Throughput-Fairness Tradeoffs in Mobility Platforms

Arjun Balasingam[⋆], Karthik Gopalakrishnan[⋆], Radhika Mittal[†], Venkat Arun[⋆],
Ahmed Saeed[⋆], Mohammad Alizadeh[⋆], Hamsa Balakrishnan[⋆], Hari Balakrishnan[⋆]
[⋆]Massachusetts Institute of Technology [†]University of Illinois at Urbana-Champaign

ABSTRACT

This paper studies the problem of allocating tasks from different customers to vehicles in mobility platforms, which are used for applications like food and package delivery, ridesharing, and mobile sensing. A mobility platform should allocate tasks to vehicles and schedule them in order to optimize both throughput and fairness across customers. However, existing approaches to scheduling tasks in mobility platforms ignore fairness.

We introduce Mobius, a system that uses guided optimization to achieve both high throughput and fairness across customers. Mobius supports spatiotemporally diverse and dynamic customer demands. It provides a principled method to navigate inherent tradeoffs between fairness and throughput caused by shared mobility. Our evaluation demonstrates these properties, along with the versatility and scalability of Mobius, using traces gathered from ridesharing and aerial sensing applications. Our ridesharing case study shows that Mobius can schedule more than 16,000 tasks across 40 customers and 200 vehicles in an online manner.

CCS CONCEPTS

• **Computer systems organization** → **Robotics**; **Sensor networks**; • **Computing methodologies** → **Planning and scheduling**; • **Applied computing** → **Transportation**; • **Networks** → *Network resources allocation.*

KEYWORDS

mobility platforms, vehicle routing, aerial sensing, ridesharing, resource allocation, optimization

ACM Reference Format:
Arjun Balasingam, Karthik Gopalakrishnan, Radhika Mittal, Venkat Arun, Ahmed Saeed, Mohammad Alizadeh, Hamsa Balakrishnan, Hari Balakrishnan. 2021. Throughput-Fairness Tradeoffs in Mobility Platforms. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24–July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458864.3467881

1 INTRODUCTION

The past decade has seen the rapid proliferation of mobility platforms that use a fleet of mobile vehicles to provide different services. Popular examples include package delivery (UPS, DHL, FedEx, Amazon), food delivery (DoorDash, Grubhub, Uber Eats), and rideshare services (Uber, Lyft). In addition, new types of mobility platforms are emerging, such as drones-as-a-service platforms [21, 27, 32, 48] for deploying different sensing applications on a fleet of drones.

In these mobility platforms, the vehicle fleet of cars, vans, bikes, or drones is a *shared infrastructure*. The platform serves multiple *customers*, with each customer requiring a *set of tasks* to be completed. For instance, each restaurant subscribing to DoorDash is a customer, with several food delivery orders (or tasks) in a city. Similarly, an atmospheric chemist and a traffic analyst might subscribe to a drones-as-a-service platform, each with their own sensing applications to collect air quality measurements and traffic videos, respectively, at several locations in the same urban area. Multiplexing tasks from different customers on the same vehicles can increase the efficiency of mobility platforms because vehicles can amortize their travel time by completing co-located tasks (belonging to either the same or different customers) in the same trip.

We study the problem of scheduling spatially distributed tasks from multiple customers on a shared fleet of vehicles. This problem involves (i) assigning tasks to vehicles and (ii) determining the order in which each vehicle must complete its assigned tasks. The constraints are that each vehicle has bounded resources (fuel or battery). While several variants of this scheduling problem have been studied, the objective has typically been to complete as many tasks as possible in bounded time, or to maximize aggregate throughput (task completion rate) [23, 44].

We identify a second—equally important—scheduling requirement, which has emerged in today's customer-centric mobility platforms: *fairness* of customer throughput to ensure that tasks from different customers are fulfilled at similar rates.[1] For example, in food delivery, the platform should serve restaurants equitably, even if it means spending time or resources on restaurants with patrons far from the current location of the vehicles. A ridesharing platform should ensure that riders from different neighborhoods are served equitably, which ridesharing platforms today do not handle well, a phenomenon known as "destination discrimination" [35, 45, 49].

We seek an online scheduler for mobility platforms that achieves both high throughput and fairness. A standard approach to achieving these goals is to track the resource usage and work done on behalf of different users in a fine-grained way and equalize resource consumption across users. Such fine-grained accounting and attribution is difficult with shared mobility: the resource used is a moving vehicle traveling toward its next task, but making that trip has a knock-on benefit, not only for the next task served, but for subsequent ones as well. However, the benefit of a specific trip is not equal across the subsequent tasks. Although it may be possible to develop a fair scheduler that achieves high throughput using fine-grained accounting and attribution, it is likely to be complex.

MobiSys '21, June 24–July 2, 2021, Virtual, WI, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8443-8/21/06.
https://doi.org/10.1145/3458864.3467881

[1] The method we develop also applies to weighted fairness.

We turn, instead, to an approach that has been used in both societal and computing systems: optimization, which may be viewed as a search through a set of feasible schedules to maximize a utility function. In our case, we can establish such a function, optimize it using both the task assignment and path selection, and then route vehicles accordingly.

In a typical mobility problem, the planning time frame for optimization could be between 30 minutes and several hours, involving hundreds of vehicles, dozens of customers, and tens of thousands of tasks. The scale of this problem pushes the limits of state-of-the-art vehicle routing solvers [7]. Moreover, fairness objectives lead to nonlinear utility functions, which make the optimization much more challenging. As a benchmark, optimizing the routes for 3 vehicles and 17 tasks over 1 hour, using the CPLEX solver [28] with a nonlinear objective function, takes over 10 hours [36].

To address these problems, a natural approach is to divide the desired time duration into shorter rounds, and then run the utility optimization. When we do this, something interesting emerges in mobility settings: the space of feasible solutions—each solution being an achievable set of rates for the customers—often *collapses into a rather small and disturbingly suboptimal set!* These feasible solutions are either fair but with dismal throughput, or with excellent throughput but starving several customers.

A simple example helps see why this happens. Consider a map with three areas, A_1, A_2, A_3, each distant from the others. There are several tasks in each area: in A_1, all the tasks are for customer C_1; in A_2, all the tasks are for customer C_2, and in A_3, all the tasks are for two other customers, C_3 and C_4. Suppose that there are two vehicles. Over a duration of a few minutes, we could either have the two vehicles focus on only two areas, achieving high throughput but ignoring the third area and reducing fairness, or, we could have them move between areas after each task to ensure fairness, but waste a lot of time traveling, degrading throughput. It is not possible here to achieve both throughput and fairness *over a short timescale*. Yet, over a long time duration, we can swap vehicles between regions to amortize the movement costs. This shows that planning over a longer timescale permits feasible schedules that are better than what a shorter timescale would permit.

Our contribution, *Mobius*, divides the desired time duration into rounds, and produces the feasible set of allocations for that round using a standard optimizer. Mobius guides the optimizer toward a solution that is not in the feasible set for one round but can be achieved over multiple rounds. This guiding is done by aiming for an objective that maximizes a weighted linear sum of customer rates in each round. The weights are adjusted dynamically based on the long-term rates achieved for each customer thus far. The result is a practical system that achieves high throughput and fairness over multiple rounds. This approach of achieving long-term fairness by setting appropriate weights across rounds allows us to use off-the-shelf solvers for the weighted Vehicle Routing Problem (VRP) for path planning in each round. Importantly, this design allows Mobius to optimize for fairness in the context of any VRP formulation, making this work complementary to the vast body of prior work on vehicle routing algorithms [3, 5, 8, 23].

Scheduling over multiple rounds also allows Mobius to handle tasks that arrive dynamically or expire before being done. Moreover, Mobius supports a tunable level of fairness modeled by α-fair utility functions [31], which generalize the familiar notions of max-min and proportional fairness.

We have implemented Mobius and evaluated it via extensive trace-driven emulation experiments in two real-world settings: (i) a ridesharing service, based on real Lyft ride request data gathered over a day, ensuring fair quality-of-service to different neighborhoods in Manhattan; and (ii) urban sensing using drones for measuring traffic congestion, parking lot occupancy, cellular throughput, and air quality. We find that:

1. Relative to a scheduler that maximizes only throughput, Mobius compromises only 10% of platform throughput in order to enforce max-min fairness.
2. Compared to dedicating vehicles to customers, Mobius improves vehicle utilization by 30-50% by intelligently sharing vehicles amongst customers.
3. Mobius can compute fair online schedules at a city scale, involving 40 customers, 200 vehicles, and over 16,000 tasks.

2 PROBLEM SETUP

Every customer subscribing to a mobility platform submits several requests over time. Each request specifies a task (e.g., gather sensor data or deliver package) and a corresponding location. The platform schedules trips for each vehicle over multiple rounds. It takes into account any changes in a customer's requirements (in the form of new task requests or expiration of older unfulfilled tasks) at the beginning of each round. We say that a customer has a backlog if they have more tasks than can be completed by all available resources within the allocated time. For simplicity of exposition, we assume each customer is backlogged (our evaluation in §7 relaxes this assumption).

Let K be the set of customers, and $T_k(\tau)$ be the set of tasks requested by customer k during a scheduling round τ. We denote $x_k(\tau)$ as the throughput achieved for customer k in scheduling round τ, i.e., the total number of tasks in $T_k(\tau)$ that are fulfilled divided by the round duration.

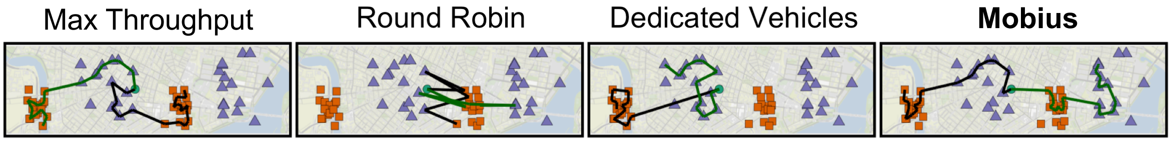
We denote $\overline{x}_k(t)$ as the long-term throughput for each customer k, after t scheduling rounds, i.e., $\overline{x}_k(t) = \frac{1}{t}\sum_{\tau=1}^{t} x_k(\tau)$ if rounds are of equal duration. A good scheduling algorithm should achieve the following objectives:

- **Platform Throughput.** Maximize the total long-term throughput after round t, i.e., $\sum_{k \in K} \overline{x}_k(t)$.
- **Customer Fairness.** For any two customers k_1,$k_2 \in K$ with backlogged tasks, ensure $\overline{x}_{k_1}(t) = \overline{x}_{k_2}(t)$.

Equalizing long-term per-customer throughputs $\overline{x}_k(t)$ provides a desirable measure of fairness for many mobility platforms: higher per-customer throughputs correlate with other performance metrics, such as lower task latency and higher revenue. Our evaluation (§7) quantifies the impact of optimizing for a fair allocation of throughputs on other platform-specific quality-of-service metrics.

Prior algorithms for scheduling tasks on a shared fleet of vehicles have focused on the VRP, i.e., only considered maximizing platform throughput [23, 44]. Achieving per-customer fairness introduces three new challenges:

Challenge #1: Attributing vehicle time to customers. Vehicle time and capacity are scarce. Consider the example in Fig. 1, with two customers and two vehicles; customer 1 has two densely-packed clusters of tasks, while customer 2 has two dispersed clusters of



	Max Throughput	Round Robin	Dedicated Vehicles	Mobius
Cust. 1	28 tasks	4 tasks	15 tasks	21 tasks
Cust. 2	13 tasks	4 tasks	13 tasks	19 tasks
Total	41 tasks	8 tasks	28 tasks	40 tasks

Figure 1: An example with two customers, two vehicles, and a 6-minute planning horizon. Mobius computes a schedule that (i) achieves a similar total throughput to that of the max throughput schedule, and (ii) preserves the customer-level fairness achieved by the round-robin and dedicated schedules.

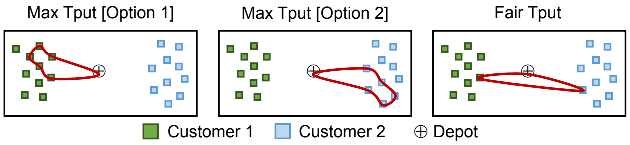


Figure 2: Imposing fairness at short timescales (e.g., one round trip) degrades throughput. Executing Options 1 and 2 provides fairness at longer timescales and leads to greater total throughput.

tasks. We show schedules and tasks fulfilled by Mobius and three other policies: (i) maximizing throughput, (ii) dedicating a vehicle per customer, and (iii) alternating round-robin between customer tasks. Notice that, to the left of the depot (center of the map), customer 2's tasks can be picked up on the way to customer 1's tasks. Thus, multiplexing both customers' tasks on the same vehicle is more desirable than dedicating a vehicle per customer, because it amortizes resources to serve both customers. However, sharing vehicles amongst customers complicates our ability to reason about fairness, because the travel time between the tasks of different customers cannot be attributed easily to each one.

Challenge #2: Timescale of fairness. Fig. 2 shows two customers and one vehicle that must return home to refuel. A high-throughput schedule would dedicate the vehicle to one of the customers. By contrast, a fair schedule would require the vehicle to round-robin customer tasks, achieving low throughput due to travel. Over a longer time duration, however, we can execute two max-throughput schedules (Options 1 and 2) to achieve both fairness and high throughput.

Challenge #3: Spatiotemporal diversity of tasks. In Fig. 1, the two customers' tasks have different spatial densities. The high-throughput schedule favors customer 1. A max-min fair schedule should, by contrast, ensure that customer 2 gets its fair share of the throughput, even if it comes at the cost of higher travel time and lower platform throughput. Striking the right balance between fairly serving a customer with more dispersed tasks and reducing extra travel time is a non-trivial problem.

Customer tasks may also *vary with time*. For example, a food delivery service might receive new requests from restaurants, or an atmospheric scientist may want to update sensing locations that they submitted to a drone service provider based on prior observations. The mobility platform must handle the dynamic arrival and expiration of tasks.

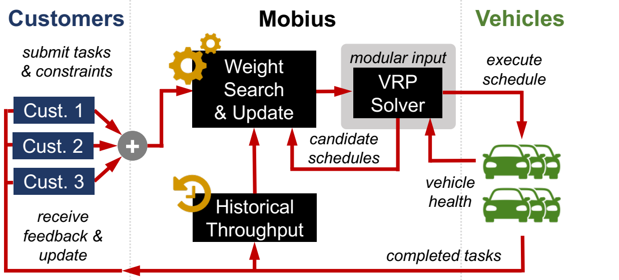


Figure 3: In each round, Mobius uses a VRP solver to compute a schedule that maximizes a weighted sum of throughputs, and automatically adjusts the weights across rounds to improve fairness.

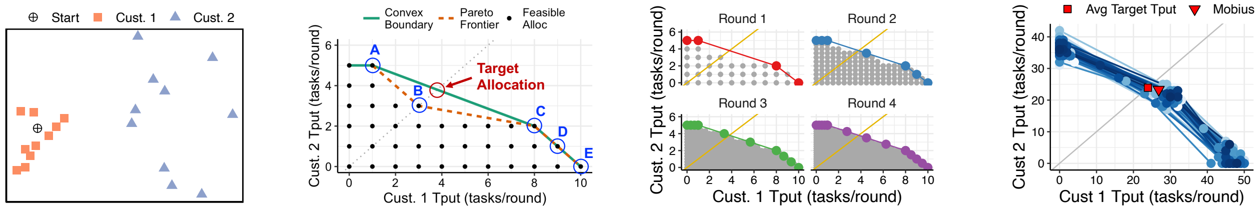
3 OVERVIEW

Any resource-constrained system exhibits an inherent tradeoff between throughput and fairness. In the best case, the most fair schedule would also have the highest throughput; however, due to the challenges described in §2, it is impossible to realize this goal in many mobility settings. Mobius instead strives for customer fairness with the best possible platform throughput; its approach is to trade some short-term fairness for a boost in throughput, while improving fairness over a longer timescale.

In each round τ, Mobius uses a VRP solver to maximize a weighted sum of customer throughputs $x_k(\tau)$.[2] Mobius sets the weights in each round to find a high throughput schedule that is approximately fair in that round. By accounting for the long-term throughputs $\overline{x}_k(t)$ delivered to each customer k in prior rounds, it is able to equalize $\overline{x}_k(t)$ over multiple rounds. We formalize this notion of balancing high throughput with fairness in §4. Mobius uses an iterative search algorithm requiring multiple invocations of a VRP solver to find a schedule that strikes the appropriate balance.

Our approach of trading off short-term fairness for throughput and longer-term fairness raises a natural question: why not directly schedule over a longer time horizon, rather than dividing the scheduling problem into rounds? Scheduling in rounds is desirable for several reasons: (i) their duration can correlate with the fuel or battery constraints of the vehicles, (ii) it provides a target timescale at which Mobius strives to provide fairness, (iii) shorter timescales make the NP-hard VRP problem more tractable to solve, and (iv) it

[2] We formally define the VRP in §5.



(a) Map. Two vehicles start at ⊕.

(b) Feasible throughputs in 1 round.

(c) Feasible throughputs over 4 rounds.

(d) Convex boundary dynamics.

Figure 4: Visualizing feasible allocations of throughput for a small problem with two customers and two vehicles. Allocations on the convex boundary trade short-term fairness for throughput. The convex boundary becomes denser over time, making the target allocation achievable.

enables Mobius to adapt to temporal variations in customer demand that are captured at the beginning of each round.

Fig. 3 shows the architecture of Mobius. In each round, customers update their task requests. Mobius then computes the best weights, generates a schedule, and dispatches the vehicles. At the end of the round, Mobius updates each customer's throughput, $\overline{x}_k(t)$, and uses this information to select weights in the next round.

4 BALANCING THROUGHPUT & FAIRNESS

We now provide the intuition behind our approach for balancing throughput and fairness using the example shown in Fig. 4. There are two customers, each requesting tasks from distributions shown on the map in Fig. 4a. We have two vehicles, each starting at ⊕. For simplicity, in §4.1, we consider planning schedules in 10-minute rounds, where the vehicles return to their start locations after 10 minutes. We renew all tasks at the beginning of each round trip. Then, in §4.2, we explain how Mobius generalizes to dynamic settings where customer tasks change with time, and vehicles do not need to return to their starting locations.

4.1 Scheduling on the Convex Boundary

Feasible allocations. We first consider the set of schedules that are feasible within the time constraint. Fig. 4b shows the tradeoff between throughput and fairness amongst these feasible schedules. Each dot represents an allocation produced by a feasible schedule; the coordinates of the dot indicate the throughputs of the respective customers. We generate the schedules by solving the VRP for each possible subset of customer tasks.[3] We also indicate the $y = x$ line (dotted gray), which corresponds to fair allocations that give equal throughput to each customer. Note that in this example both vehicles can more easily service Customer 1. Hence, an allocation that maximizes total throughput without regard to fairness (labeled C) favors Customer 1.

Pareto frontier of feasible allocations. The Pareto frontier over all feasible allocations is denoted by the dashed orange line, containing A, B, C, D, and E. If an allocation on the Pareto Frontier achieves throughputs of x_1 and x_2 for Customers 1 and 2 respectively, there exists no feasible allocation ($\hat{x}_1$,$\hat{x}_2$) such that $\hat{x}_1 > x_1$ and $\hat{x}_2 > x_2$. The allocation that maximizes total throughput will always lie on the Pareto frontier. An allocation on the Pareto frontier is strictly superior, and therefore more desirable than other feasible allocations. So which allocation on the Pareto frontier do we pick? A strictly fair allocation lies at the point where the Pareto frontier intersects the $y = x$ line (labeled B in Fig. 4b). However, allocation B has low total throughput, because the vehicles spend a significant part of the 10 minutes traveling between task clusters.

Convex boundary of the Pareto frontier. To capture the subset of allocations that do not significantly compromise throughput, we use the *convex boundary* of all feasible allocations, denoted by the turquoise line in Fig. 4b. The convex boundary is the smallest polygon around the feasible set such that no vertex bends inward [9], and the *corner points* are the vertices determining this polygon. The *target allocation* is the point where the $y = x$ lines intersects the boundary (shown in red). It has high throughput and is fair, but it may not be feasible (as in this example). Is it still possible to achieve the target throughput in such cases?

Scheduling over multiple rounds. Our key insight is that it is possible to achieve the target allocation over multiple rounds of scheduling by selecting different feasible allocations on the convex boundary in each round. In a given round, Mobius chooses the feasible allocation on the convex boundary that best achieves our fairness criteria. In our example, it chooses allocation A in its first round. By choosing allocation A over allocation B (which achieves equal throughput), Mobius compromises on short-term fairness for a boost in throughput. However, as we discuss next, it compensates for this choice in subsequent rounds. Notice that if Mobius instead chooses B, it would not be able to recover from the resulting loss in throughput.

As we compute a 10-minute schedule for each round, the set of feasible allocations expands; this allows Mobius to compensate for any prior deviation in fairness. Fig. 4c illustrates how the feasible set evolves over several 10-minute rounds of planning. The feasible allocations (denoted by gray dots) possible after round T are derived from the cumulative set of tasks completed in T rounds. Notice that over the four rounds, the set of feasible allocations becomes denser, and the Pareto frontier approaches the convex boundary. Thus, the target allocation (i.e., the allocation on the convex boundary that coincides with the $y = x$ line) becomes feasible.

In summary, the key insights driving the design of Mobius are: (i) the convex boundary describes a set of allocations that trade off short-term fairness for a boost in throughput, and (ii) the Pareto frontier approaches the convex boundary over multiple rounds of planning, making it possible to correct for unfairness over a slightly longer timescale.

4.2 Scheduling in Dynamic Environments

In practice, environments are more dynamic: customer tasks may not recur at the same locations, and vehicles need not return to their start locations regularly. Thus, the convex boundary may not be identical

[3] The VRP is NP-hard (§5), but because the input size is small for this example, we use Gurobi [25] to compute optimal schedules.

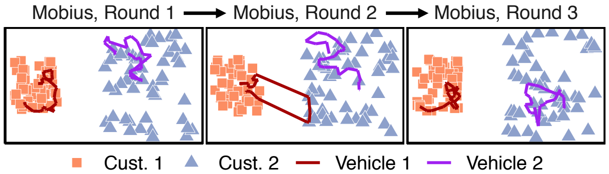


Figure 5: The difference in spatial density of tasks leads to short-term unfairness (Rounds 1 and 3). Mobius compensates for this by directing more resources to the underserved customer (Round 2).

in each round. However, in practice, because (i) vehicles move continuously over space and (ii) customer tasks tend to observe spatial locality, the convex boundary does not change drastically over time.

To illustrate this, we extend the example in Fig. 4, by creating a map with the same densities as in Fig. 4a, but with 50 tasks per customer. To simulate dynamics, we create a new task for each customer every 3 minutes at a location chosen uniformly at random within a bounding box. We still consider two vehicles starting at the same location (i.e., in the middle of customer 1's cluster) and plan in 10-minute rounds. We eliminate the return-to-home constraint. In order to adapt to the customers' changing tasks, we compute new 10-minute schedules every 1 minute (i.e., 10-minute rounds slide in time by 1 minute). We run this simulation for 60 minutes.

In order to understand how these dynamics impact the convex boundary as we plan iteratively, we show in Fig. 4d the convex boundary of 10-minute schedules at each 1-minute replanning interval. Notice that the convex boundaries hover around a narrow band, indicating that we can still track the target throughput reliably. The red square marks the value of the average target throughput across all 50 convex boundaries; we also mark the throughput achieved by Mobius's scheduling algorithm (§5).

In addition to the convex boundary remaining relatively stable from one timestep to the next, this method of replanning at much quicker intervals (e.g., 1 minute) than the round duration (e.g., 10 minutes) makes Mobius resilient to uncertainty in the environment.[4] For instance, Mobius can react to streaming requests in a punctual manner, and can also incorporate requests that are unfulfilled due to unexpected delays (e.g., road traffic or wind). Moreover, since Mobius uses a VRP solver as a building block to compute its schedule (§3), it can also leverage algorithms that solve the stochastic VRP [8], where requests arrive and disappear probabilistically.

4.3 Visualizing Routes Scheduled by Mobius

To illustrate how Mobius converges to fair per-customer allocations without significantly degrading platform throughput, in Fig. 5 we show 3 consecutive 10-minute round schedules computed by Mobius, for the dynamic example in Fig. 4d. In Rounds 1 and 3, we observe that Mobius decides to dedicate one vehicle to each customer in order to give them both sufficiently high throughput; here, customer 2 receives lower throughput because its tasks are more dispersed. However, in Round 2, Mobius compensates for this short-term unfairness by scheduling an additional vehicle to customer 2, while also collecting a few tasks for customer 1 in the outbound trip.

5 MOBIUS SCHEDULING ALGORITHM

Based on the insights in §4, we design Mobius to compute a schedule on the convex boundary in each round, such that the long-term throughputs $\overline{x}_k(t)$ approach the target allocation. Mobius works in two steps:

(1) In each round, Mobius finds the *support allocations*, which we define as the corner points on the convex boundary of the current round, near the target allocation (§5.1). For example, in Fig. 4b, Mobius would find support allocations A and C.
(2) Amongst the support allocations found in step (1), Mobius selects the one that *steers the long-term throughputs* $\overline{x}_k(t)$ toward the target allocation (§5.2).

In this section, we present Mobius in the context of strict fairness (i.e., $\overline{x}_k(t)$ must lie along the $y=x$ line). §5.3 provides a theoretical analysis of Mobius's optimality under simplifying assumptions, and §5.4 describes our implementation. In §6, we extend Mobius's formulation to work with a class of fairness objectives.

5.1 Finding Support Allocations

Since the convex boundary of the Pareto frontier is equivalent to the convex boundary of the feasible set of schedules, a naive way to find the support allocations is to compute the Pareto frontier, take its convex boundary, and then identify the support allocations near the target allocation. However, computing the Pareto frontier is computationally expensive because it requires invoking an NP-hard solver an exponential number of times in the number of tasks. Mobius uses a VRP solver as a building block to find a subset of the corner points of the convex boundary around the target allocation.

The VRP involves computing a path $\mathcal{P}_v$ for each vehicle v, defined as an ordered list of tasks from the set of all tasks $\{T_k(\tau) \mid k \in K\}$, such that the time to complete $\mathcal{P}_v$ does not exceed the total time budget B for a round. VRP solvers maximize the platform throughput without regard to fairness.

We capture different priorities amongst customer tasks by assigning a weight w_k to each customer k's tasks. Let $\mathbf{x} \in \mathbb{R}^{|K|}$ represent a throughput vector, where x_k is the throughput for customer k, and let $\mathbf{w} \in \mathbb{R}^{|K|}$ represent a weight vector, with a weight w_k for each customer k.[5]

The weighted VRP seeks to maximize the total *weighted throughput* of the system, where each task is allowed a weight. We can describe this as a mixed-integer linear program:

$$\underset{\mathcal{P}_v, \forall v \in V}{\text{argmax}} \quad \sum_{k \in K} w_k x_k = \underset{\mathcal{P}_v, \forall v \in V}{\text{argmax}} \; \mathbf{w}^\top \mathbf{x} \tag{1}$$

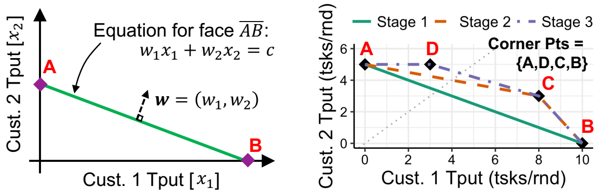
$$\text{s.t.} \quad c(\mathcal{P}_v) \le B \qquad \forall v \in V \tag{2}$$

$$\mathcal{P}_v \text{ is a valid path} \qquad \forall v \in V, \tag{3}$$

where $c(\cdot)$ specifies the time to complete a path. Equation (2) enforces that, for each vehicle, the time to execute the selected path does not exceed the budget. Equation (3) captures constraints that are specific to the vehicles (e.g., if vehicles must return to home at the end of each round) and customers (e.g., if tasks are only valid during specific windows during the scheduling horizon). The weighted VRP (also called the prize-collecting VRP) is NP-hard, but there are several known algorithms with optimality bounds [5, 44].

[4] §7 further evaluates the effectiveness of Mobius's algorithm for dynamic, real-world customer demand.

[5] $\mathbf{x}$ and $\mathbf{w}$ vary with each round τ. We drop the round index τ whenever there is no ambiguity about the current round.



(a) Computing **w** for a face.

(b) Finding support allocations.

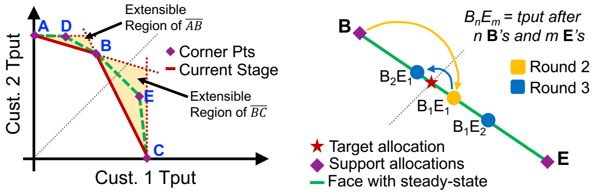
Figure 6: Using a blackbox VRP solver as a building block, Mobius runs an iterative search algorithm to find the support allocations.

Using weights to find the corner points. We can adjust the weight vector $\mathbf{w}$ in order to capture a bias toward a particular customer; $\mathbf{w}$ describes a direction in the customer throughput space, reflecting that bias. Fig. 6a visualizes $\mathbf{w}$ in a 2-D customer throughput space. A solver optimizing for Equation (1) searches for the schedule with the highest throughput in the direction of $\mathbf{w}$ [10], thus requiring the schedule to lie on the convex boundary. For example, $\mathbf{w_1} = (1,0)$ finds the schedule on the convex boundary that prioritizes customer 1 (i.e., along the x-axis), and $\mathbf{w_2} = (0,1)$ finds a schedule that prioritizes customer 2 (i.e., along the y-axis).

Searching on the convex boundary. Recall that, for strict fairness, the target allocation is the point where the $y = x$ line intersects the convex boundary for the current round (§4). At the start of the search, Mobius does not yet know the convex boundary, so it cannot know the target allocation. To find allocations on the convex boundary, Mobius employs an iterative search algorithm, analogous to binary search; in each stage, it tries to find a new allocation on the convex boundary in the direction of the $y = x$ line. Mobius begins the search with allocations along the customer axes. For two customers, it begins with weights $\mathbf{w_1}$ and $\mathbf{w_2}$ above, which gives two allocations on the convex boundary. In each stage of the search, Mobius computes a new weight vector, using allocations found on the convex boundary in the previous stage, in order to find a new allocation on the convex boundary. It terminates when no new allocation can be found. By searching in the right direction, Mobius only needs to compute a subset of corner points on the convex boundary.

To better illustrate the algorithm, consider the example in Fig. 6b, with 2 customers. Mobius starts the search by looking at the two extreme points on the customer 1 (x_1) and customer 2 (x_2) axes, which correspond to prioritizing all vehicles for either customer. So in stage 1, Mobius computes these schedules, using the weight vectors $\mathbf{w_1} = (1,0)$ and $\mathbf{w_2} = (0,1)$, which give the allocations A and B, respectively, in Fig. 6b. After stage 1, $\{A,B\}$ is the current set of corner points determining the convex boundary.

In the next stage, Mobius computes a new weight $\mathbf{w}$ to continue the search in the direction normal to $\overline{AB}$ (Fig. 6a). Let the equation for the face $\overline{AB}$ be $w_1x_1 + w_2x_2 = c$, where w_1, w_2, and c can be derived using the known solutions on the line, A and B. So, by invoking the VRP solver (Equation (1)) with $\mathbf{w} = (w_1, w_2)$, we try to find a schedule on the convex boundary, with the highest throughput in the direction normal to $\overline{AB}$. Let $\hat{x}_1$ and $\hat{x}_2$ be the throughputs for the schedule computed with weight $\mathbf{w}$. If $(\hat{x}_1, \hat{x}_2)$ lies above this line, i.e., $w_1\hat{x}_1 + w_2\hat{x}_2 > c$, then the point $(\hat{x}_1, \hat{x}_2)$ is a valid extension to the convex boundary. In this example, Mobius finds a new allocation C; so, the new set of corner points is $\{A,C,B\}$.



(a) Extensible region of face $\overline{BC}$.

(b) Throughputs in each round.

Figure 7: Mobius (a) finds the support allocations nearest the target allocation in each round, and (b) converges to the target allocation.

Notice that this extension in stage 2 creates two new faces on the convex boundary, $\overline{AC}$ and $\overline{CB}$. But, the $y = x$ line only passes through $\overline{AC}$. So, in stage 3, Mobius continues the search, extending $\overline{AC}$ by the computing the weights as described above (normal to $\overline{AC}$), and discovers a new allocation D. Finally, Mobius tries to extend the face $\overline{DC}$ because it intersects the $y = x$ line. It finds no valid extension, and so, it terminates its search on the face $\overline{DC}$, and returns the support allocations D and C.

Generalizing to more customers. Mobius computes a weight for each customer $k \in K$, i.e., $\mathbf{w} \in \mathbb{R}^{|K|}$. Faces on the convex boundary become $|K|$-dimensional hyperplanes, described by the equation $\sum_{k \in K} w_k x_k = c$. Mobius solves for $\mathbf{w}$ using the $|K|$ allocations that define each face, and finds $|K|$ support allocations at the end of the search. Recall from the example in §5.1 that each stage produced 2 new faces and that Mobius only continued the search by extending 1 face. With $|K|$ customers, even with $|K|$ new faces after each stage, Mobius only invokes the VRP solver once to continue the search. A naive algorithm, by contrast, would require $|K|$ calls to the VRP solver in each stage. Thus Mobius scales easily with more customers by pruning the search space efficiently.

5.2 Scheduling Over Rounds

In each round, Mobius finds $|K|$ support allocations, which determine the face of the convex boundary that contains the target allocation. It then selects a support allocation among these $|K|$ such that the per-customer long-term throughputs $\overline{x}_k(t)$ approach the target throughput. By tracking $\overline{x}_k(t)$ over many rounds, Mobius can select allocations that compensate for any unwanted bias introduced to some customer in a prior round.

Mathematically, to choose a schedule in round t, Mobius considers the effect of each support allocation $\mathbf{x}(t)$ on the average throughput $\overline{\mathbf{x}}(t+1)$. The average throughput is defined for each customer k as $\overline{x}_k(t+1) = \gamma_t x_k(t) + (1-\gamma_t)\overline{x}_k(t)$, where $\gamma_t = 1/(t+1)$. Mobius chooses $\mathbf{x}(t)$ such that $\overline{\mathbf{x}}(t+1)$ is closest to the $y = x$ line (in Euclidean distance).

5.3 Optimality of Mobius

Mobius is optimal in a round. We can prove that Mobius finds the support allocations nearest the target throughput (in Euclidean distance). We illustrate this through the example in Fig. 7a, where the corner points of the convex boundary are $\{A,D,B,E,C\}$, and B is closest to the target allocation. In the previous stage, Mobius discovered B, and it needs to pick one face to continue the search. The shaded yellow regions indicate the extensible regions of the two candidate faces

$\overline{AB}$ and $\overline{BC}$. The extensible region of a face describes the space of allocations that can be obtained by searching with the weight vector that defines that face, while maintaining a convex boundary (§5.1). Since Mobius finds a new allocation on the convex boundary in every stage of the search, no allocation can exist outside these regions; otherwise, the resulting set of discovered allocations would no longer be convex. Thus, the best face for Mobius to continue the search is indeed $\overline{BC}$, because its extensible region is the only one that may contain a better allocation closer to the $y=x$ line. Our technical report [6] contains a formal proof that the optimal support allocation (i.e., the allocation closest to the line $y=x$) is unique and that Mobius finds it.

Optimality over multiple rounds. Under a static task arrival model, we can show that the schedules computed by Mobius achieve throughputs that are optimal at the end of every round, i.e., the achieved throughput has the minimum distance possible to the target allocation after each round. This model assumes the convex boundary remains the same across rounds. One way to realize this is to require (i) the vehicles return to their starting locations at the end of each round, and (ii) all tasks are renewed at the beginning of each round. We make these simplifying assumptions only for ease of analysis; our evaluation in §7 does not use them.

We describe an intuition for this result below. [6] Per the static task arrival model, the convex boundary is the same in *each subsequent round*; therefore, Mobius finds the same support allocations in every round. By taking into account the long-term per-customer rates, $\overline{x}_k(t)$, Mobius oscillates between these support allocations in each round at the right frequency, such that $\overline{x}_k(t)\ \forall k \in K$ converges to the target allocation over multiple rounds. We illustrate this in Fig. 7b, which shows the support allocations B and E. The face $\overline{BE}$ contains the target allocation, denoted by the star. Because Mobius oscillates between B and E, the allocation $(\overline{x}_1(t), \overline{x}_2(t))$ must lie along $\overline{BE}$. Mobius chooses B in the first round because its throughput is closer than E to the target allocation. In the second round, it chooses E, moving the average throughput to B_1E_1. In the third round, Mobius chooses B, moving the average throughput to B_2E_1. Notice that if it had instead chosen E in the third round, the average throughput would be B_1E_2, which is further away from the target throughput. Thus, this myopic choice between B and E results in the closest solution to the target allocation after any number of rounds. Additionally, notice that the length of the jump (e.g., from B to B_1E_1 and from B_1E_1 to B_2E_1) decreases in each round; therefore, Mobius *converges* to the target throughput.

5.4 Implementation

We implement the core Mobius scheduling system in 2,300 lines of Go.[7] It plugs directly with external VRP solvers implemented in Python or C++ [25, 39]. Mobius exposes a simple, versatile interface to customers, which we call an interest map. An interest map consists of a list of desired tasks, where each task includes a geographical location, the time to complete the task once the vehicle has reached the location, and a task deadline (if applicable). In each round, Mobius gathers and merges interest maps from all customers, before computing a schedule. At the end of each round, it informs the customers of the tasks that have been completed, and customers can submit updated interest maps. Interest maps serve as an abstraction for Mobius to ingest and aggregate customer requests; however, the merged interest map is directly compatible with standard weighted VRP formulations [5, 19] without modification. Thus, Mobius acts as an interface between customers and vehicles, using a VRP solver as a primitive in its scheduling framework (Fig. 3).

Bootstrapping VRP solvers. Since the VRP is NP-hard [44], solvers resort to heuristics to optimize Equation (1). In practice, we find that state-of-the-art solvers do not compute optimal solutions; however, we can aid these solvers with initial schedules that the heuristics can improve upon. We warm-start the VRP solvers with initial schedules generated by the following policies: (i) maximizing throughput, (ii) dedicating vehicles (assuming a sufficient number of vehicles), and (iii) a greedy heuristic that maximizes our utility function (§6). [8] At the beginning of each round, Mobius builds a suite of warm start solutions. Then, prior to invoking the VRP solver with some weight vector $\mathbf{w}$, Mobius chooses the initial schedule from its warm start suite with the highest weighted throughput (i.e., objective of Equation (1)). Mobius also caches the schedules found from all invocations to the VRP solver (§5.1), to use for warm start throughout the round. Mobius parallelizes all independent calls to the VRP solver (e.g., when computing warm start schedules and when generating $|K|$ schedules to initialize the search along the convex boundary).

6 GENERALIZING TO α-FAIRNESS

The fairness objective we have considered so far aims to provide all customers with the same long-term throughput (maximizing the minimum throughput). However, an operator of a mobility platform may be willing to slightly relax their preference for fairness for a boost in throughput. To navigate throughput-fairness tradeoffs, we can generalize Mobius's algorithm (§5) to optimize for a general class of fairness objectives. We use the α-parametrized family of utility functions U_α, developed originally to characterize fairness in computer networks [31]:

$$U_\alpha(\mathbf{y}) = \sum_{k\in K} \frac{{y_k}^{1-\alpha}}{1-\alpha}, \tag{4}$$

where $\mathbf{y} \in \mathbb{R}^{|K|}$ and y_k is the throughput of customer k (either short-term x_k or long-term $\overline{x}_k$). U_α captures a general class of concave utility functions, where $\alpha \in \mathbb{R}_{\geq 0}$ controls the degree of fairness. For instance, when $\alpha = 0$, the utility simplifies to the throughput-maximizing objective defined in Equation 1 (assuming all customers have the same weight). By contrast, when $\alpha \longrightarrow \infty$, the objective becomes maximizing the minimum customer's throughput (i.e., max-min fairness). $\alpha = 1$[9] corresponds to proportional fairness, where the sum of log-throughputs of all customers is maximized; this ensures that no individual customer's throughput is completely starved.

Generalizing Mobius's search algorithm. When Mobius generalizes to α-fairness, the target allocation is no longer simply the point on the convex boundary that intersects the $y = x$ line. The target allocation is instead the allocation on the convex boundary with the greatest utility U_α. When searching the convex boundary in each round, Mobius determines which candidate face

[6] See our technical report [6] for a formal proof.

[7] github.com/mobius-scheduler/mobius

[8] Our technical report [6] includes a detailed description of this heuristic.

[9] U_α is not defined at $\alpha = 1$, so we take the limit as $\alpha \longrightarrow 1$.

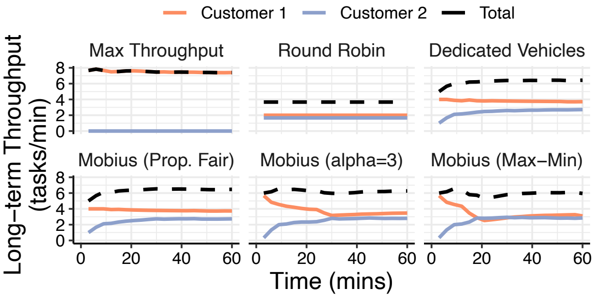


Figure 8: Mobius can tune its allocation to deliver proportional fairness ($\alpha = 1$) and max-min fairness (approximated with $\alpha = 100$).

contains the target throughput by using Lagrange Multipliers to find the point *along the face* [10] with the greatest utility. Once it finds each support allocation $\mathbf{x}$, Mobius incorporates the historical throughput $\overline{\mathbf{x}}$ to select the schedule with greatest cumulative utility $U_\alpha(\gamma_t \mathbf{x}(t+1) + (1-\gamma_t)\overline{\mathbf{x}}(t))$, where γ_t is as defined in §5.2.

An example. Fig. 8 shows a time-series chart of long-term customer and platform throughputs for the example described in §4.2. By adapting to different schedules on the convex boundary, Mobius converges to a fair allocation of rates without degrading total throughput. α allows Mobius to compute *expressive schedules*; for instance, $\alpha = 1$ strives to maximize total throughput without starving either customer. Additionally, Mobius (max-min)[11] converges to a fair allocation of long-term throughputs within 20 minutes.

7 REAL-WORLD EVALUATION

We evaluate Mobius using trace-driven emulation (§7.1) in two real-world mobility platforms. In §7.2, we apply Mobius to Lyft ridesharing in Manhattan and demonstrate that it scales to large online problems. In §7.3, we deploy Mobius on a shared aerial sensing system, involving multiple apps with diverse spatiotemporal preferences. Our evaluation focuses on answering the following questions:

- How does Mobius compare to traditional approaches in online scheduling for large-scale mobility problems?
- How robust is Mobius in the presence of dynamic spatiotemporal demand from customers?
- How can we tune Mobius's timescale of fairness?
- What other benefits does Mobius provide to customers, beyond optimizing per-customer throughputs?

7.1 Online Trace-Driven Emulation

We implement a trace-driven emulation framework to compare Mobius against other scheduling schemes, under the same real-world environment. This framework replays timestamped traces of requests submitted by each customer, by streaming tasks to the scheduler as they arrive, and sending task results back to the customer.

Capturing environment dynamics and uncertainty. To emulate dynamic customer demand, our emulation framework streams tasks according to the timestamps in the trace—so Mobius has no visibility into future tasks. To emulate uncertainty in customer demand, we cancel tasks that are not scheduled in 10 minutes. Additionally, the case studies in §7.2 and §7.3 consider scenarios where *at least* one customer is backlogged (defined in §2). If no customers are backlogged, then the platform can fulfill all tasks within the planning horizon, and the resulting schedule would have maximal throughput and fairness. Thus, the problems are only interesting when at least one customer is backlogged; Mobius is effective and required only in such situations.

Backend VRP solver. We use the Google OR-Tools package [39] as our backend weighted VRP solver (Equation (1)). OR-Tools is a popular package for solving combinatorial optimization problems, and supports a variety of VRP constraints, including budget, capacity, pickup/dropoff, and time windows. Our case studies involve VRPs with different sets of constraints. We run our experiments on an Amazon EC2 `c5.9xlarge` instance with 36 CPUs.

Baselines. In our experiments, we evaluate Mobius's throughput and fairness against two baseline routing algorithms: (i) a max throughput scheduler, and (ii) dedicated vehicles. The max throughput scheduler simply runs the backend VRP solver on the same input of customer tasks fed into Mobius for a round. This solution provides a benchmark on the platform capacity, and quantifies the maximum achievable total throughput. We compute the "dedicated vehicles" schedule by first distributing the vehicles evenly among all customers,[12] and then invoking the max throughput scheduler once for each customer. This solution provides a benchmark schedule that divides vehicle time equally among all customers. As shown in §2, round-robin scheduling achieves very low throughput; hence we omit it from the results in this section.

To the best of our knowledge, Mobius is the first algorithm that explicitly optimizes for customer fairness in mobility platforms. We considered evaluating Mobius by running a scheduler that optimizes throughput and fairness over a longer timescale using a mixed-integer linear program solver (e.g., Gurobi [25] or CPLEX [28]); however, this is not feasible in practice, because (i) customer demands arrive in a streaming fashion, and (ii) these solvers do not scale beyond tens of tasks [36]. Thus, we believe the baselines described above offer reasonable comparisons for Mobius.

Microbenchmarks. In addition to the real-world case studies (§7.2-§7.3), we also evaluate Mobius on microbenchmarks created from synthetic customer demand, including scenarios where Mobius is optimal (under the static task arrival model, §5.3). We compare Mobius against max throughput, dedicating vehicles, and round robin, and show, through controlled experiments, that (i) it provides provably good throughput and fairness for a variety of spatial demand patterns, (ii) it scales for different numbers of vehicles, (iii) it controls its timescale of fairness, and (iv) it can tune its fairness parameter α. We also report the runtime of Mobius in various environments. We include these results in our technical report [6].

7.2 Case Study: Lyft Ridesharing in Manhattan

Setting. Motivated by the issue of "destination discrimination" [35, 45, 49] discussed in §1, we consider a ridesharing service that receives requests from different neighborhoods (customers) in a large metro area. Some neighborhoods are easier to travel to than others, and rider demand out of a neighborhood can vary

[10] Our technical report [6] shows how to find the face containing the target throughput.
[11] Mobius approximates max-min fairness ($\alpha \to \infty$) with $\alpha = 100$.

[12] Dedicating vehicles is most suitable when the number of vehicles is a multiple of the number of customers.

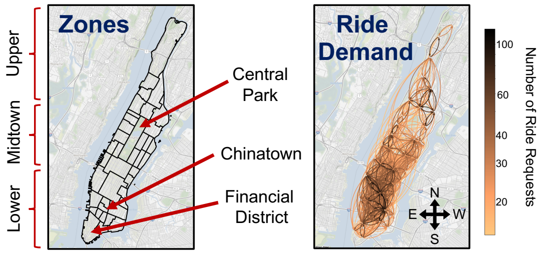


Figure 9: Maps of zones (customers) and demand in Manhattan, indicating skews in both spatial coverage and volume of ride requests.

with the time of day. We show that Mobius can guarantee a fair quality-of-service (in terms of max-min fair task fulfillment) to all neighborhoods throughout the course of a day, without significantly compromising throughput. We also show that, although it optimizes for an equal allocation of throughputs, Mobius does not degrade other quality-of-experience metrics, such as rider wait times. We further demonstrate that Mobius is a scalable *online* platform that generates schedules for a large city-scale problem.

Ridesharing demand. We use a 13-hour trace of 16,817 timestamped Lyft ride requests, published by the New York City Taxi and Limousine Commission, involving 40 neighborhoods (zones) in Manhattan over the course of a day [14]. Each request consists of a pickup and a dropoff zone, and we seek to provide pickups from all zones equitably. The map in Fig. 9 (left) demarcates the customer zones.

Fig. 9 (right) illustrates the scale of this scheduling problem. It visualizes traffic on the top 1,000 (out of 3,300) pickup-dropoff pairs; the color of each arrow indicates the volume of ride requests for that pickup-dropoff location. Notice that both the distance of rides and the volume of requests originating from zones vary vastly throughout the island. A significant fraction of requests arrive into and depart from Lower Manhattan. Some zones in Upper Manhattan have as few as 15 unique outbound trajectories, while other zones have hundreds.

Moreover, ridesharing demand varies significantly with the time of day. For instance, a busy zone near Midtown Manhattan sees the load vary from around 200 to 600 requests/hour, and a quiet zone near Central Park experiences a minimum load of 3 requests/hour and peak load of 24 requests/hour. Notice that the dynamic range of demand throughout the 13 hours also varies across zones.

Experiment setup. This ridesharing problem maps to the capacitated pickup/delivery VRP formulation [19]. It computes schedules that maximize the total number of completed rides, such that (i) a ride's pickup and dropoff are completed on the same vehicle, and (ii) each vehicle is completing at most one ride request at any point in time. We configure the solver to retrieve real-time traffic-aware travel time estimates from the Google Maps API [24], and we constrain OR-Tools to report a solution within 3 minutes.

We use the trace described above in our emulation framework (§7.1). We compute schedules for a fleet of 200 vehicles.[13] In order to ensure that the schedules are not myopic, we plan our routes with 45-minute horizons; however, to reduce rider wait times, we recompute the schedule every 10 minutes, while ensuring that we honor any requests that we have already committed to in the schedule. We assume that riders cancel requests that are not incorporated into a schedule within 10 minutes of the request time.

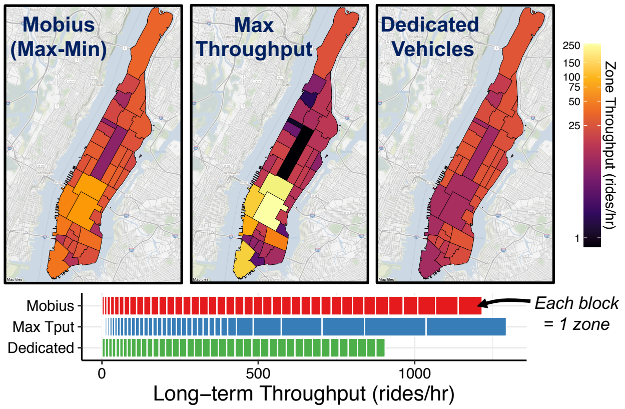


Figure 10: Long-term throughputs for zones in Manhattan after 13 hours. A good scheduler should have a stacked plot with large evenly-sized blocks, and a map with bright (high throughput) and homogeneous (fair) colors across zones.

Fairness with high vehicle utilization. Since Mobius plans continuously, having several allocations on the convex boundary at its disposal, we expect it to converge to a fair allocation of rates, despite the skew in demand. Fig. 10 shows the long-term throughputs achieved for each zone by different scheduling algorithms, after 13 hours. The color of each zone in the map indicates that zone's throughput. Bright colors correspond to high throughput, and a homogeneous mix of colors indicates a fair allocation. Beneath the maps, we also stack the zone throughputs to indicate how each scheduler divides up the total platform throughput across the zones; ideally we would like large, evenly-sized blocks.

The max throughput scheduler divides the platform throughput most unevenly across zones. In particular, we see that while it serves nearly 200 rides/hour out of the Financial District (Lower Manhattan), it virtually starves zones near Central Park. From the demand map (Fig. 9), notice that (i) a majority of rides originate from Lower Manhattan, and (ii) most of these trips are destined for neighboring zones. Thus, the best policy to maximize the total number of trips completed is to stay in Lower Manhattan, which is what the max throughput scheduler does.

The bar chart indicates that dedicating 5 vehicles to each zone results in 40% lower platform throughput than the max throughput scheduler. This is because a heterogeneous demand across zones cannot be effectively satisfied by an equal division of resources (vehicles). Nevertheless, Fig. 10 shows that this scheduler shares the platform throughput most evenly across zones. The division of per-zone throughputs is not perfectly even, in spite of dedicating an equal number of vehicles, because (i) ride requests from different zones can have different trip lengths, and (ii) some zones have inherently low demand and do not backlog the system, leaving some vehicles idle.

By contrast, Mobius strikes the best balance between throughput and fairness. It achieves roughly equal zone throughputs, while compromising only 10% of the maximum platform throughput. Compared to dedicating vehicles, we see, from the map, that Mobius achieves higher throughput for most zones by identifying an incentive to chain requests from different zones. For example,

[13] The number of vehicles does not matter, since we compare Mobius to the platform capacity (from the max throughput scheduler).

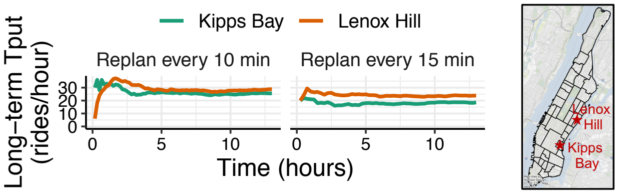


Figure 11: Time series of long-term throughputs for two zones for different replanning horizons. Frequent replanning ensures fairness (equal throughputs) at shorter timescales.

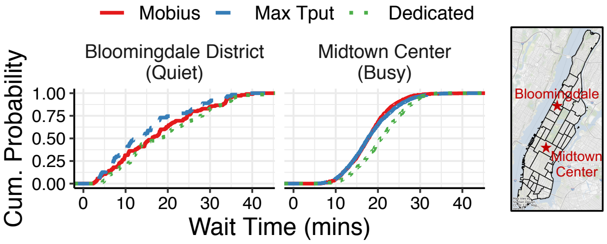


Figure 12: Distributions of rider wait times for two zones. Even though Mobius compromises some throughput for fairness, it delivers similar wait times as the max throughput scheduler.

Mobius combines two requests from different zones into the same trip, when the dropoff of the first request is close to the pickup of the second request. While this helps improve efficiency, Mobius also prioritizes pickups from zones with a historically low throughput to ensure fairness across zones. This ridesharing simulation reveals that it is possible to achieve a fair allocation of rates in a practical setting *without* significantly degrading platform throughput.

Controlling the timescale of fairness. Mobius's replanning interval controls the timescale over which it is fair. The more often that Mobius replans, the more up-to-date its record of long-term customer throughputs; Mobius can then adapt to short-term unfairness quickly by finding a more suitable schedule on the convex boundary. Recall that when replanning frequently, the convex boundary does not change drastically between scheduling intervals (§4.2), if the spatial distribution of tasks do not change rapidly with time. So, in practice, we do not expect to deviate far from the ideal target throughput. Fig. 11 shows the long-term throughputs achieved for two zones, for replanning timescales of 10 minutes and 15 minutes. Mobius equalizes throughputs better when it replans more frequently.

Rider wait times. Platform operators prefer high throughput schedules because they translate directly to high revenue; low throughput would lead to more cancelled rides. While Fig. 10 demonstrates that Mobius is fair without degrading throughput, we would like to know if optimizing for fairness impacts rider wait time (i.e., the time between requesting a ride and being picked up).

Fig. 12 compares the distributions of rider wait times for rides originating from Bloomingdale District (a quiet neighborhood west of Central Park) and from Midtown Center (a busy district near Times Square). We compute wait times are only for fulfilled tasks. Notice that in both zones—with two very different demand patterns—the distribution of wait times for Mobius is comparable to that of the max throughput scheduler.

We observe that the wait times in the quiet zone are slightly higher for Mobius (average of 17 minutes, compared with 15 minutes for max throughput). This is because the wait times for Mobius are computed for significantly more tasks (Mobius fulfills 66.7% more ride requests than does max throughput). The schedule that dedicates vehicles sees higher wait times than Mobius, especially when rides originate from a busy zone (e.g., Midtown Center), since vehicles would be idle until they return to their assigned zone to pick up a new rider.

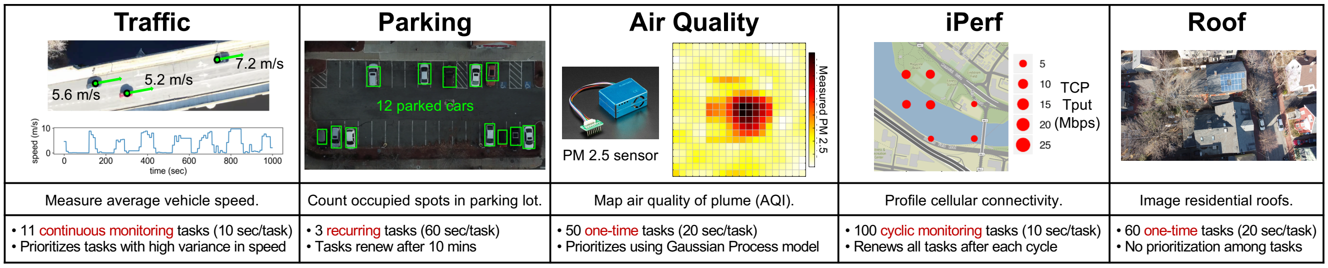
Scalability. This case study demonstrates that Mobius is practical at an urban scale. In fact, when scheduling its fleet of taxis, New York City's Yellow Cab restricts its scheduling region to Manhattan and organizes its requests according to approximately 40 taxi zones [7, 13]. In our experiments, the backend VRP solver (i.e., max throughput scheduler) computes each 45-minute schedule in 3 minutes (capped by the timeout). We observe that Mobius takes 5-6 minutes; Mobius sees a speedup by (i) parallelizing calls to the VRP solver and (ii) warm-starting the VRP solver with initial schedules (§5.4). These optimizations help Mobius easily scale to tens of thousands of tasks. We believe we can further improve the speed by leveraging parallelism in the backend VRP solver [43] (OR-Tools does not expose a multi-threaded solver).

7.3 Case Study: Shared Aerial Sensing Platform

Setting. The recent proliferation of commodity drones has generated an increased interest in the development of aerial sensing and data collection applications [2, 4, 16, 20, 33, 34], as well as general-purpose drone orchestration platforms [26, 37, 40]. An emerging mobility platform is a drones-as-a-service system [21, 27, 32, 46, 48], where developers submit apps to a platform that deploys these app tasks on a shared fleet of drones. App (customer) semantics in a drone sensing platform can show significant heterogeneity in both space and time. To ensure a satisfactory quality-of-service for all applications, a scheduler must not only efficiently multiplex tasks from different applications in each flight (typically constrained to 20 minutes due to the battery life [17]), but also share task completion throughput equitably across apps. Since apps can be reactive (i.e., sensing preferences change as apps receive measured data), Mobius must additionally provide a sustained rate-of-progress to each app, as opposed to "bursty" throughput.

Sensing apps. We implement 5 popular urban sensing apps to evaluate Mobius in this drones-as-a-service context, summarized in Fig. 13. Fig. 14 depicts the locations for the sensing tasks submitted by each app. We describe each app below:

- The *Traffic app* continuously monitors road traffic congestion over 11 contiguous segments of road in an urban area. To measure average vehicle speed, it collects 10-second video clips at each road segment, detects all cars using YOLOv3 [42], and tracks the trajectory [11] of each vehicle. After gathering multiple initial samples at all 11 locations, the app prioritizes the locations with the highest variance in speed, in order to collapse uncertainty in its overall estimates of road congestion.
- The *Parking app* counts parked cars at 3 sites, by monitoring each lot for 1 minute; to maintain fresh estimates of counts, this app renews these 3 tasks after 10 minutes.
- The *Air Quality app* measures PM2.5 concentration around a plume [1], submitting a candidate list of 100 one-time sampling locations. This app is also reactive; on receiving a measurement, it updates a Gaussian Process model [41] and cancels any unfulfilled tasks with high predicted accuracy.



Traffic	Parking	Air Quality	iPerf	Roof
Measure average vehicle speed.	Count occupied spots in parking lot.	Map air quality of plume (AQI).	Profile cellular connectivity.	Image residential roofs.
• 11 continuous monitoring tasks (10 sec/task) • Prioritizes tasks with high variance in speed	• 3 recurring tasks (60 sec/task) • Tasks renew after 10 mins	• 50 one-time tasks (20 sec/task) • Prioritizes using Gaussian Process model	• 100 cyclic monitoring tasks (10 sec/task) • Renews all tasks after each cycle	• 60 one-time tasks (20 sec/task) • No prioritization among tasks

Figure 13: Summary of aerial sensing applications, which span a variety of spatial demand and reactive/continuous sensing preferences. We collected ground truth data for each of these applications using real drones, and created traces to evaluate Mobius.

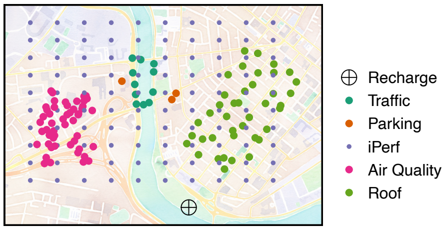


Figure 14: Map of tasks for 5 aerial sensing apps, spanning a 1 square mile area in Cambridge, MA. Mobius replans every 5 minutes, in order to incorporate new requests. Each drone returns to recharge every 15 minutes.

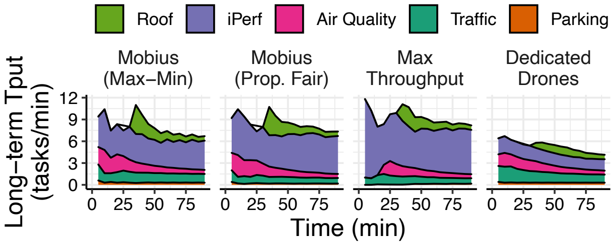


Figure 15: Long-term throughputs achieved over 90 minutes. Mobius achieves high throughput and best shares it amongst the apps.

- The *iPerf app* builds a map of cellular coverage in the air, by profiling throughput at 100 spatially-dispersed locations. It renews all tasks after each cycle of 100 measurements is complete.
- The *Roof app* submits 60 one-time tasks to image roofs over a residential area.

Notice that these apps collectively have a variety of spatiotemporal characteristics. For instance, the Traffic app changes its requests with time, based on the uncertainty in speed estimates and the freshness in measurements. By contrast, the Air Quality app changes its requests with space, using a statistical model to collapse uncertainty in a task based on nearby measurements. The iPerf app has no temporal preferences, and instead functions as a "free-riding" app that gathers quick measurements over a large area.

Ground-truth data collection. To run our drones-as-a-service platform on real-world sensor data, which is critical to the performance of the reactive and continuous monitoring apps, we separately gather 90 minutes of ground-truth data for each app, using real drones. This gives us a trace of timestamped measurement values of each app. We then use our trace-driven emulation framework (§7.1) to evaluate different scheduling algorithms. Fig. 13 shows highlights from our data collection. For example, to collect ground-truth for the Traffic app, we instrument 6 DJI Mavic Pros [17] to continuously gather video and track cars over the 11 measurement locations (Fig. 14) for 90 minutes. Similarly, for the iPerf and Air Quality apps, we program a DJI F450 drone [18] equipped with an LTE dongle and a PM2.5 sensor [1] to gather measurements at their respective measurement locations. We instrument our drone to communicate its location, battery status, and measurement data to a dashboard hosted on an EC2 instance, from which we observe the drone's progress on our laptop.

Experiment setup. We configure our backend solver to estimate travel time as the Euclidean distance between the sensing tasks plus the sensing time for the destination task. In order to be sufficiently reactive to the Traffic and Air Quality apps, we schedule in 5-minute rounds, and require that the drones return to recharge their batteries every 15 minutes. We run our trace-driven emulation framework with 5 drones. Additionally, we configure the Roof app to join the system after 30 minutes.

High throughput, high fairness. To understand how Mobius divides the platform throughput, we show the long-term throughput for each app over 90 minutes in Fig. 15. Mobius (max-min) achieves 55% more throughput than dedicating drones and only 15% less throughput than maximizing throughput. Mobius with a proportional fairness objective similarly outperforms max throughput and dedicated vehicles in navigating the throughput-fairness tradeoff. Note that the throughputs of the Air Quality and Roof apps decay with time, after their one-time tasks are fulfilled.

Because these apps have variable demand (e.g., 100 tasks for iPerf and 3 tasks for Parking), studying throughput is not sufficient. Hence, we plot the tasks completed as a fraction of demand for each app in Fig. 16. Notice that, under Mobius, even the most starved app (iPerf) completes nearly 34% of its tasks; by contrast, max throughput and dedicated drones deliver worst-case task completions of 30% and 13%, respectively. Even though dedicating drones guarantees equal drone time for each app, it is extremely unfair toward apps with higher demand or more spatially-distributed tasks.

Impacts of sensing and travel times. Fig. 14 would suggest that the Air Quality and Roof tasks are easier to service, since their tasks are more spatially concentrated; however, their tasks take 20 seconds each (Fig. 13). The max throughput scheduler understands this tradeoff in terms of maximizing throughput, and thus prioritizes the iPerf app, since its 10-second tasks (Fig. 13) are cheap to complete. By contrast, Mobius additionally understands how to navigate this tradeoff in terms of fairness; for instance, it forgoes some iPerf tasks to complete more 20-second AQI measurements.

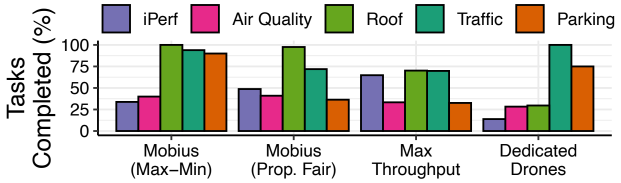


Figure 16: Percentage of tasks completed per app. Mobius fulfills nearly all requests for the Traffic and Parking apps, before allocating "excess" vehicle time to the more backlogged apps.

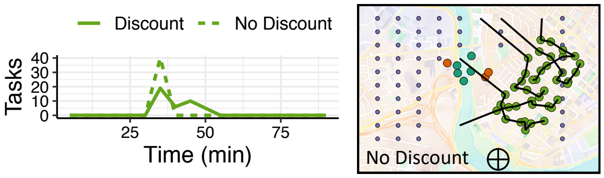


Figure 17: Discounting long-term throughput allows Mobius to gradually respond to the sudden presence of the transient Roof app, instead of dedicating all drones to it.

Reliable rate-of-progress. In enforcing either proportional or max-min fairness, Mobius does not starve any app, at any instant of time. Indeed, Fig. 15 indicates that Mobius delivers a reliable rate-of-progress to the Air Quality app, gradually giving it roughly 3 tasks/min over the first 20 minutes. By contrast, the max throughput scheduler is more "bursty", and only services this app after 20 minutes. As a result, we find that, with Mobius, the root-mean-square error (RMSE) of the Gaussian Process model for the air quality drops more rapidly.

Catering to transient apps. Recall that the Roof app joins the platform after 30 minutes. Fig. 15 indicates that Mobius rapidly adapts to this change in demand with a spike in throughput for the Roof app at the cost of lower throughput for the iPerf and Air Quality apps. Notice that this spike in Mobius's schedule is larger in magnitude than the one in the max throughput schedule. This is because Mobius realizes that, when the Roof app joins, it has no accumulated throughput, while other apps have amassed higher throughput from living in the system for longer. Fig. 17 (right) plots the routes for all 5 drones during minutes 30-35; all drones immediately flock to the Roof app. With Mobius, an operator can choose to respond to the arrival of new apps by discounting throughput accumulated in prior rounds. Fig. 17 (left) shows how Mobius can control the Roof app's rate of task fulfillment, with a discount factor of 0.1.

8 RELATED WORK

Shared mobility and sensing platforms. Ridesharing platforms rely on different flavors of the VRP; these systems have typically been interested in maximizing profit (i.e., throughput) [3, 12], minimizing the size of the fleet [47], and planning in an online fashion [7]. Similarly, there has been a large amount of recent work on drones-as-a-service platforms, which have primarily addressed challenges surrounding data acquisition [46], multi-tenancy and security [27], and programming interfaces [26, 37]. All of these systems use a throughput-maximizing algorithm under the hood. Mobius is motivated by the advent of *customer-centric* mobility platforms in a variety of domains, where guarantees on quality-of-service to customers are paramount to the viability [45] of these services [35].

Vehicle routing problem. The VRP has been extensively studied by the Operations Research community [44]. Many variants of the problem have been considered, ranging from the budget-constrained VRP [5], capacitated VRP [23], VRP with time windows [19], predictive routing under stochastic demands [8, 26], etc. Prior work has extended the VRP to consider multiple objectives, such as minimizing the variance in vehicle travel time or tasks completed by each vehicle [29]. These load balancing objectives, however, do not consider customer-level fairness, which is the focus of Mobius. Moreover, Mobius abstracts out fairness from the underlying vehicle scheduling problem, making its techniques complementary to the large body of work on the VRP and its variants.

Fair resource allocation in computer systems. Our approach to formalizing throughput and fairness in mobile task fulfillment is inspired by α-fair bandwidth allocation in computer networks [31, 38]. However, as noted in §1, mobility platforms introduce new challenges around attributing cost to serve customers, that do not arise when addressing fairness in switch scheduling [15], congestion control [30], and multi-resource compute environments [22]. Mobius develops a novel set of techniques to address these challenges.

9 CONCLUSION

We developed Mobius, a scheduling system that can deliver both high throughput and fairness in shared mobility platforms. Mobius uses the insight that, when operating over rounds, scheduling on the convex boundary of feasible allocations, as opposed to the Pareto frontier, provably improves on fairness with time. We showed that Mobius can handle a variety of spatial and temporal demand distributions, and that it consistently outperforms baselines that aim to maximize throughput or achieve fairness at smaller timescales. Additionally, through real-world ridesharing and aerial sensing case studies, we demonstrated that Mobius is versatile and scalable.

There are several opportunities for extending the capabilities of Mobius. First, Mobius assumes that customers are not adversarial. Developing strategyproof mechanisms that incentivize truthful reporting of tasks by customers is an open problem. Second, we design Mobius to only balance customer throughputs. We believe the optimization techniques we developed (§5) can be extended to support other platform objectives, such as task latency, vehicle revenue, and driver fairness. Finally, incorporating predictive scheduling, where the platform can strategically position vehicles in anticipation of future tasks, is an interesting direction for future work, as it can further increase platform throughput.

ACKNOWLEDGMENTS

We thank Songtao He, Favyen Bastani, Sam Madden, our shepherd, and the anonymous MobiSys reviewers for their helpful discussions and thoughtful feedback. This research was supported in part by the NSF under Graduate Research Fellowship grant #2389237. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The NASA University Leadership Initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity.

REFERENCES

[1] Adafruit. Pm2.5 air quality sensor. https://learn.adafruit.com/pm25-air-quality-sensor.

[2] R. S. Allison, J. M. Johnston, G. Craig, and S. Jennings. Airborne optical and thermal remote sensing for wildfire detection and monitoring. *Sensors*, 16(8):1310, 2016.

[3] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. Natl. Acad. Sci. USA*, 114(3):462–467, 2017.

[4] A. Amarasinghe, C. Suduwella, C. Elvitigala, L. Niroshan, R. J. Amaraweera, K. Gunawardana, P. Kumarasinghe, K. D. Zoysa, and C. Keppetiyagama. A machine learning approach for identifying mosquito breeding sites via drone images. In M. R. Eskicioglu, editor, *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, SenSys 2017, Delft, Netherlands, November 06-08, 2017*, pages 68:1–68:2. ACM, 2017.

[5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[6] A. Balasingam, K. Gopalakrishnan, R. Mittal, V. Arun, A. Saeed, M. Alizadeh, H. Balakrishnan, and H. Balakrishnan. Throughput-fairness tradeoffs in mobility platforms. https://arxiv.org/abs/2105.11999, 2021.

[7] D. Bertsimas, P. Jaillet, and S. Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Oper. Res.*, 67(1):143–162, 2019.

[8] D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Oper. Res.*, 40(3):574–585, 1992.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*, chapter Convex Sets, page 21–66. Cambridge University Press, 2004.

[10] S. Boyd and L. Vandenberghe. *Convex Optimization*, chapter Convex Optimization Problems, pages 146–148. Cambridge University Press, 2004.

[11] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] A. Braverman, J. G. Dai, X. Liu, and L. Ying. Empty-car routing in ridesharing systems. *Oper. Res.*, 67(5):1437–1452, 2019.

[13] N. T. . L. Commission. Taxi & limousine commission - homepage. https://www1.nyc.gov/site/tlc/index.page.

[14] N. T. . L. Commission. Tlc trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[15] A. J. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In L. H. Landweber, editor, *SIGCOMM '89, Proceedings of the ACM Symposium on Communications Architectures & Protocols, Austin, TX, USA, September 19-22, 1989*, pages 1–12. ACM, 1989.

[16] A. Dhekne, A. Chakraborty, K. Sundaresan, and S. Rangarajan. Trackio: Tracking first responders inside-out. In J. R. Lorch and M. Yu, editors, *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, pages 751–764. USENIX Association, 2019.

[17] DJI. Dji - official website. https://www.dji.com/.

[18] DJI. Flame wheel arf kit: Multirotor flying platform for entertaining and amateur ap. https://www.dji.com/flame-wheel-arf.

[19] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[20] J. C. L. Fargeas, P. T. Kabamba, and A. R. Girard. Cooperative surveillance and pursuit using unmanned aerial vehicles and unattended ground sensors. *Sensors*, 15(1):1365–1388, 2015.

[21] FlytBase. Flytos: Operating system for drones. https://flytbase.com/flytos/.

[22] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In D. G. Andersen and S. Ratnasamy, editors, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011.

[23] B. Golden, S. Raghavan, and E. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Operations Research/Computer Science Interfaces Series. Springer US, 2008.

[24] Google, Inc. Google maps platform | distance matrix api. https://developers.google.com/maps/documentation/distance-matrix/overview, 2020.

[25] Gurobi Optimization, LLC. Gurobi optimizer reference manual. http://www.gurobi.com", 2020.

[26] S. He, F. Bastani, A. Balasingam, K. Gopalakrishnan, Z. Jiang, M. Alizadeh, H. Balakrishnan, M. J. Cafarella, T. Kraska, and S. Madden. Beecluster: drone orchestration via predictive optimization. In E. de Lara, I. Mohomed, J. Nieh, and E. M. Belding, editors, *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 299–311. ACM, 2020.

[27] A. V. Hof and J. Nieh. Androne: Virtual drone computing in the cloud. In G. Candea, R. van Renesse, and C. Fetzer, editors, *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, pages 6:1–6:16. ACM, 2019.

[28] IBM. Ibm cplex optimizer. https://www.ibm.com/analytics/cplex-optimizer, 2021.

[29] N. Jozefowiez, F. Semet, and E. Talbi. Multi-objective vehicle routing problems. *Eur. J. Oper. Res.*, 189(2):293–309, 2008.

[30] F. Kelly. Fairness and stability of end-to-end congestion control. *Eur. J. Control*, 9(2-3):159–176, 2003.

[31] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *J. Oper. Res. Soc.*, 49(3):237–252, 1998.

[32] S. Mahmoud, N. Mohamed, and J. Al-Jaroodi. Integrating uavs into the cloud using the concept of the web of things. *J. Robotics*, 2015:631420:1–631420:10, 2015.

[33] W. Mao, Z. Zhang, L. Qiu, J. He, Y. Cui, and S. Yun. Indoor follow me drone. In T. Choudhury, S. Y. Ko, A. Campbell, and D. Ganesan, editors, *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17, Niagara Falls, NY, USA, June 19-23, 2017*, pages 345–358. ACM, 2017.

[34] V. Mersheeva and G. Friedrich. Multi-uav monitoring with priorities and limited energy resources. In R. I. Brafman, C. Domshlak, P. Haslum, and S. Zilberstein, editors, *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, pages 347–356. AAAI Press, 2015.

[35] S. Middleton. Discrimination, Regulation, and Design in Ridehailing. Master's thesis, Massachusetts Institute of Technology, 5 2018.

[36] J. C. Molina, I. Eguia, J. Racero, and F. Guerrero. Multi-objective vehicle routing problem with cost and emission functions. *Procedia - Social and Behavioral Sciences*, 160:254–263, 2014. XI Congreso de Ingenieria del Transporte (CIT 2014).

[37] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level programming of drone sensor networks. In Á. Lédeczi, P. Dutta, and C. Lu, editors, *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, SenSys '14, Memphis, Tennessee, USA, November 3-6, 2014*, pages 177–190. ACM, 2014.

[38] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti. Numfabric: Fast and flexible bandwidth allocation in datacenters. In M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, editors, *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 188–201. ACM, 2016.

[39] L. Perron and V. Furnon. Or-tools. https://developers.google.com/optimization/routing/vrp.

[40] R. Petrolo, Y. Lin, and E. W. Knightly. ASTRO: autonomous, sensing, and tetherless networked drones. In *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, DroNet@MobiSys 2018, Munich, Germany, June 10-15, 2018*, pages 1–6. ACM, 2018.

[41] C. E. Rasmussen. Gaussian processes in machine learning. In O. Bousquet, U. von Luxburg, and G. Rätsch, editors, *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 63–71. Springer, 2003.

[42] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[43] É. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[44] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*, volume 9 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2002.

[45] Uber Technologies. What is destination discrimination? https://help.uber.com/driving-and-delivering/article/what-is-destination-discrimination?nodeId=9bde02cc-3d43-4837-9384-d28c57755fd9, 2021.

[46] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. N. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman. Farmbeats: An iot platform for data-driven agriculture. In A. Akella and J. Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 515–529. USENIX Association, 2017.

[47] M. M. Vazifeh, P. Santi, G. Resta, S. H. Strogatz, and C. Ratti. Addressing the minimum fleet problem in on-demand urban mobility. *Nat.*, 557(7706):534–538, 2018.

[48] J. Yapp, R. Seker, and R. F. Babiceanu. UAV as a service: A network simulation environment to identify performance and security issues for commercial uavs in a coordinated, cooperative environment. In J. Hodický, editor, *Modelling and Simulation for Autonomous Systems - Third International Workshop, MESAS 2016, Rome, Italy, June 15-16, 2016, Revised Selected Papers*, volume 9991 of *Lecture Notes in Computer Science*, pages 347–355, 2016.

[49] D. Zipper. Did uber just enable discrimination by destination? https://www.bloomberg.com/news/articles/2019-12-11/the-discrimination-risk-in-uber-s-new-driver-rule, 2019.