper bound, our rolling horizon approach still allows us to reduce 90% of the induced congestion in the online LC scenario. In high-congestion (HC) scenarios, i.e., scenarios in which the system already encounters exogenous congestion and the AMoD fleet induces further congestion, we observe an average congestion reduction of 66% and 56% in the full-information and rolling-horizon settings. Surprisingly, we show that these reductions can be reached by shifting trip departures of a maximum of two minutes in the LC scenario and by six minutes in the HC scenario while meeting each trip's arrival time.

1.2. Technical challenges and our approach.

Solving the staggered routing problem at scale requires tackling at least three intertwined challenges.

First, following the commonly employed approach whereby one keeps track of the congestion level on each arc at predetermined time intervals (see, e.g., time-expanded construction in Köhler et al. (2002)) is simply not possible as the size of the resulting optimization problem grows too quickly, making the solution of even moderately sized instances beyond reach. We tackle this challenge by taking a so-called Lagrangian perspective, that is, we "follow" each trip along their path and only keep track of the congestion this trip encounters at the time when it *enters* each arc on its path. This allows us to formalize an optimization problem that is significantly more compact.

Second, while the Lagrangian perspective we adopt allows to reduce the problem dimensionality, it makes it more difficult to determine the congestion level each trip encounters along the travelled arcs, leading to a set of non-linear constraints. Although easily handled through big-M reformulations,

the number of such constraints grows quickly such that the resulting continuous relaxation performs poorly for large problem instances. We resolve this challenge by showing how to significantly reduce their number without affecting optimality. We do so by carefully reformulating the problem over a multigraph.

Third, the resulting MILP can still not be solved at scale by state-of-the-art solvers. We tackle this challenge by developing a local-search algorithm that we embed within the branch and bound tree generated when solving the MILP. Specifically, whenever we find a new incumbent by solving the MILP, we stop and improve this solution through our local search before feeding it back to warmstart the MILP. While our local search follows a greedy approach based on delay severity, its development requires careful considerations as efficiently evaluating the impact that modifying one departure time has on all other trips is nontrivial.

Complimentary to this, scaling our solution to complex real-world networks introduces additional computational challenges due to the large number of road segments involved. By utilizing contraction hierarchies (cf. Geisberger et al. 2008), we reduce the network's complexity while preserving shortest path lengths. This strategy significantly improves the model's applicability and computational efficiency, making it suitable for studying staggered routing in real-world scenarios.

1.3. Related work

Our work connects with various streams of literature for the optimization of AMoD systems, an area that has recently attracted significant attention. Given the sheer volume of literature in this space, providing a comprehensive overview is beyond the scope of this work, and we refer the interested reader to Zardini et al. (2022) and Narayanan et al. (2020). Instead, we focus on two streams that are closest to our work: fleet control at the *strategic* and *operational* level.

A significant bulk of the existing literature for *strategic* fleet control focuses on time-invariant mesoscopic formulations that utilize network flow models to describe vehicle routing. Various contributions have focused on such steady-state regimes, applying either capacity constraints to model congestion (see, e.g., Rossi et al. 2018, 2020, Estandia et al. 2021) or alternatively utilizing a piecewise approximation of a Bureau of Public Roads (BPR) function (see, e.g., Salazar et al. 2019, Bahrami & Roorda 2020, Bang & Malikopoulos 2022). However, none of these paradigms is suitable for our study as they remain time-invariant and, therefore, do not allow the study of staggered routing.

Existing *operational* control approaches are instead based on fine-grained models that do incorporate a time dimension as they aim at devising an online control policy. Several works have been published in this domain, focusing on planning tasks such as vehicle-to-request dispatching, vehicle rebalancing, and request pooling. In this context, various methodological approaches have been studied, ranging from model predictive control for vehicle dispatching (see, e.g., Iglesias et al. 2018, Tsao et al. 2018, Liu & Samaranayake 2022) and request pooling (see, e.g., Alonso-Mora et al. 2017), to deep reinforcement learning for rebalancing (see, e.g., Jiao et al. 2021, Gammelli et al. 2021, Skordilis et al. 2021, Liang et al. 2021) and dispatching (see, e.g., Xu et al. 2018, Li et al.

2019, Tang et al. 2019, Zhou et al. 2019, Sadeghi Eshkevari et al. 2022, Enders et al. 2023), to optimization-augmented learning pipelines (see, e.g., Zhang et al. 2017, Jungel et al. 2023). However, these approaches focus on optimizing the routing decisions over space but do not focus on the possibility of staggering the departures over time, which is the focus of this work. Surprisingly, even studies that explicitly model congestion are scarce as existing research often bases travel times on precise forecasts or deterministic assumptions.

Finally, our work draws inspiration from the seminal work of Vickrey (1969) and its extensions (Li et al. 2020), albeit with significant differences. Indeed, while Vickrey's pioneering bottleneck model lays the theoretical groundwork to analyze commuters' departure time decisions to avoid congestion, their model has a purely *descriptive* purpose. To the contrary, our work ventures beyond descriptive modeling of traffic behaviors by actively optimizing departure times to minimize congestion.

1.4. Roadmap

The remainder of this paper is structured as follows. Section 2 outlines the problem setting for the staggered routing problem. Section 3 details our matheuristic developed for solving the staggered routing problem. In Section 4, we provide an overview of our experimental design, while in Section 5, we present the results obtained from our numerical study. Finally, we conclude in Section 6 with some remarks and future research directions.

2. Problem setting

We begin by considering an offline problem wherein an AMoD operator coordinates an autonomous vehicle fleet to serve a given set of trips, e.g., trips arising within a day or a specific time window within a city. Each trip is operated by a single vehicle and can correspond to both on-duty, i.e., customer delivery, or off-duty, i.e., rebalancing activities, of the vehicle. Importantly, vehicles operate trips along a *prespecified* route, e.g., the shortest, within the road network. Within this context, we focus on enhancing operational efficiency *after* vehicle routing decisions are taken. That is, we assume that the origin, destination, and route for each trip (equivalently, vehicle) are fixed. To do so, we aim to stagger trip departures, i.e., postpone trip departure times beyond their originally requested times, to reduce congestion caused by route overflows and thus minimize the fleet's travel time.

We model the road network within which the fleet operates as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. The nodes $\mathcal{V}$ represent trip origins, destinations, and road intersections, while the arcs $\mathcal{A}$ are the road segments connecting these nodes. Given a set of trips $\mathcal{R}$, let a quadruple $(p^r, e^r, \ell^r, \bar{\sigma}^r)$ define a trip $r \in \mathcal{R}$, with p^r denoting the sequence of arcs composing the trip's fixed route through the network, e^r being the earliest departure from the trip's origin, ℓ^r being the latest acceptable arrival at the destination, and $\bar{\sigma}^r$ being the maximum staggering time applicable to trip departure.

A trip r on arc a incurs a travel time $\tau_a^r(f_a^r)$ that is the sum of a free-flow travel time τ_a and a congestion-induced delay, which we determine by the number of AMoD trips f_a^r on arc a when trip

r starts traversing the arc.[1] We model such a congestion-induced delay as a piece-wise linear, non-decreasing, and convex function, resulting in a travel time over each arc such as that represented in Figure 1. Without loss of generality, the total travel time encountered by trip r on arc a can thus be written as

$$\tau_a^r(f_a^r) = \tau_a + \max_{k \in K}\{0, \mu_a^k \cdot (f_a^r - \hat{f}_a^k)\}, \tag{2.1}$$

where $\tau_a > 0$, $\mu_a^k > 0$ and $\hat{f}_a^k > 0$

Remark 1. Two comments are in order. First, while other congestion models exist, e.g., simpler capacity models (Estandia et al. 2021) or more accurate microscopic models (Levin et al. 2017), we believe that our approach to modeling congestion provides a good balance between accuracy and computational complexity that is appropriate for the mesoscopic nature of our study. In this respect, it is important to remark that, while in mesoscopic studies, the relation between travel time and congestion is often taken to be a fourth-order polynomial (see, e.g., US Bur. Public Roads 1964), our approach not only approximates such a function in a computationally convenient way, but it also allows to accommodate other, potentially different, relations. Second, when detecting the congestion level for a vehicle entering the arc, we ignore whether the other vehicles have just begun traversing the arc or are nearing completion. We believe that the resulting trade-off between slightly overestimating congestion and deriving a tractable model is reasonable for the scope of our study.

In this setting, a solution π is a vector with a departure time for each trip r. As we assume no vehicle idles once its trip has started, this information suffices to determine both the time as well as the number of vehicles f_a^r encountered by every vehicle when entering arc a of its route p^r. In turn, this sequence determines the travel times over all arcs and, thus, the trips' arrival times.

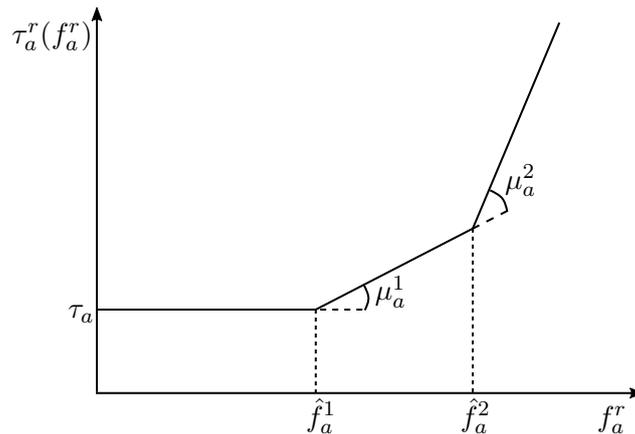We require a solution π to satisfy the following constraints, which we collect in the feasible set Π:

Figure 1: **Travel time of trip r on arc a as a function of the number of AMoD trips f_a^r.**

---

[1]   Note that our formulation easily allows to consider the background presence of non-AMoD traffic, see Section 4 for details.

i) The departure time of each trip r must occur after e^r.

ii) The departure time of each trip r can only be staggered for a maximum amount of time $\bar{\sigma}^r$.

iii) The departures in π must ensure that the arrival at the destination of each trip r occurs before ℓ^r.

In this work, we seek a solution $\pi^\star \in \Pi$ that minimizes the total fleet travel time, i.e., the sum of the travel times each trip incurs over all arcs of its path

$$\min_{\pi \in \Pi} \mathrm{Z}(\pi) = \sum_{r \in \mathcal{R}} \sum_{a \in p^r} \tau_a^r(\pi), \tag{2.2}$$

where, with slight abuse of notation, we explicitly denote the fact that the travel time τ_a^r encountered on arc a by trip r depends on the entire vector of departure times.

Finally, while (2.2) describes the offline version of the problem we will consider, one can easily transform this offline setting into an online problem by discretizing time and taking decisions on batches of trips that arrive in the system according to a point process. In our numerical studies, we will comment on the difference between the maximum potential of staggering identified by solving the offline problem and the improvement attained by solving its online counterpart.

3. Methodology

The staggered routing problem as introduced in Section 2 is NP-hard, which we show in Appendix A. This motivates us to develop an efficient matheuristic that allows to solve large-scale instances in the following. First, we introduce a MILP that allows us to obtain solutions for Problem (2.2) in Section 3.1. Towards this goal, we first introduce a basic model that uses an indicator function to quantify the aggregated flow, before we elaborate on how to model the respective indicator constraints efficiently. Second, we present a matheuristic that combines our MILP with a local search scheme to solve large real-world instances in Section 3.2. Finally, Section 3.3 adapts the previously developed algorithm to solve the online counterpart of our problem in a rolling-horizon fashion.

3.1. Mixed integer linear program

Our objective is to formulate (2.2) as a MILP. Towards this goal, note that minimizing the total travel time is equivalent to minimizing the congestion-induced delay as we have no control over the arcs' free-flow travel times τ_a. With this in mind, we perform an epigraphic reformulation of the objective function which we achieve by introducing x_a^r and requesting that $x_a^r \geq \mu_a^k \cdot (f_a^r - \hat{f}_a^k)$ for all arcs, trips, and $k \in K$. To represent the entry time of trip r on arc a on its path, we introduce the continuous variable s_a^r. Further, we let $\delta^r(a)$ denote the successor arc of arc a on route p^r, and $\underline{a}^r$, $\bar{a}^r$ denote the first and last arcs of trip r, respectively. Finally, we let $\mathcal{R}_a$ denote the set of all

trips that transit through arc a. Then, problem (2.2) can be written as:

$$\min \sum_{r \in \mathcal{R}} \sum_{a \in p^r} x_a^r, \tag{3.1a}$$

s.t.

$$x_a^r \geq \mu_a^k \cdot \left(f_a^r - \hat{f}_a^k \right), \qquad \forall k \in K, r \in \mathcal{R}, a \in p^r \tag{3.1b}$$

$$s_{a'}^r = s_a^r + \tau_a + x_a^r, \qquad \forall r \in \mathcal{R}, a \in p^r \setminus \bar{a}^r, a' = \delta^r(a) \tag{3.1c}$$

$$f_a^r = \sum_{r' \in \mathcal{R}_a \setminus \{r\}} \mathbb{I}(s_a^{r'} \leq s_a^r < s_a^{r'} + \tau_a + x_a^{r'}), \qquad \forall r \in \mathcal{R}, a \in p^r \tag{3.1d}$$

$$e^r \leq s_a^r \leq e^r + \bar{\sigma}^r, \qquad \forall r \in \mathcal{R}, a = \underline{a}^r \tag{3.1e}$$

$$s_a^r + \tau_a + x_a^r \leq \ell^r, \qquad \forall r \in \mathcal{R}, a = \bar{a}^r \tag{3.1f}$$

$$x_a^r, s_a^r, f_a^r \geq 0. \qquad \forall r \in \mathcal{R}, a \in p^r \tag{3.1g}$$

Our objective (3.1a) jointly with constraints (3.1b) minimizes the total delay over all trips and arcs. Constraints (3.1c) propagate forward the time at which trip r enters arc a along its route p^r, while constraints (3.1d) calculate the number of vehicles encountered by each trip r on arc a. This is achieved using an indicator function $\mathbb{I}(\cdot)$, which is set to one if trip r enters arc a when r' is traversing the same arc. Constraints (3.1e)-(3.1f) ensure that the start times and drop-off times meet the requirements. Constraints (3.1g) state the variable domains.

To convert the latter model into a MILP, we replace the indicator function in Constraints (3.1d) with big-M constraints that replicate its logic. To do so, we introduce binary variables $\alpha_a^{r,r'}, \beta_a^{r,r'}$ and $\gamma_a^{r,r'}$ for each pair of distinct trips (r, r') traversing the same arc, whose logic is as follows:

$$\alpha_a^{r,r'} = \begin{cases} 0, \text{ if } s_a^r < s_a^{r'} \\ 1, \text{ otherwise} \end{cases} \qquad \beta_a^{r,r'} = \begin{cases} 0, \text{ if } s_a^r \geq s_a^{r'} + \tau_a + x_a^{r'} \\ 1, \text{ otherwise} \end{cases} \qquad \gamma_a^{r,r'} = \begin{cases} 0, \text{ if } \alpha_a^{r,r'} + \beta_a^{r,r'} < 2 \\ 1, \text{ otherwise} \end{cases}$$

We enforce this logic and therefore evaluate f_a^r by replacing constraints (3.1d) with the following:

$$s_a^r - s_a^{r'} + \varepsilon \leq M_1 \cdot \alpha_a^{r,r'}, \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1h}$$

$$s_a^{r'} - s_a^r \leq M_2 \cdot (1 - \alpha_a^{r,r'}), \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1i}$$

$$s_a^{r'} + \tau_a + x_a^{r'} - s_a^r \leq M_3 \cdot \beta_a^{r,r'}, \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1j}$$

$$s_a^r - s_a^{r'} - \tau_a - x_a^{r'} + \varepsilon \leq M_4 \cdot (1 - \beta_a^{r,r'}), \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1k}$$

$$\gamma_a^{r,r'} - \alpha_a^{r,r'} - \beta_a^{r,r'} \geq -1, \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1l}$$

$$2\gamma_a^{r,r'} - \alpha_a^{r,r'} - \beta_a^{r,r'} \leq 0, \qquad \forall a \in \mathcal{A}, r, r' \in \mathcal{R}_a, r \neq r' \tag{3.1m}$$

$$f_a^r = \sum_{r' \in \mathcal{R}_a \setminus \{r\}} \gamma_a^{r,r'} \qquad \forall a \in \mathcal{A}, r \in \mathcal{R}_a \tag{3.1n}$$

where ε is a small constant and M_i are sufficiently large numbers.[2] First, observe that the resulting program is indeed a MILP. However, this formulation presents significant computational challenges due to the number of big-M constraints required to check if a given pair of vehicles travels simultaneously on the same arc. Specifically, each constraint in (3.1h)-(3.1m) needs to be repeated for all pairs of trips that transit through each given arc. This formulation results, unfortunately, in prohibitive solution times, even for modest problem sizes, as the continuous relaxation used in the solution of the MILP performs poorly. To enhance computational efficiency, we leverage the problem structure to identify and remove as many redundant big-M constraints as possible in the following.

A key observation is that not every distinct pair of trips traversing the same arc may influence each other's travel time, largely because each trip is constrained by specific time windows. However, the original problem formulation introduced in Section 2 only defines these windows for each entire trip, not for individual arcs. Based on these trip-specific time windows, our approach constructs arc-specific time windows to identify trips that, by traversing the arc simultaneously, may incur a delay.

We then leverage these bounds to build a representation of the problem on a multidigraph, in which parallel arcs represent identical road segments, but each arc is associated with either a set of potentially conflicting trips, i.e., trips that may influence each other's travel time and thus contain conflicting arcs, i.e., arcs on which trip induced congestion may arise, or trips that will not delay each other, containing only non-conflicting arcs. This differentiation allows us to apply big-M constraints selectively, focusing only on conflicting arcs and relevant trip pairs, thereby significantly reducing the number of big-M constraints appearing in the MILP. Additionally, we explore methods to merge or exclude non-conflicting arcs from the model, further minimizing its complexity and the computational overhead of the local search method discussed later in the section.

In the rest of this section, we will proceed as follows. In Section 3.1.1, we detail how to determine arc-specific time windows for each trip. In Section 3.1.2 and 3.1.3, we detail how we derive the multigraph representation introduced above. Building upon this, we are finally able to reduce the number of big-M constraints needed by our MILP model in Section 3.1.4.

3.1.1. Determining arc-dependent time windows

To distinguish trips that may conflict from trips that do not, we construct upper and lower bounds on the entry and exit times for each given arc on a trip. At a high level, the goal is to make these bounds as tight as possible while ensuring that every feasible solution satisfies them, which allows to reduce the number of big-M constraints.

Specifically, we let $\underline{e}_a^r$, $\bar{e}_a^r$ represent the earliest and latest *entry* time of trip r in arc a. Similarly, we use $\underline{\ell}_a^r$, $\bar{\ell}_a^r$ to represent earliest and latest *exit* times. Note, however, that knowledge of the earliest entry time $\underline{e}_a^r$ and latest exit time $\bar{\ell}_a^r$ for all arcs on a trip suffices to reconstruct also the

---

[2]   Note that the use of a small constant ε is necessary. Indeed, in its absence, given a pair of distinct trips (r, r') jointly traversing arc a, for s_a^r being equal to $s_a^{r'}$, setting $\alpha_a^{r,r'}$ to either zero or one would satisfy Constraints (3.1h)-(3.1i). Details on our choice of M_i can be found in Section 3.1.4.

other quantities, i.e., the latest entry and the earliest exit over all the arcs on that trip. In fact, by continuity, the latest entry on an arc $\bar{e}_a^r$ must equal the latest exit from the previous arc on that trip, while the earliest exit from an arc $\underline{\ell}_a^r$ must equal the earliest entry on the subsequent arc on that trip. This is the case for all arcs in the trip except for the very first and last arc. For the first arc $a = \underline{a}^r$, we set the latest entry as $\bar{e}_a^r = e^r + \bar{\sigma}^r$, while for the last arc $a = \bar{a}^r$ we determine the earliest exit by letting the vehicle travel in free flow through that arc so that $\underline{\ell}_a^r = \underline{e}_a^r + \tau_a$.

Thus, in the following, we focus on determining only the earliest entry $\underline{e}_a^r$ and the latest exit $\bar{\ell}_a^r$ times. The idea is to initialize these quantities by propagating the earliest entry times over subsequent arcs on a trip as if vehicles were traveling in free-flow, starting from the earliest departure e^r at the first arc. Similarly, for the latest exit times, which we propagate backward over preceding arcs on a trip, starting from the latest arrival at the last arc ℓ^r. In a second phase, we then refine and tighten these time windows by bounding the minimum and maximum congestion encountered by each trip over its arcs. We now describe both of these steps in further detail and refer to Algorithm 1 for a formal overview of the approach.

Initialization. For each trip, we set the earliest entry time $\underline{e}_a^r$ for its first arc equal to the earliest departure time of the trip. We then calculate the earliest entry times for the subsequent arc by adding the nominal travel time of the preceding arc, that is $\underline{e}_{\delta^r(a)}^r = \underline{e}_a^r + \tau_a$ for all $a \in p^r \setminus \bar{a}^r$. For each trip, we set the latest exit time $\bar{\ell}_a^r$ for its last arc equal to the latest arrival time of the trip ℓ^r. We then calculate the latest exit time for the preceding arc by subtracting the nominal travel time of the arc, that is $\bar{\ell}_a^r = \bar{\ell}_{\delta^r(a)}^r - \tau_{\delta^r(a)}$ for all $a \in p^r \setminus \bar{a}^r$.

We then construct a queue whose elements are the tuples indexed by (a, r), i.e., arc and trip, and sort such queue based on the smallest earliest entry times. We then process, in order, one element of the queue at a time to update its earliest entry and latest exit time. First, we describe the method for updating time windows. Following that, we detail how employing a priority queue results in tighter bounds compared to an unsorted approach.

Updating earliest entry times. Given an element (a, r), we now update, i.e., move forward in time, the earliest entry times on arcs subsequent to a to tighten the time windows. Towards this goal, we compute the minimum conflicts that trips r inevitably incur on arc a and denote it with $\underline{f}_a^r$. This quantity can be determined by leveraging the existing time windows as

$$\underline{f}_a^r = \sum_{r' \in \mathcal{R}_a \setminus \{r\}} \mathbb{I}(\underline{e}_a^r > \bar{e}_a^{r'} \wedge \bar{e}_a^r < \underline{\ell}_a^{r'}).$$

Algorithm 1 Compute arc-specific time windows

Input: Set of trips $\mathcal{R}$

1: $Q, \underline{e}, \bar{\ell}, \boldsymbol{x} \leftarrow$ INITIALIZE($\mathcal{R}$)
2: **while** Q is not empty **do**
3:      $(a, r) \leftarrow$ Q.POP()   ▷ Extract tuple with highest priority.
4:      $\underline{e}, \boldsymbol{x} \leftarrow$ UPDATEEARLIESTENTRYTIMES(a, r)
5:      $\bar{\ell} \leftarrow$ UPDATELATESTEXITTIMES(a, r)
6: LB $\leftarrow$ GETINITIALLOWERBOUND($\boldsymbol{x}$)
7: **return** ($\underline{e}, \bar{\ell}, $LB)

We denote $\underline{x}_a^r$ as the delay that r incurs on arc a in presence of $\underline{f}_a^r$. As the earliest entries on arcs subsequent to a are surely delayed by $\underline{x}_a^r$, we update $\underline{e}_{a'}^r \leftarrow \underline{e}_{a'}^r + \underline{x}_a^r$ for every arc $a' \in p^r$ such that $\underline{e}_{a'}^r > \underline{e}_a^r$.

Updating latest exit times. To update the latest exit time $\bar{\ell}_a^r$, we determine the maximum number of conflicts, $\bar{f}_a^r$, that a trip r can encounter on arc a. A simple counting argument reveals that this is equal to the number of other trips $r' \in \mathcal{R}_a$ whose time windows $[\underline{e}_a^{r'}, \bar{\ell}_a^{r'}]$ intersect $[\underline{e}_a^r, \bar{e}_a^r]$. We then update the latest exit time from arc a based on the latest entry in such arc and the maximum number of conflicts computed, that is, $\bar{\ell}_a^r \leftarrow \min(\bar{\ell}_a^r, \bar{e}_a^r + \tau_a^r(\bar{f}_a^r))$.

Computing an initial lower bound. We conclude Algorithm 1 by calculating an initial lower bound LB, which already allows evaluating the quality of a solution π. After finalizing the computation of arc-dependent time windows, the initial lower bound results from summing up the minimum delays that trips unavoidably incur on each arc:

$$\text{LB} = \sum_{r \in \mathcal{R}} \sum_{a \in p^r} \underline{x}_a^r.$$

The priority queue. The priority queue, organized by ascending earliest entry times $\underline{e}_a^r$, improves upon an unsorted approach to processing the sequence of the time window for two main reasons:

i) Processing pair (a, r) enables us to tighten both the earliest entry times for arcs after a in r's route, as well as the latest exit time from a. In this context, proceeding in an ascending order aligns with this forward-looking update mechanism.

ii) Ordering by $\underline{e}_a^r$ allows to tighten the maximum number of conflicts $\bar{f}_a^r$. As seen above, r may conflict with r' on a if $[\underline{e}_a^r, \bar{e}_a^r]$ overlaps with $[\underline{e}_a^{r'}, \bar{\ell}_a^{r'}]$. At the time the algorithm processes pair (a, r), the forward-looking update logic highlighted in point i) ensures $\underline{e}_a^r$, $\underline{e}_a^{r'}$ and $\bar{e}_a^r$ to be as tight as possible. Nevertheless, $\bar{\ell}_a^{r'}$ may or may not be as tight as possible, depending on the following case distinction:

   1) If $\underline{e}_a^r > \underline{e}_a^{r'}$, the algorithm processed r' before r on a. Thus, $\bar{\ell}_a^{r'}$ is as tight as possible consequent to the update logic of point i).

   2) If $\underline{e}_a^r < \underline{e}_a^{r'}$, r is processed before r'. Although $\bar{\ell}_a^{r'}$ is not as tight as possible, the determinant for overlap is $\underline{e}_a^{r'}$, which is as tight as possible.

Note that changes to a trip's earliest entry time can disrupt the queue's ordering. To address this, we maintain the queue's priorities to ensure that tuples are processed in an ascending time sequence.

Discussion. Notice that our current method tends to overestimate the worst-case delay by focusing solely on the bounds related to the specific arc under consideration. We do not dynamically adjust these bounds based on changing scenarios across consecutive arcs. This conservative approach might not capture the interplay of delays across arcs. As the scalability of the approach largely hinges on the tightness of departure bounds, addressing this limitation through a dynamic bound adaptation process presents a promising direction for future research.

Still, one can iterate the bounding procedure multiple times to achieve more precise bounds. This can be achieved by populating Q with newly computed earliest departure bounds. This iterative approach continues until no further modifications in the bounds are observed.

3.1.2. Calculating conflicting sets

After determining arc-specific time windows for all trips, we proceed by calculating conflicting sets $\mathcal{S}_a^i$, i.e., sets containing pairs of trips that can incur delay by conflicting. To do so, we define for each arc a a set $\mathcal{P}_a$ that contains all pairs of distinct trips belonging to $\mathcal{R}_a$ that might potentially conflict:

$$\mathcal{P}_a := \{(r, r') \in \mathcal{R}_a \times \mathcal{R}_a \mid r \neq r' \wedge [\underline{e}_a^r, \bar{e}_a^r] \cap [\underline{e}_a^{r'}, \bar{\ell}_a^{r'}] \neq \emptyset\}.$$

By construction, $\mathcal{P}_a$ contains all tuples representing potential conflicts between trips that utilize a. Still, within $\mathcal{P}_a$, distinct subsets of tuples built on trips that do not appear in any other subset may emerge. This separation enables us to identify subsets of trips that may conflict with each other but not with other trips utilizing arc a. To isolate these dependencies, we split $\mathcal{P}_a$ into subsets of maximal size such that each subset contains only tuples formed from trips exclusive to that particular subset, ensuring that no trip is included in more than one subset. Technically, we partition $\mathcal{P}_a$ into n sets $\mathcal{P}_a^i$ of maximal size that satisfy the following conditions:

i) The subsets $\mathcal{P}_a^i$ are pairwise disjoint and their union constitutes the entire set $\mathcal{P}_a$, i.e., $\mathcal{P}_a := \bigcup_{i=1}^{n} \mathcal{P}_a^i$ and $\mathcal{P}_a^i \cap \mathcal{P}_a^j = \varnothing$ for all $i \neq j$.

ii) For any subset $\mathcal{P}_a^i$ containing more than one pair, it holds that for each pair $(r, r') \in \mathcal{P}_a^i$, there exists another pair $(r'', r''') \in \mathcal{P}_a^i$ such that the intersection of their components is non-empty: $\{r, r'\} \cap \{r'', r'''\} \neq \varnothing$. If no such pair (r'', r''') exists, then (r, r') constitutes a singleton subset in the partition.

With $\mathcal{P}_a^i$ containing all tuples relative to a subset of trips, we can finally identify those trips that may incur a delay. According to Equation (2.1), this is the case when the number of conflicts that a trip in $\mathcal{P}_a^i$ has, i.e., the number of trips traversing an arc at the same time, exceeds the first threshold $\hat{f}_a^1$, beyond which it is not possible to travel on arc a at free-flow speed. Then, we can tighten the maximum number of conflicts $\bar{f}_a^r$ by counting the number of pairs in $\mathcal{P}_a^i$ in which a trip r appears as the first element. Accordingly, we filter our partition of $\mathcal{P}_a^i$ to derive conflicting sets $\mathcal{S}_a^i$, each containing trips that may induce delay, as:

$$\mathcal{S}_a^i := \left\{ (r, r') \in \mathcal{P}_a^i \mid \bar{f}_a^r \geq \hat{f}_a^1 \right\}.$$

3.1.3. Multigraph representation

After identifying all conflicting sets, we aim to modify our original graph $\mathcal{G}$ towards a multigraph representation. This allows us to maintain Constraints (3.1h)-(3.1n) at a minimum. Figure 2 shows an example of our graph modification that consists of two steps:
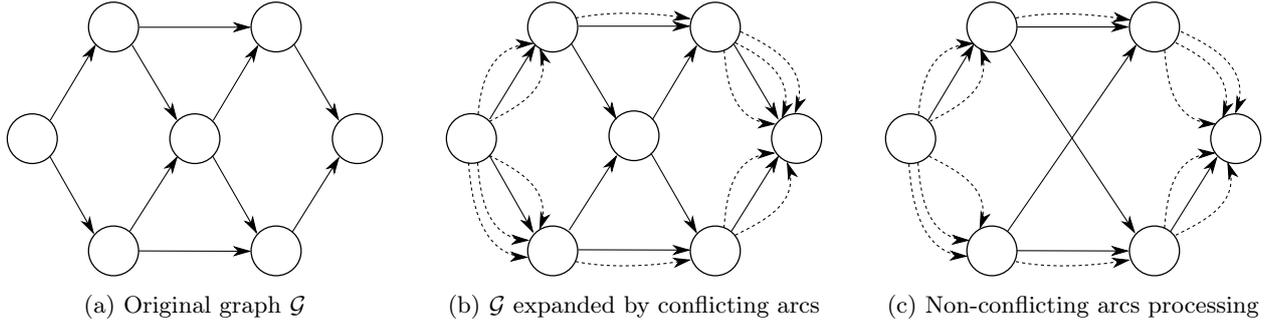
(a) Original graph $\mathcal{G}$ (b) $\mathcal{G}$ expanded by conflicting arcs   (c) Non-conflicting arcs processing

Figure 2: Schematic representation of the graph transformation process upon identification of conflicting sets: (a) initial graph structure. (b) expansion with conflicting arcs (dashed lines). (c) result of merging sequences of non-conflicting arcs and removing unused arcs. The merging process involves removing the central node and replacing its incident arcs with non-conflicting arcs with the original arcs' cumulative length.

in the first step, we transition to a multigraph by adding a parallel arc for each conflicting set; in the second step, we sparsen our graph by removing arcs that are no longer used and merging sequences of conflict-free arcs.

Step 1: For each conflicting set $\mathcal{S}_a^i$, we add a conflicting arc $\hat{a}$ as a parallel arc to its original counterpart a. This arc shares the characteristics of a, i.e., it has the same start point, endpoint, nominal travel time, and capacity. After adding a conflicting arc $\hat{a}$, we modify the routes that belong to the trips contained in $\mathcal{S}_a^i$ by replacing the original arc a with the respective conflicting arc $\hat{a}$, such that all trips that share a potential conflict traverse the same conflicting arc.

Let the set of conflicting arcs be $\hat{\mathcal{A}} \subset \mathcal{A}$. After Step 1, only trips that cannot induce delay on an arc a will still use arc $a \in \mathcal{A} \setminus \hat{\mathcal{A}}$. Accordingly, we will refer to trips that utilize only arcs $a \in \mathcal{A} \setminus \hat{\mathcal{A}}$ as non-conflicting trips and to the respective arcs as non-conflicting arcs. We remove trips from $\mathcal{R}$ that traverse only non-conflicting arcs.

Step 2: To sparsen our arc set, we aim to merge as many non-conflicting arcs as possible. We combine sequences of non-conflicting arcs in trip routes into single arcs, which we add to $\mathcal{A}$. Each new arc starts at the origin of the first arc in the sequence and ends at the destination of the last arc, and its length equals the sum of the lengths of all arcs in the sequence. We update accordingly the routes p^r in which such sequences appear. Finally, we remove arcs from $\mathcal{A}$ that do not appear in any trip's route and update $\mathcal{R}_a$ for every arc in $\mathcal{A}$ accordingly.

3.1.4. Tightened Big-M constraints

The remaining task involves updating the definition of Constraints (3.1h)-(3.1m) as follows:

$$\text{Constraints (3.1h)} - \text{(3.1m)}, \qquad \forall a \in \hat{\mathcal{A}}, (r, r') \in \mathcal{P}_a$$

thereby applying these constraints exclusively to conflicting arcs, $\hat{\mathcal{A}}$, and to each pair of trips that might conflict and potentially cause delays.

To minimize the M_i constants in the remaining big-M constraints, we leverage the time windows obtained with Algorithm 1. For any pair of trips (r, r') within $\mathcal{P}_a$, where a belongs to the set of

conflicting arcs $\hat{\mathcal{A}}$, we extract the following relationships from Constraints $(3.1\text{h}) - (3.1\text{k})$:

$$M_1 \geq \max(s_a^r - s_a^{r'} + \varepsilon) = \max(s_a^r) - \min(s_a^{r'}) + \varepsilon = \bar{e}_a^r - \underline{e}_a^{r'} + \varepsilon, \qquad \forall a \in \hat{\mathcal{A}}, (r, r') \in \mathcal{P}_a$$

$$M_2 \geq \max(s_a^{r'} - s_a^r) = \max(s_a^{r'}) - \min(s_a^r) = \bar{e}_a^{r'} - \underline{e}_a^r, \qquad \forall a \in \hat{\mathcal{A}}, (r, r') \in \mathcal{P}_a$$

$$M_3 \geq \max(s_a^{r'} + \tau_a + x_a^{r'} - s_a^r)$$
$$= \max(s_a^{r'} + \tau_a + x_a^{r'}) - \min(s_a^r) = \bar{\ell}_a^{r'} - \underline{e}_a^r, \qquad \forall a \in \hat{\mathcal{A}}, (r, r') \in \mathcal{P}_a$$

$$M_4 \geq \max(s_a^r + \tau_a + x_a^r - s_a^{r'} + \varepsilon)$$
$$= \max(s_a^r + \tau_a + x_a^r) - \min(s_a^{r'}) + \varepsilon = \bar{\ell}_a^r - \underline{e}_a^{r'} + \varepsilon. \qquad \forall a \in \hat{\mathcal{A}}, (r, r') \in \mathcal{P}_a$$

3.2. Matheuristic

We develop a matheuristic incorporating our MILP into a local search scheme to find good solutions for large real-world instances. While this matheuristic is not guaranteed to find an optimal solution, it leverages the MILP from Section 3.1 and can thus detect an optimal solution if encountered. Algorithm 2 shows a high-level pseudo-code of our matheuristic. After creating an initial solution π that is agnostic to staggering, we apply a local search to improve it by staggering its trip departures. We then aim to further improve π by iterating between our MILP and our local search: whenever we find a new incumbent by solving the MILP, we stop and improve this incumbent with our local search. Once the local search terminates, we feed its final solution back to warmstart the MILP. Our matheuristic stops either when the MILP certifies optimality of π, i.e., $Z(\pi) = \text{LB}$, or after a maximum time η elapsed. Note that even if the MILP cannot certify optimality within the computational time limit, our matheuristic has the advantage of providing a lower bound to indicate the quality of π. In the remainder of this section, we detail our construction routine to obtain an initial solution (Section 3.2.1) before we describe our local search (Section 3.2.2).

3.2.1. Constructing initial solutions

To construct an initial solution π, we set its departure time variables s_a^r, delay variables x_a^r and conflict counting variables f_a^r to infinity. We then sort the trip-specific earliest departures e^r of every trip r into a priority queue Q and initialize an empty set $\mathcal{T}_a$ for each arc a to track when each trip completes the traversal of a. Then, we construct an initial solution iteratively:

i) We start by extracting the departure time s_a^r with the highest priority from Q.

Algorithm 2 Matheuristic

Input earliest departure times $\boldsymbol{e}$, initial lower bound LB, time limit η

1: $\pi \leftarrow \textsc{constructSolution}(\boldsymbol{e})$
2: $\pi \leftarrow \textsc{localSearch}(\pi)$
3: **while** time elapsed $\leq \eta$ **do**
4:      LB$, \pi \leftarrow \textsc{solveMILP}(\text{LB}, \pi)$
5:      **if** $Z(\pi) = \text{LB}$ **then**
6:          **return** π
7:      $\pi \leftarrow \textsc{localSearch}(\pi)$
8: **return** π

---

ii) We then set $f_a^r = |\{t \in \mathcal{T}_a \,|\, s_a^r < t\}|$, that is the number of trips that have not completed their traversal of arc a by time s_a^r, and update $x_a^r = \max_{k \in K}\{0, \mu_a^k \cdot (f_a^r - \hat{f}_a^k)\}$.

iii) Finally, we add the time in which the arc traversal is completed, which is $s_a^r + x_a^r + \tau_a$, to $\mathcal{T}_a$ by setting $\mathcal{T}_a = \mathcal{T}_a \cup (s_a^r + x_a^r + \tau_a)$. If a is not the last arc of route p^r, we insert $s_a^r + x_a^r + \tau_a$ into Q.

We repeat these steps until Q is empty.

Note that once we obtained a feasible solution, it is possible to compute values for the MILP's binary variables $\alpha_a^{r,r'}, \beta_a^{r,r'}$ and $\gamma_a^{r,r'}$ with a support function checking their activation conditions as introduced in Section 3.1.

3.2.2. Local search

Our local search aims to iteratively remove conflicts based on a priority queue, aiming to first resolve conflicts that induce the largest delays. Let the tuple (r, r', a) index the conflict that r has with r' on arc a. Given a conflict (r, r', a), we resolve it by either shifting r backward or r' forward in time without violating the trips' time windows. In this context, we compute additional quantities for each conflict, utilizing the relations shown in Figure 3. Specifically, consider a conflict (r, r', a) with $s_a^r < s_a^{r'}$. We then compute the time overlap Δ between r and r', which denotes how much staggering we need to apply to resolve the conflict. Formally:

$$\Delta = s_a^r + \tau_a + x_a^r - s_a^{r'} + \varepsilon.$$

Clearly, $\Delta = \phi = \rho$ indicates the required backward shift ϕ for r and the required forward shift ρ for r' to resolve the conflict by staggering just one of the two trips. However, shifting a trip by Δ may cause time window violations. Therefore, we limit ϕ and ρ to their feasible counterparts, ϕ^{f} and ρ^{f}. These values represent the maximum backward and forward time shifts applicable without violating the time window of each trip. Formally, ϕ^{f} reads:

$$\phi^{\text{f}} = \min(s^r - e^r, \Delta),$$

(a) Conflict metrics
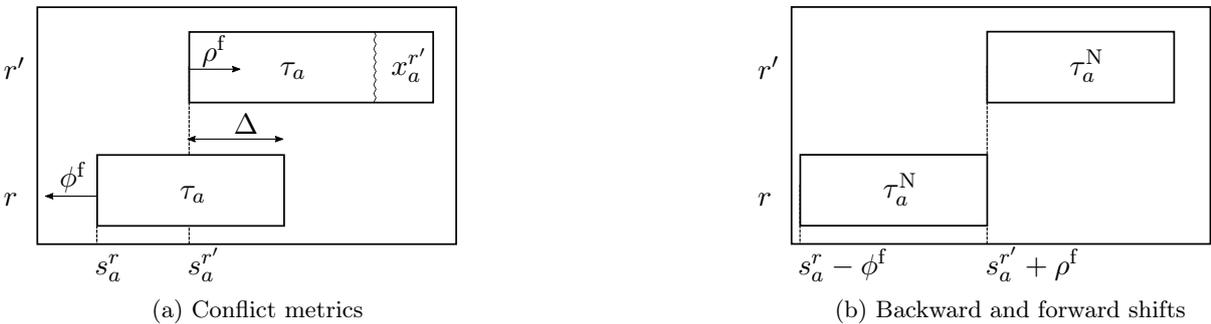
(b) Backward and forward shifts

Figure 3: Figure (a) illustrates the quantities computed in the local search for resolving a conflict (r, r', a), where r' incurs delay x_a^r. It illustrates the time overlap Δ, the feasible backward shift ϕ^{f}, and forward shift ρ^{f}. Figure (b) depicts trips r and r' post the application of these shifts, resulting in conflict resolution and the consequent elimination of the arc delay.

while ρ^{f} reads:

$$\rho^{\mathrm{f}} = \min(\bar{\sigma}^{r'} - (s^{r'} - e^{r'}), \Delta),$$

with s^r denoting the starting time of a trip r.

Based on these quantities, Algorithm 3 provides the pseudocode for our local search: given an initial solution π, we sort conflicts into a priority queue Υ, thus ordering conflicts based on their induced delay in decreasing order (l.1). We then process Υ until it is empty (l.2). In each iteration, we pop the first conflict from Υ (l. 3) and compute ρ^{f}, ϕ^{f}, and Δ as defined above (l. 4). Only if the sum of ρ^{f} and ϕ^{f} allows to compensate Δ (l.5), we resolve the conflict (l.6). If this leads to an improved solution (l.7), we update our solution (l.8) as well as our priority queue (l.9) since new conflicts may arise. If the conflict cannot be resolved, we directly proceed with the next item in Υ without updating π or Υ. In the remainder of this section, we elaborate on constructing and evaluating a neighborhood to create a new solution that resolves a conflict.

While our local search follows a rather simple greedy repair selection based on conflict severity, its complexity lies in efficiently evaluating a neighborhood, i.e., computing the impact of resolving one conflict on all other trips' departure times. In the best case, resolving a conflict induces changes only in arc entry times for the conflicting trips, while in the worst case, resolving a conflict can lead to a very large cascade of changes in other trip departures. Algorithm 4 shows how we efficiently update our solution when resolving a conflict: we first apply a minimum staggering to the departures of r and r'. When staggering a trip departure time to resolve a conflict, we insert a corresponding tuple (r, a), consisting of the trip's index r and its route's first arc a into a priority queue Q, which we use to maintain trips that require recalculating arc entry times due to the applied staggering. We sort the tuples in Q in ascending order based on their start times s_a^r (l.1). After resolving the initial conflict, we start processing trips in Q (l.3) by checking whether the trip's updated departure time s_a^r creates a change in the arc entry times of another trip r' (l.4). In this case, we insert r' into Q. If inserting r' into Q invalidates a further propagation of arc entry times for trip r – i.e., if $s_a^{r'} < s_a^r$, which disrupts the order of our priority queue – we reinsert s_a^r into Q and restart evaluating Q (l. 5&6). If s_a^r does not induce changes in other trips, we recursively propagate the change in trip r's arc entry times on subsequent arcs until a change arises (l.7–l.13).

Algorithm 3 Local Search

    Input: solution π
1: $\Upsilon \leftarrow \mathrm{IDENTIFYCONFLICTS}(\pi)$
2: **while** Υ is not empty **do**
3:     $(r, r', a) \leftarrow \Upsilon.\mathrm{POP}()$
4:     $\rho^{\mathrm{f}}, \phi^{\mathrm{f}}, \Delta \leftarrow \mathrm{ANALYZECONFLICT}(r, r', a)$
5:     **if** $\rho^{\mathrm{f}} + \phi^{\mathrm{f}} \geq \Delta$ **then**
6:         $\bar{\pi} \leftarrow \mathrm{RESOLVECONFLICT}(\pi, r, r', \rho^{\mathrm{f}}, \Delta)$
7:         **if** $\mathrm{Z}(\bar{\pi}) < \mathrm{Z}(\pi)$ **then**
8:             $\pi \leftarrow \bar{\pi}$
9:             $\Upsilon \leftarrow \mathrm{IDENTIFYCONFLICTS}(\bar{\pi})$
10: **return** π

Algorithm 4 RESOLVECONFLICT

    Input: solution π, trips r and r', feasible forward shift ρ^{f}, time overlap Δ

1: $Q \leftarrow$ STAGGERDEPARTURES$(r, r', \rho^{\mathrm{f}}, \Delta)$
2: **while** Q is not empty **do**
3:     $(r, a) \leftarrow$ Q.POP()
4:     $r', x_a^r \leftarrow$ UPDATESOLUTION(r, a)
5:     **if** $r' \neq$ FALSE **then**
6:         $Q \leftarrow$ INSERTTRIPS(r, r', a)
7:     **else**
8:         **while** $r' =$ FALSE **do**
9:             $s_{\delta^r(a)}^r \leftarrow s_a^r + \tau_a + x_a^r$
10:        $a \leftarrow \delta^r(a)$
11:        $r', x_a^r \leftarrow$ UPDATESOLUTION(r, a)
12:        **if** $r' \neq$ FALSE **then**
13:            $Q \leftarrow$ INSERTTRIPS(r, r', a)
14: **return** π

Our overall update ends once Q is empty and all arc entry times have been correctly adjusted. To conclude this section, we detail how we stagger departure times, check for solution changes, and maintain the priority queue.

STAGGERDEPARTURES$(r, r', \rho^{\mathrm{f}}, \Delta)$: To resolve a conflict, we need to shift either r or r' or both in time by Δ. Clearly, it is desirable to shift only one of these trips in time to keep the number of newly arising conflicts and entry time changes at a minimum. Accordingly, we proceed as follows to resolve a conflict:

1. If $\rho^{\mathrm{f}} \geq \Delta$ we adjust $s_a^r \leftarrow s_a^r + \Delta$ and push (r, a) into Q.

2. If a conflict cannot be resolved by shifting only trip r in time, i.e., $\rho^{\mathrm{f}} < \Delta$, we stagger both trips such that $s_a^r \leftarrow s_a^r + \rho^{\mathrm{f}}$ and $s_a^{r'} \leftarrow s_a^{r'} - (\Delta - \rho^{\mathrm{f}})$ and push (r, a) as well as (r', a) into Q.

UPDATESOLUTION(r, a): After staggering trips to resolve a conflict, we need to update the arc entry times of these trips as well as the arc entry times of all other trips that might be affected by the change. To do so, we first compute the number of conflicts f_a^r that arise for the respective trip and arc as:

$$f_a^r = \sum_{r' \in \mathcal{R}_a \setminus \{r\}} \mathbb{I}(s_a^{r'} \leq s_a^r < s_a^{r'} + \tau_a + x_a^{r'}),$$

which subsequently allows us to update the corresponding delay as:

$$x_a^r = \max_{k \in K}\{0, \mu_a^k \cdot (f_a^r - \hat{f}_a^k)\}.$$

We can then check whether a new conflict arises or an existing conflict gets resolved between r and some other trip r' due to the updated entry time of r. If either is the case, we return the index of trip r'; otherwise, we return a FALSE flag to indicate that no new conflict was induced or resolved by staggering r.

INSERTTRIPS(r, r', a): If we detect a new conflict between r and r', we add (r', a) to Q. Here, we need to account for the following two cases: if $s_a^r < s_a^{r'}$, changes induced in r' do not affect

r. It suffices to push (r', a) into Q. However, if $s_a^{r'} < s_a^r$, changes in r' affect r, which invalidates preceding updates made to r. In this case, we additionally push (r, a) back to Q to ensure valid arc entry time updates upon the algorithm's termination.

3.3. Online algorithm

While we discussed our matheuristic for a static problem setting, we can straightforwardly apply it to an online problem setting where trips enter the system over time, e.g., following a point process. In this context, we focus on batched decision-making, which allows us to apply our matheuristic in a rolling-horizon fashion.

Specifically, we divide the static time horizon into $n \in \{1, \ldots, N\}$ distinct epochs, each spanning a fixed time interval of size Δt. Before a new epoch n starts, we then take a staggering decision for all requests that entered the system after the last decision taken, collected in a set $\mathcal{R}^n$, formally:

$$\mathcal{R}^n := \{r \in \mathcal{R} \,|\, (n - 1) \cdot \Delta t \leq e^r < n \cdot \Delta t\}. \qquad \forall n \in \{1, \ldots, N\}$$

We can then compute a solution π^n for the respective batch of requests by running our matheuristic with a suitable time limit η.

Algorithm 5 shows how we apply our matheuristic in such a rolling horizon setting: we iterate over all epochs (l.2), each time focusing on the respective request set $\mathcal{R}^n$ (l.3), and calculating the relative earliest departure times vector $\mathbf{e}^n$, and a lower bound for epoch n as described in Sections 3.1.1-3.1.3 (l.4).

For each epoch, we compute a solution π^n by running our matheuristic (l.5). Afterward, we can fix this solution for all trips in $\mathcal{R}^n$. Still, we need to identify trips that span multiple epochs to identify all conflicts in future epochs during preprocessing. To do so, we collect such trips in a set $\mathcal{R}^T$, formally:

$$\mathcal{R}^T := \{r \in \mathcal{R}^n \,|\, s_{a'}^r > n \cdot \Delta t\},$$

with a' being the last arc of route p^r. Note that after the respective adjustments, trips in $\mathcal{R}^T$ can also conflict with trips being in $\mathcal{R}^n$ but not in $\mathcal{R}^T$. Maintaining those trips by incorporating them

---

Algorithm 5 Online algorithm

> **Input** Set of trips $\mathcal{R}$, number of epochs N, time limit η
1: $\mathcal{R}^T \leftarrow \varnothing; \mathcal{R}^D \leftarrow \varnothing$
2: **for** $n \in \{1, \ldots, N\}$ **do**
3:      $\mathcal{R}^n \leftarrow \textsc{getTripsEpoch}(\mathcal{R})$
4:      $\boldsymbol{e}^n, \text{LB} \leftarrow \textsc{preprocess}(\mathcal{R}^n, \mathcal{R}^T, \mathcal{R}^D)$
5:      $\pi^n \leftarrow \textsc{matheuristic}(\boldsymbol{e}^n, \text{LB}, \eta)$
6:      **if** $n < N$ **then**
7:          $\mathcal{R}^T \leftarrow \textsc{getTransferTrips}(\pi^n)$
8:          $\mathcal{R}^D \leftarrow \textsc{getDummyTrips}(\pi^n, \mathcal{R}^T)$
9: $\boldsymbol{\pi} \leftarrow (\pi^n)_{n \in \{1, \ldots, N\}}$ $\triangleright$ Vector of epoch solutions
10: $\pi \leftarrow \textsc{getSolution}(\boldsymbol{\pi})$ $\triangleright$ Solution for every trip in $\mathcal{R}$
11: **return** π

---

directly in $\mathcal{R}^T$ poses a computational overhead in the matheuristic, as it necessitates the inclusion of additional conflicting trips to accurately calculate their arc entry times.

To mitigate this overhead, we identify all conflicts (r, r', a) in π^n, where r is part of $\mathcal{R}^T$, r' is not, and a is included in p^r. For each identified conflict, we generate an artificial trip r' with $p^{r'}$ consisting solely of the arc a, $e^{r'}$ being the arc entry time of the conflicting trip $s_a^{r'}$, $\ell^{r'}$ matching the time in which the conflicting trip completes the arc traversal, and $\bar{\sigma}^{r'}$ being zero to preclude any staggering time decisions by the algorithm. We maintain these dummy trips in a separate set $\mathcal{R}^D$ and take them into account in future epochs.

4. Design of Experiments

We used Python 3.9.10 and Gurobi 10.0.1 to build and solve the MILP model. To implement our matheuristic, we used C++ to implement the local search and integrated it into the Python framework via pybind11 (Jakob et al. 2017). We performed all experiments on an Intel(R) i9-9900 CPU, 3.1 GHz, with 56 GB of RAM.

To create a realistic case study, we use the New York taxi data set (NYC Taxi & Limousine Commission 2015) and focus on the area of Manhattan in New York City.

Road network. We extract the Manhattan road network from the OpenStreetMap dataset to obtain a network with 7782 arcs and 3213 nodes (OpenStreetMap 2024). Studying staggered routing on this raw network remains challenging as it requires checking conflicts for a large number of arcs. To decrease the number of arcs while preserving the length of the shortest paths within the network, we use contraction hierarchies (cf. Geisberger et al. 2008), a well-known reduction technique for online routing, to contract the road network graph. We employ iterative node contractions that merge arcs while preserving the shortest routes between the remaining nodes. The efficiency of this contraction hinges on the order in which nodes are processed. We first contract nodes with the smallest *edge difference* - a metric capturing the difference between the number of shortcuts added at the time of node contraction and the number of incident arcs. Here, we adopt the *lazy update* approach of Geisberger et al. (2008) to reduce edge difference recalculations and enhance efficiency.

Trip data. We obtain trips $\mathcal{R}$ from the NYC taxi dataset, collected in January 2015 (NYC Taxi & Limousine Commission 2015), which contains taxi trips' origins, destinations, and departure times. We determine trip routes by selecting the shortest route connecting the nearest network nodes to the origin and destination with the minimum number of arcs. Trip start times e^r correspond to the taxi ride's start times provided in the dataset. We assign trip deadlines ℓ^r as the time of arrival at the destination in the uncontrolled scenario plus a relative quota equal to 25% of the route's nominal travel time plus a fixed tolerance of 30 seconds for each trip. We calculate the maximum allowable departure time shift $\bar{\sigma}^r$ as a percentage ς^{MAX} of the total nominal travel time required to traverse the route p^r. Unless specified differently, we set this percentage to 10%.

Parameterization. We set the nominal travel times τ_a assuming vehicles traveling at 20 kilometers per hour, which approximates the average traffic speed recorded in New York City downtown

in 2022 (Pishue 2023). As we focus on the impact and control of a large AMoD fleet, we assume a baseload of traffic that is out of the operator's control and adjust arc capacities c_a accordingly to account for potential congestion induced by the AMoD fleet. Specifically, we consider two scenarios: first, a LC scenario, which reflects a setting in which the exogenous traffic in the system is uncongested, and the AMoD fleet may induce congestion in case of inefficient operations. Second, a HC scenario in which the exogenous traffic already encounters congestion that might be amplified by the AMoD fleet operations. To create these scenarios, we calibrate the residual capacities to accommodate one AMoD trip every 15 seconds for the LC scenario and 30 seconds for the HC scenario. This is done by setting, respectively, arc capacities as $c_a = \tau_a/15$ and $c_a = \tau_a/30$ and rounding the resulting value to its nearest integer. To calibrate the arc travel time function, we considered a single non-flat piece, a maximum flow at which it is possible to travel the arc under free-flow conditions equal to $\hat{f}_a = c_a$, and a slope equal to $\mu_a = \frac{0.5 \cdot \tau_a}{c_a}$.

Instances. To create our instances, we sampled 5000 trips between four and five p.m. over 31 days, sorting them by start time. To gauge their complexity, we compute the uncontrolled solution for each instance both in the LC and HC scenarios and calculate its corresponding total delay $\bar{Z}$, the number of conflicting arcs $|\hat{\mathcal{A}}|$, and the maximum number of trips traversing the same conflicting arc $|\mathcal{R}_{\hat{a}}|^{\mathrm{MAX}}$. Table 1 summarizes the characteristics of these instances, detailing their minimum, maximum, and average values. The total fleet delay in the HC scenario is roughly ten times greater than in the LC scenario. Additionally, there is a consistent rise in the number of conflicting arcs between scenarios, with an increase of approximately 1000 arcs. However, the difference in the peak number of trips per conflicting arc is relatively modest between scenarios, averaging an increase of eight trips.

5. Results

In the following, we discuss the findings of our numerical studies. First, we analyze our matheuristic performance for both an idealized offline setting as well as a myopic online setting in Section 5.1. We then proceed with a fine-grained analysis in Section 5.2 before discussing the trade-off between congestion-related delay reduction and trip departure time shifts in Section 5.3.

Table 1: Minimum, maximum, and average values of the total delay in the uncontrolled solution $\bar{Z}$, the number of conflicting arcs $|\hat{\mathcal{A}}|$, and the maximum number of trips traversing the same conflicting arc $|\mathcal{R}_{\hat{a}}|^{\mathrm{MAX}}$ for low congestion (LC) and high congestion (HC) instances.

	LC			HC		
	Min	Max	Avg	Min	Max	Avg
$\bar{Z}$ [min]	54	133	82	348	1005	577
$|\hat{\mathcal{A}}|$	1400	2489	1971	2496	3660	3165
$|\mathcal{R}_{\hat{a}}|^{\mathrm{MAX}}$	52	161	102	76	161	110

5.1. Algorithmic performance

Figure 4 shows the distribution of the delay reduction Θ obtained when using our matheuristic compared to the uncontrolled setting for both the LC and HC scenario. We report the delay reduction obtained when using our matheuristic in a full-information offline scenario (OFF) to obtain an upper bound on the staggered routing performance. Additionally, we report the delay reduction obtained when using our matheuristic in a rolling-horizon fashion in an online scenario (ON) to quantify the benefit of staggering in a real-world setting. For further technical discussions, we refer the readers to Appendix C. We used a time limit of two hours to compute the offline solutions while limiting the computation of the rolling horizon solutions to six minutes, which equals the length of an epoch.

Focusing on the offline delay reduction, we observe a median delay reduction of 99% and a minimum reduction of 80% for the LC scenario, which shows that our matheuristic allows us to mitigate at minimum 80% of the fleet-induced congestion in an initially uncongested system. For the HC scenario, we observe a delay reduction within 50% and 80%, with a median reduction of 66%, which shows that our matheuristic effectively reduces fleet-induced congestion even in a setting where the system is already congested by exogenous traffic.

Result 1 (offline performance). In a full information setting, staggered routing allows for an average delay reduction of 94% and 66%, respectively, for an LC and HC scenario.

Focusing on the improvement potential in an online decision-making setting, we observe an average delay improvement of 90% and 56% for the LC and HC scenario when applying our matheuristic in a naive rolling horizon fashion. The observed performance difference of ten percentage points
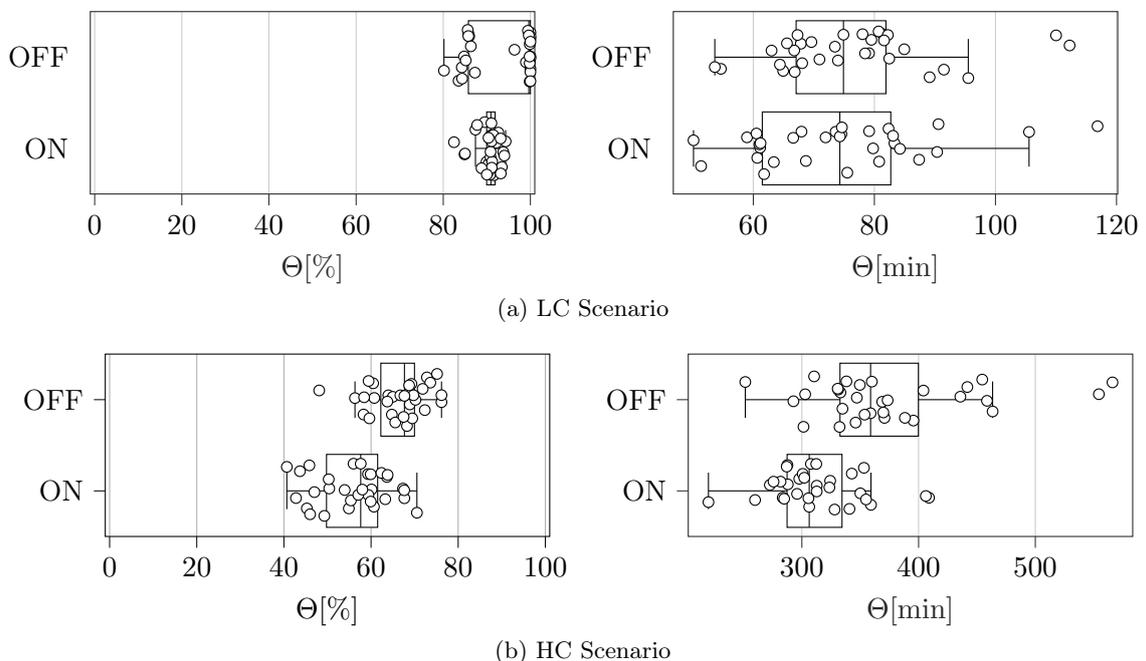

(a) LC Scenario

(b) HC Scenario

Figure 4: Distribution of the relative and absolute delay reduction Θ over all instances for a full information offline scenario (OFF) and a rolling-horizon online scenario (ON).

between our naive online algorithm and the full information bound remains surprisingly small in both LC and HC scenarios and points to the fact that even a temporally local staggering of trips allows for the reduction of a significant share of congestion-related delay. At the same time, it points to the potential of developing a more sophisticated prescriptive online algorithm that might obtain a performance close to the full information bound.

Result 2 (online performance). In a rolling horizon setting, staggered routing allows for an average delay reduction of 90% and 56%, respectively, for an LC and HC scenario.

5.2. Fine-grained analysis.

To deepen our understanding of the impact of staggered routing, Figure 5 shows a distribution over arcs that exhibit a certain cumulative delay within the LC or HC scenario. We detail this distribution for the arc-based delay resulting in an uncontrolled setting (UNC) in which no staggering is applied, when solving the offline full information setting (OFF), and when solving an online rolling horizon setting (ON). Note that we exclude arcs that do not exhibit any congestion in either of these settings to focus the distributions on congestion effects. Figure 6 complements this analysis by showing the spatial distribution of trips incurring delay on certain arcs.

Clearly, the HC scenario exhibits a significantly higher cumulative delay on arcs as the existing exogenous congestion in the system reinforces the congestion induced by the AMoD fleet. However, we observe in both the LC and the HC scenarios that staggering trip departures allow us to mitigate a significant share of the congestion induced by the AMoD fleet. Specifically, our matheuristic removes congestion completely for approximately 6500 and 10000 arcs in the LC and HC scenarios,
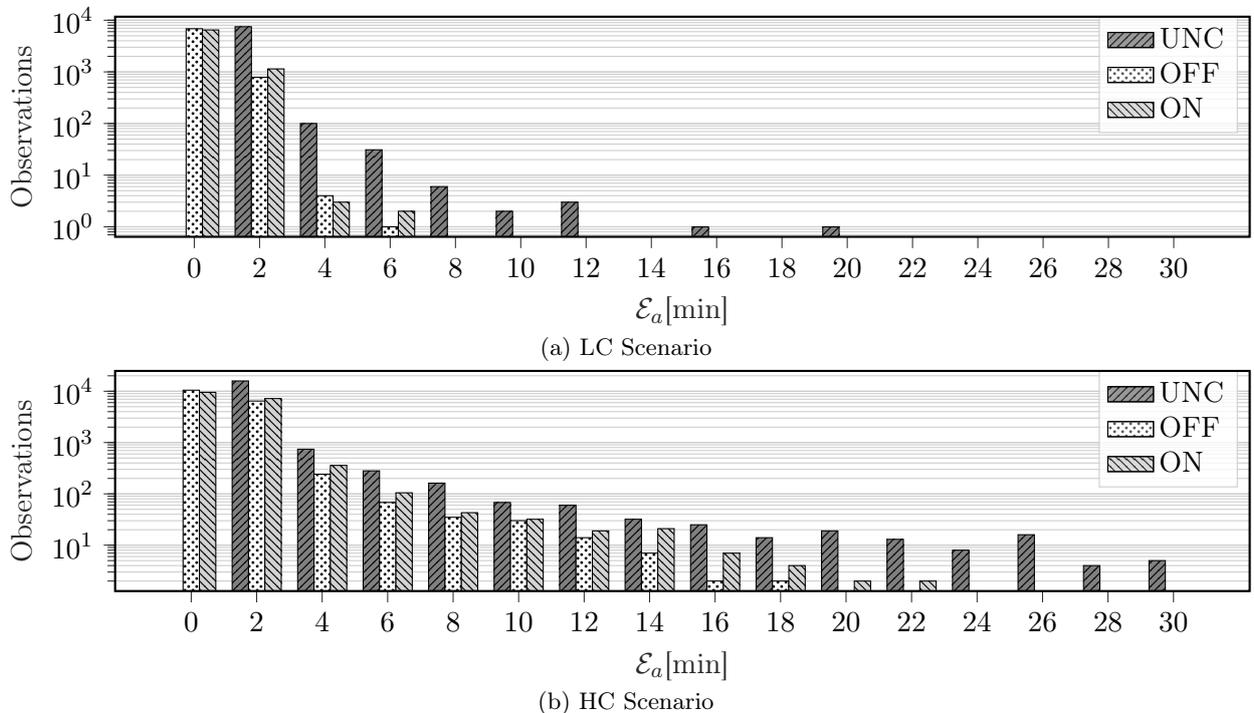

(a) LC Scenario

(b) HC Scenario

Figure 5: Distribution of the arc-based cumulative delay ($\mathcal{E}_a$) over all instances for the uncontrolled (UNC), offline full information (OFF), and online rolling horizon (ON) settings.

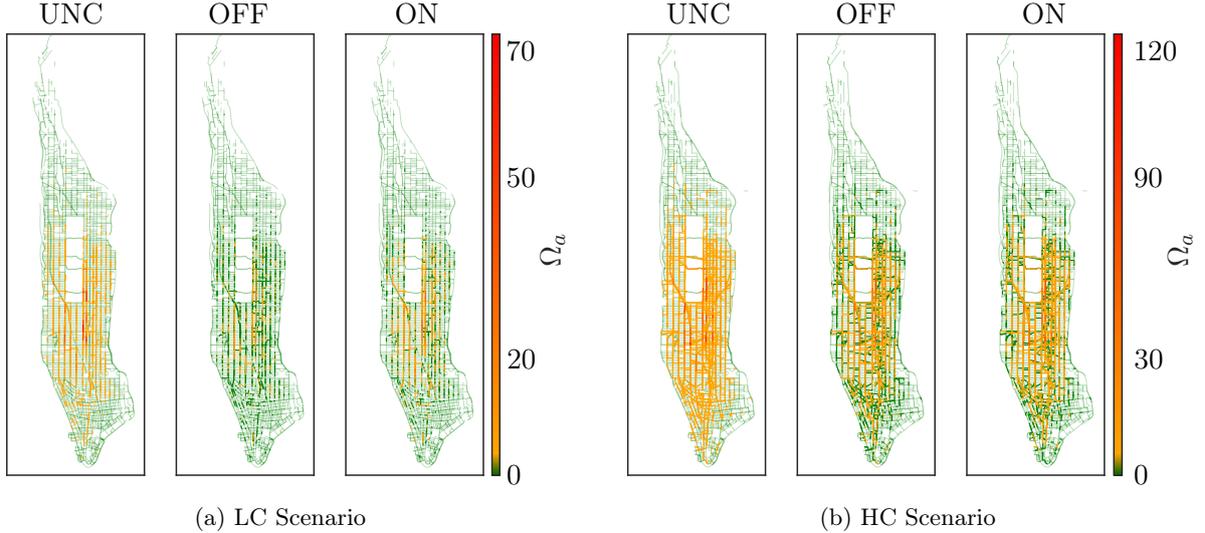(a) LC Scenario

(b) HC Scenario

Figure 6: Spatial distribution of the number of trips per arc that experience congestion (Ω_a) in the uncontrolled (UNC), offline (OFF), and online (ON) settings for both the LC and HC scenarios.

independent of whether we apply it in an offline or online setting. Interestingly, the differences between the offline and online settings arise for the remaining mildly congested arcs for which the online algorithm does not succeed in anticipating temporal interdependencies.

Figure 6 shows the spatial distribution of the number of trips per arc that experience delay in the respective settings. As can be seen, only a few local congestion effects remain in the LC scenario when comparing the offline and online solutions after staggering to the uncontrolled setting. Contrarily, some main roads exhibit a decreased but still significant congestion in the HC scenario.

Result 3 (congestion reduction). Staggered routing increases the number of uncongested arcs, respectively, for LC and HC scenarios by 6841 and 10479 when comparing the full information solutions and 6483 and 9524 when comparing the naive online solutions against the uncontrolled solutions.

5.3. Sensitivity analyses

Clearly, the improvement potential of staggered routing depends on the amount of staggering allowed for each trip and constitutes a trade-off between passenger convenience, i.e., modifying a customer's departure time up to a certain threshold and obtaining a system optimum from a total congestion perspective. For this analysis, we focus on the full information setting for one single instance to discuss the maximum improvement potential of the respective trade-off. To analyze this trade-off, we ignore our basic setting and vary the maximum amount a trip's departure can be staggered within $\varsigma^{\mathrm{MAX}} \in \{0\%, \ldots, 25\%\}$ of its nominal travel time with a step width of 2.5 percentage points. Here, $\varsigma^{\mathrm{MAX}} = 0\%$ equals the uncontrolled setting discussed in the preceding subsections.

Figure 7 highlights the resulting trade-off by showing the dependency of the total accumulated delay $Z(\pi)$ and the distribution of realized trip departure shifts σ^r in dependence of the maximum
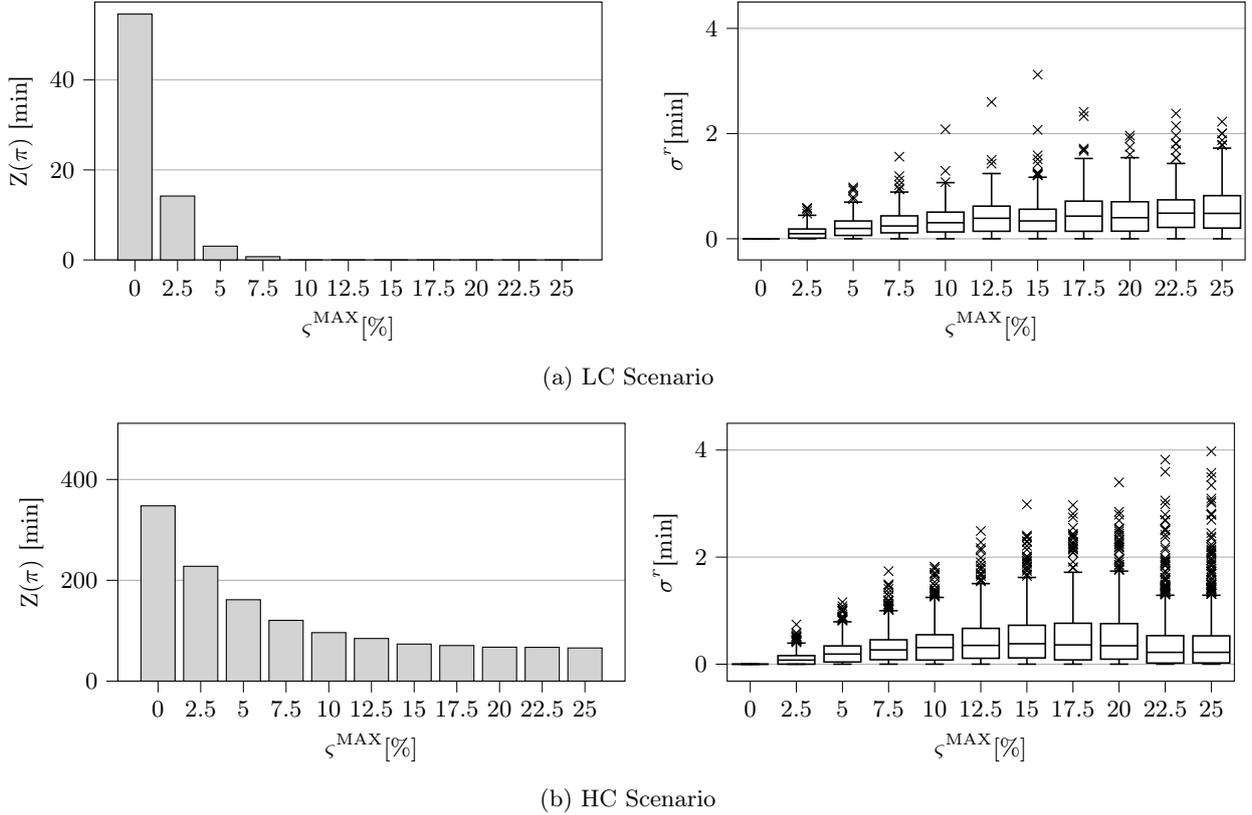
(a) LC Scenario

(b) HC Scenario

Figure 7: Trade off between realized trip departure times shifts σ^r, the total system delay $Z(\pi)$, and the maximum possible shift of a trip's departure relative to its length ς^{MAX} for both the LC and HC scenario.

possible trip departure shift ς^{MAX}, which indicates the maximum possible departure shift relative to each trip's length. We report results for both the LC and HC scenarios. As can be seen, we observe diminishing returns for delay reduction when increasing ς^{MAX} in both scenarios but at different scales. While it is possible to entirely mitigate the average cumulative delay in the LC scenario with any $\varsigma^{MAX} \geq 7.5\%$, the HC scenario shows a plateau of an average cumulative delay of around 65 minutes for $\varsigma^{MAX} \geq 20.0\%$. Remarkably, mitigating the entire delay in the LC scenario with maximum realized departure time shifts below two minutes is possible when choosing $\varsigma^{MAX} = 7.5\%$. Choosing a higher ς^{MAX} solely leads to higher departure time shifts. In the HC scenario, we observe increasing realized departure time shifts when increasing ς^{MAX} while yielding a decreasing overall delay in the system. Surprisingly, the realized departure time shifts remain mostly below six minutes even for high ς^{MAX}; only two outliers exhibit a shift of six minutes.

Result 4 (LC scenario trade off). In the LC scenario, a maximum trip departure shift of 7.5% relative to each trip's length allows to mitigate delays induced by the AMoD fleet entirely at the price of shifting any trip departure by at most two minutes.

Result 5 (HC scenario trade off). In the HC scenario, we observe a decreasing cumulative delay at the price of increasing realized trip departure times when increasing ς^{MAX}. Still, the realized trip departure times remain, except from two outliers at most six minutes even for high ς^{MAX}.

In conclusion, our results show that there is a trade-off between allowing higher maximum and realized departure time shifts and reducing delay in the system. However, a significant delay reduction can be obtained by accepting reasonable departure shifts between two and six minutes for both a system that comprises solely fleet-induced congestion (LC scenario) and a system that is already congested and incurs additional congestion induced by the AMoD fleet (HC scenario). In this context, it's important to recognize that while adjusting the departure time of a trip might not be preferable for a passenger, it nevertheless does not compromise the passenger's primary objective of reaching their destination punctually. This is because, independently of the ς^{MAX} parameterization, alterations to the departure time are only permissible as long as the trip finishes within the designated time window.

6. Conclusion

We studied the impact of staggered routing, i.e., delaying the departure of trips as long as a trip's prefixed arrival time will not be exceeded to reduce local congestion bottlenecks when operating an AMoD fleet. We formalized the underlying planning problem in a full information offline setting as a MILP. We leveraged this MILP to devise an efficient matheuristic that can be used to solve large-scale full information offline instances, while at the same time being amenable to be used in a rolling-horizon online setting. We applied this matheuristic to a real-world case study for the Manhattan area in New York City. We compared the impact of staggered routing for both a full-information bound computed in an idealized offline setting and for the solution obtained in a rolling-horizon online setting against an uncontrolled case in which staggering is not allowed.

Our results show that in low-congestion scenarios, i.e., scenarios in which only the AMoD fleet induces congestion but the system without the fleet would remain uncongested, staggering trip departures allow to mitigate on average 94% of the induced congestion in a full information setting. While not reaching this full information bound, our naive rolling horizon approach still allows us to reduce 90% of the induced congestion in the LC scenario. In high-congestion scenarios, i.e., scenarios in which the system already encounters exogenous congestion and the AMoD fleet induces further congestion, we observe an average reduction of 66% as the full information bound and an average reduction of 56% in our online setting. Surprisingly, we show that these reductions can be reached by shifting trip departures at a maximum of two minutes in the LC scenario and by six minutes in the HC scenario, still preserving each trip's desired arrival time.

Our work presents a first step towards leveraging staggered routing in AMoD systems to reduce congestion. Specifically, we anticipate two fruitful avenues for future research. First, focusing on efficient solution methodologies that solve large-scale instances with high solution quality. Our paper represents a first step in this direction. Still, one may improve upon our results by finding better lower bounds and further enhancing algorithmic components tailored to the respective problem setting. Second, developing prescriptive online algorithms to apply staggered routing in practice. Our myopic rolling-horizon implementation already shows a significant improvement potential, giving

hope to achieve even better performance with a more sophisticated online algorithm incorporating a learning-based prescriptive element.

Acknowledgements

References

Alonso-Mora, J., Wallar, A., & Rus, D. (2017). Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Arnott, R., de Palma, A., & Lindsey, R. (1990). Economics of a bottleneck. *Journal of Urban Economics*, *27*, 111–130.

Bahrami, S., & Roorda, M. J. (2020). Optimal traffic management policies for mixed human and automated traffic flows. *Transportation Research Part A: Policy and Practice*, *135*, 130–143.

Bang, H., & Malikopoulos, A. A. (2022). Congestion-aware routing, rebalancing, and charging scheduling for electric autonomous mobility-on-demand system. In *2022 American Control Conference (ACC)* (pp. 3152–3157).

Enders, T., Harrison, J., Pavone, M., & Schiffer, M. (2023). Hybrid multi-agent deep reinforcement learning for autonomous mobility on demand systems. arXiv:2212.07313 [cs.LG]. doi:10.48550/arXiv.2212.07313.

Estandia, A., Schiffer, M., Rossi, F., Luke, J., Kara, E. C., Rajagopal, R., & Pavone, M. (2021). On the interaction between autonomous mobility on demand systems and power distribution networks—an optimal power flow approach. *IEEE Transactions on Control of Network Systems*, *8*, 1163–1176.

Gammelli, D., Yang, K., Harrison, J., Rodrigues, F., Pereira, F. C., & Pavone, M. (2021). Graph neural network reinforcement learning for autonomous mobility-on-demand systems. In *2021 60th IEEE Conference on Decision and Control (CDC)* (pp. 2996–3003). IEEE.

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. C. McGeoch (Ed.), *Experimental Algorithms* (pp. 319–333). Springer Berlin Heidelberg.

Houshmand, A., Wollenstein-Betech, S., & Cassandras, C. G. (2019). The penetration rate effect of connected and automated vehicles in mixed traffic routing. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (pp. 1755–1760).

Iglesias, R., Rossi, F., Wang, K., Hallac, D., Leskovec, J., & Pavone, M. (2018). Data-driven model predictive control of autonomous mobility-on-demand systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1—-7). IEEE Press.

Jacquillat, A. (2022). Predictive and prescriptive analytics toward passenger-centric ground delay programs. *Transportation Science*, *56*, 265–298.

Jahn, O., Möhring, R. H., Schulz, A. S., & Stier-Moses, N. E. (2005). System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations research*, *53*, 600–616.

Jakob, W., Rhinelander, J., & Moldovan, D. (2017). pybind11 – seamless operability between c++11 and python. URL: https://github.com/pybind/pybind11.

Jalota, D., Paccagnan, D., Schiffer, M., & Pavone, M. (2023). Online routing over parallel networks: Deterministic limits and data-driven enhancements. *INFORMS Journal on Computing*, *35*, 560–577.

Jiao, Y., Tang, X., Qin, Z. T., Li, S., Zhang, F., Zhu, H., & Ye, J. (2021). Real-world ride-hailing vehicle repositioning using deep reinforcement learning. *Transportation Research Part C: Emerging Technologies*, *130*, 103289.

Jungel, K., Parmentier, A., Schiffer, M., & Vidal, T. (2023). Learning-based online optimization for autonomous mobility-on-demand fleet control. arXiv:2302.03963 [math.OC]. doi:10.48550/arXiv.2302.03963.

Köhler, E., Langkau, K., & Skutella, M. (2002). Time-expanded graphs for flow-dependent transit times. In *Algorithms—ESA 2002: 10th Annual European Symposium Rome, Italy, September 17–21, 2002 Proceedings 10* (pp. 599–611). Springer.

Levin, M. W. (2017). Congestion-aware system optimal route choice for shared autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, *82*, 229–247.

Levin, M. W., Kockelman, K. M., Boyles, S. D., & Li, T. (2017). A general framework for modeling shared autonomous vehicles with dynamic network-loading and dynamic ride-sharing application. *Computers, Environment and Urban Systems*, *64*, 373–383.

Levy, J. I., Buonocore, J. J., & Von Stackelberg, K. (2010). Evaluation of the public health impacts of traffic congestion: a health risk assessment. *Environmental health*, *9*, 1–12.

Li, M., Qin, Z., Jiao, Y., Yang, Y., Wang, J., Wang, C., Wu, G., & Ye, J. (2019). Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference* WWW '19 (pp. 983—-994). New York, NY, USA: Association for Computing Machinery.

Li, Z.-C., Huang, H.-J., & Yang, H. (2020). Fifty years of the bottleneck model: A bibliometric review and future research directions. *Transportation Research Part B: Methodological*, *139*, 311–342.

Liang, E., Wen, K., Lam, W. H., Sumalee, A., & Zhong, R. (2021). An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Transactions on Neural Networks and Learning Systems*, *33*, 4742–4756.

Liu, Y., & Samaranayake, S. (2022). Proactive rebalancing and speed-up techniques for on-demand high capacity ridesourcing services. *IEEE Transactions on Intelligent Transportation Systems*, *23*, 819–826.

Narayanan, S., Chaniotakis, E., & Antoniou, C. (2020). Shared autonomous vehicle services: A comprehensive review. *Transportation Research Part C: Emerging Technologies*, *111*, 255–293.

NYC Taxi & Limousine Commission (2015). TLC Trip Record Data. URL: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

Oh, S., Seshadri, R., Azevedo, C. L., Kumar, N., Basak, K., & Ben-Akiva, M. (2020). Assessing the impacts of automated mobility-on-demand through agent-based simulation: A study of singapore. *Transportation Research Part A: Policy and Practice*, *138*, 367–388.

OpenStreetMap (2024). URL: https://www.openstreetmap.org.

Paccagnan, D., & Gairing, M. (2023). In congestion games, taxes achieve optimal approximation. *Operations Research*. (Ahead of Print).

Patriksson, M. (2015). *The traffic assignment problem: models and methods*. Courier Dover Publications.

Pavone, M. (2015). Autonomous mobility-on-demand systems for future urban mobility. In *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte* (pp. 399–416). Berlin, Heidelberg: Springer Berlin Heidelberg.

Pigou, A. (1920). *The economics of welfare*. London, Macmillan.

Pishue, B. (2023). *2022 INRIX Global Traffic Scorecard*. Technical Report INRIX.

Rossi, F., Iglesias, R., Alizadeh, M., & Pavone, M. (2020). On the interaction between autonomous mobility-on-demand systems and the power network: Models and coordination algorithms. *IEEE Transactions on Control of Network Systems*, *7*, 384–397.

Rossi, F., Zhang, R., Hindy, Y., & Pavone, M. (2018). Routing autonomous vehicles in congested transportation networks: structural properties and coordination algorithms. *Autonomous Robots*, *42*, 1427–1442.

Roughgarden, T. (2005). *Selfish routing and the price of anarchy* volume 74. MIT press.

Ruch, C., Lu, C., Sieber, L., & Frazzoli, E. (2020). Quantifying the efficiency of ride sharing. *IEEE Transactions on Intelligent Transportation Systems*, *22*, 5811–5816.

Sadeghi Eshkevari, S., Tang, X., Qin, Z., Mei, J., Zhang, C., Meng, Q., & Xu, J. (2022). Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* KDD '22 (pp. 3838—-3848). New York, NY, USA: Association for Computing Machinery.

Salazar, M., Lanzetti, N., Rossi, F., Schiffer, M., & Pavone, M. (2019). Intermodal autonomous mobility-on-demand. *IEEE Transactions on Intelligent Transportation Systems*, *21*, 3946–3960.

Skordilis, E., Hou, Y., Tripp, C., Moniot, M., Graf, P., & Biagioni, D. (2021). A modular and transferable reinforcement learning framework for the fleet rebalancing problem. *IEEE Transactions on Intelligent Transportation Systems*, *23*, 11903–11916.

Sriskandarajah, C., & Ladet, P. (1986). Some no-wait shops scheduling problems: Complexity aspect. *European Journal of Operational Research*, *24*, 424–438. Flexible Manufacturing Systems.

Tang, X., Qin, Z. T., Zhang, F., Wang, Z., Xu, Z., Ma, Y., Zhu, H., & Ye, J. (2019). A deep value-network based approach for multi-driver order dispatching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* KDD '19 (pp. 1780—-1790). New York, NY, USA: Association for Computing Machinery.

Tsao, M., Iglesias, R., & Pavone, M. (2018). Stochastic model predictive control for autonomous mobility on demand. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (pp. 3941–3948).

US Bur. Public Roads (1964). *Traffic assignment manual for application with a large, high speed computer* volume 2. Washington, DC: US Gov. Print. Off.

Vickrey, W. S. (1969). Congestion theory and transport investment. *The American Economic Review*, (pp. 251–260).

Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., & Ye, J. (2018). 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* KDD '18 (pp. 905—913). New York, NY, USA: Association for Computing Machinery.

Zardini, G., Lanzetti, N., Pavone, M., & Frazzoli, E. (2022). Analysis and control of autonomous mobility-on-demand systems. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*, 633–658.

Zhang, L., Hu, T., Min, Y., Wu, G., Zhang, J., Feng, P., Gong, P., & Ye, J. (2017). A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '17 (pp. 2151—-2159). New York, NY, USA: Association for Computing Machinery.

Zhang, W., & Guhathakurta, S. (2017). Parking spaces in the age of shared autonomous vehicles: How much parking will we need and where? *Transportation Research Record*, *2651*, 80–91.

Zhou, M., Jin, J., Zhang, W., Qin, Z., Jiao, Y., Wang, C., Wu, G., Yu, Y., & Ye, J. (2019). Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* CIKM '19 (pp. 2645—-2653). New York, NY, USA: Association for Computing Machinery.

Appendix A Hardness Proof

To prove the hardness of the staggered routing problem as defined in Section 2, we utilize the hardness result for the three machines job shop scheduling problem with unit processing times and no-wait constraints (J3-UPT-NWT) with makespan (the maximum completion time of a job) minimization as objective, which is NP-hard (Sriskandarajah & Ladet 1986).

Theorem 1. The staggered routing problem is NP-hard.

Proof We begin by defining instances for J3-UPT-NWT and for the staggered routing problem:

Definition 1. Let I be an instance of the J3-UPT-NWT with $m = 3$ machines and jobs J_i, with $i = 1, \ldots, n$, where each job comprises $N(J_i)$ operations that each require one unit of processing time and must be executed continuously from start to finish. The operations are allocated to specific machines, and the order in which machines are visited may vary among different jobs. Each machine can process a maximum of one job at a time. The J3-UPT-NWT seeks for a job schedule S, i.e., the starting times $s(J_i)$ of the first operation of each job J_i, that minimizes the schedule's makespan D.

Definition 2. Let I' be an instance of the staggered routing problem with n trips as defined in Section 2. Trips must proceed along their routes without interruption within designated time windows. The travel time for a trip along an arc is influenced by the number of trips on the arc at the specific time the trip enters the arc. If the number of trips on the arc falls below a predefined arc-specific threshold capacity, the trip proceeds at the nominal travel time designated for that arc. Conversely, exceeding this threshold triggers a delay, as detailed in Equation (2.1). The objective of the staggered routing problem is to find the trip departures s_a^r that minimize the total travel time.

Given these definitions, we prove the hardness of the staggered routing problem by contradiction.

Assumption 1. It exists an algorithm $\mathscr{A}$ that finds a feasible solution for every instance of the staggered routing problem in polynomial time.

Step 1: Consider an instance I' that comprises n trips traversing $m = 3$ arcs, where at least two trips have different routes. The earliest possible departure time for trips is set to zero, with no restrictions on the staggering of departures. All trips must finish by time D. Each arc has a nominal travel time and a threshold capacity of one; exceeding this threshold on any arc introduces an infinite delay.

Step 2: Instance I' can be transformed in polynomial time to an instance I as follows: the route of each trip corresponds to a job to be executed without interruptions from start to finish. The act of traversing the arcs within a trip's route mirrors the operations within a job. These operations are executed by a dedicated machine (an arc), each with a unit processing time (equivalent to nominal travel time), and concurrent processing of operations on the same machine is prohibited (otherwise, one incurs infinite delay). The latest arrival times translate to the schedule's makespan under consideration. Finally, the departure times from the arcs on a route correspond to the scheduling times of the job operations.

Step 3: Observe that selecting the job's starting times in instance I to decide whether a schedule S of makespan of D exists is equivalent to finding feasible trip departure times in instance I', such that we obtain a polynomial-time transformation from the respective staggered routing problem instance instance to an equivalent J3-UPT-NWT instance. Then, if $\mathscr{A}$ can find a feasible solution in polynomial time for I', it can also find a feasible solution for I within polynomial time. However, finding a solution for I in polynomial time is impossible unless P = NP, contradicting Assumption 1. Accordingly, there exists at least one instance of the staggered routing problem that cannot be solved in polynomial time. □

Appendix B   Table of Variables in the MILP

Table 2: Table of Variables in the MILP.

Variable	Description
$\mathcal{R}$	Set of all trips $r \in \mathcal{R}$
$\mathcal{A}$	Set of all arcs $a \in \mathcal{A}$
$\mathcal{R}_a$	Subset of $\mathcal{R}$, set of trips traversing arc a
p^r	Route of trip r
$\delta^r(a)$	Arc succeeding a on route p^r
$\underline{a}^r$	First arc of p^r
$\bar{a}^r$	Last arc of p^r
x_a^r	Delay on arc a for trip r
μ_a^k	Slope of the k-th segment of arc a travel time function.
τ_a	Nominal travel time for arc a
f_a^r	Number of vehicles encountered by trip r on arc a
$\hat{f}_a^k$	Flow value at which the k-th segment of arc a travel time function begins.
s_a^r	Departure time of trip r on arc a
e^r	Earliest possible departure time for trip r
ℓ^r	Latest possible arrival time for trip r
$\bar{\sigma}^r$	Maximum staggering applicable to trip's r departure
$\alpha_a^{r,r'}, \beta_a^{r,r'}, \gamma_a^{r,r'}$	Logical variables checking concurrent presence of trips (r, r') on arc a
ε	Small constant
M_i	Large constant

Appendix C   Optimality Gaps

Figures 8–10 show the distribution of the optimality gap (Figure 8), the absolute value of the lower bound (Figure 9), and the distance between the upper bound and the lower bound (Figure 10) for both the LC scenario and the HC scenario, obtained when running our matheuristic with a time limit of two hours.

Focusing on the relative optimality gaps in Figure 8, we observe a bipolar distribution in which optimality gaps accumulate either around 0% or around 100% for the LC scenario. For the HC scenario, all relative optimality gaps range between 90% and 100%. At first sight, these optimality
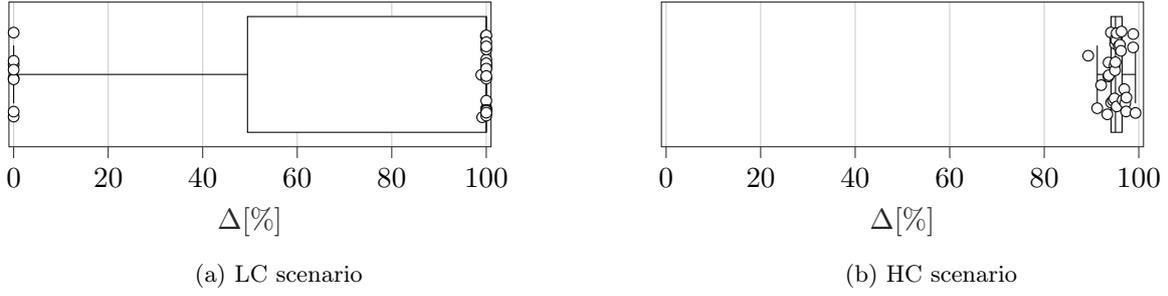
(a) LC scenario

(b) HC scenario

Figure 8: Distribution of the optimality gap Δ [%] over all instances.

(a) LC scenario

(b) HC scenario

Figure 9: Distribution of the absolute value of the lower bound LB [min] over all instances.
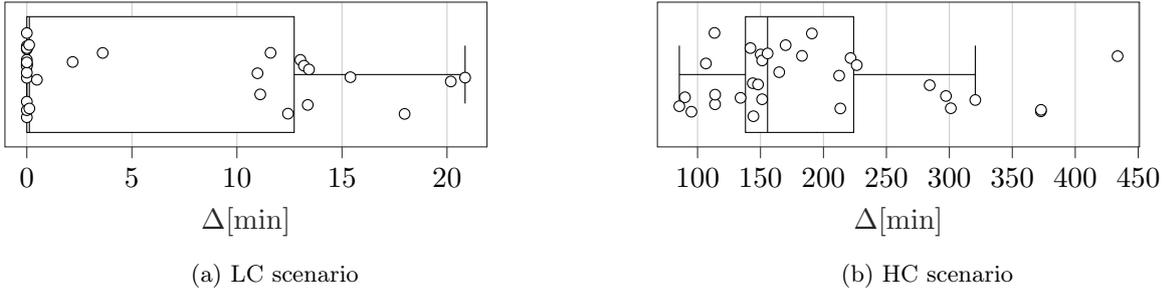

(a) LC scenario

(b) HC scenario

Figure 10: Distribution of the distance between the upper and lower bound UB-LB [min] over all instances.

gaps look anything but satisfying in terms of algorithmic performance, but unravel as follows at second sight: as our objective is to minimize the total congestion-related delay in the system, a trivial lower bound on our objective value is an objective value of zero. Clearly, our problem bears a huge degree of symmetry, which leads to weak lower bounds that cannot be significantly improved in our MILP formulation, see Figure 9. In such cases, an absolute distance of even a few minutes of delay left in the system can create a large optimality gap.

To support our claim that our matheuristic finds good solutions, we compare these optimality gaps to the delay reductions shown in Figure 4. In fact, a comparison to the uncontrolled solution provides more insights compared to a lower bound comparison: relating Figure 4 to Figure 8, it becomes obvious that the maximum improvement potential to reduce delay remains at 20% and 50% respectively for the LC and HC scenarios, although the optimality gap distribution shows values up to 100%.

Remark 2 (lower bound). When minimizing the total congestion-related delay in a staggered routing setting, the relative optimality gap is an uninformative indicator of solution quality due to the nature of the objective function and the problem's weak lower bounds.

In this context, Figure 10 reports the difference between the solutions found and the lower bound, indicating that our matheuristic founds good solutions despite large optimality gaps. Specifically, we observe an average distance between the upper (UB) and lower (LB) bound of 6 and 193 minutes for the LC and HC scenarios. As the instances exhibit an average delay of 82 min and 577 min respectively in an uncontrolled setting, the absolute distance between the upper and lower bound appears rather small, indicating that our matheuristic effectively reduces congestion via staggering.