LUISS

Department
of Business & Management

Course of Social Network Analysis

A Study on "Empty-Car Routing in Ridesharing Systems" and Simulation on Chicago Transportation Network Dataset

Prof. Scarsini Marco
SUPERVISOR

ID No. 233741
CANDIDATE

Academic Year 2020/2021

Index

1 Introduction

The Ride-hailing & Taxi market segment includes all online and offline booking channels that connect passengers and drivers. This includes traditional taxi services that can be booked by phone, Transportation Network Companies (TNCs) that offer rides in private vehicles, as well as Ride Pooling services. Ride-hailing marketplaces match millions of riders and drivers every day. According to *statista.com* (2021), The global ride-sharing market is expected to grow to by more than 50 percent between 2020 and 2021. The market value is expected to amount to around 117 billion U.S. dollars in 2021. DiDi, Uber, and Lyft are among the key companies in this industry. They definitely represent disruptive players in the transportation industry by matching the supply of drivers with the demand for rides more efficiently than traditional taxi services. Yet, they do not employ any drivers, but rather they operate as two-sided markets between riders and independent users of the platform who serve as drivers. In this characteristic of the industry lie two of the hardest challenges for such platforms, namely handling the decentralized nature of the system and the disparity between the flow of passengers requesting a ride in some region of the city and the available drivers willing to accept them. A crucial objective for ride-sharing platform is to create incentive for drivers to relocate to areas of the city where new passengers arrive more frequently.

In this project, we focus on the study by Braverman et al. (2019), who approach this task by considering a specific centralized control mechanism, the "Empty-car routing", whose aim is to maximize the availability of empty cars in an area when a passenger arrives. In order to mitigate the aforementioned demand-supply imbalances that arise in such systems, the authors consider a controller which is able to move empty cars to regions where they are most needed so as to minimize unfulfilled requests by riders, who are impatient and choose a different transportation mean if they are not served in a short time. It is important to notice that, being this a *centralized mechanism*, in which drivers are supposed to obey the directions given by the controller, it is not directly applicable as it is by companies like Uber, Lyft and others, as they deal instead with decentralized systems, where drivers act independently and all the platform can do is to create effective incentive for them to relocate where most needed. However, Braverman et al. (2019) state that their study can be a best-

case performance benchmark against which to compare how well decentralized mechanisms perform. Moreover, it might be insightful to compute the difference in revenues for drivers between the case when they are free to relocate on their own and the "Empty-car routing" system, so that the platform could use part of that difference to create further incentive for drivers to behave as expected. Moreover, they experiment their theoretical findings on real-world data by *Didi Chuxing* (2019) Chuxing from a Chinese city and evaluate the performance of the resulting policy.

In this project we first illustrate the work by Braverman et al. (2019), providing explanation of the analytical problem they solved and the required tools. Our main purpose is to reproduce their analysis on a different dataset, namely "Chicago Transportation Network Providers", reporting the full code for the data manipulation and solving procedure of the problem, and thus, provide the results of our analyses and explanation for the differences with those of Braverman et al. (2019).

2 Literature Review

Academic activity on ride-sharing systems has increased significantly in the recent years due to the exceptional growth of the related industry. The complexity of the optimization algorithms for such systems offers a variety of different potential approaches and many papers examine the operations of ride-hailing platforms from diverse perspectives: some studies focus on the *matching* strategies, which involve finding and assigning the most suitable driver to a rider's request; for instance, Curry et al. (2019) model the matching process of raid-sharing systems as a Markov Chain encapsulating the geographic mobility of supply and demand over time. They account for the previous work on the problem, which considered the matching policies in the limit of such Markov chains, and analyze under what conditions this limit is valid. Özkan and Ward (2020) consider the two-sided dynamic matching problem (where multiple items arrive at both sides randomly with potentially time-varying rates) which encapsulates the framework of ride-sharing systems, proposing matching policies based on a continuous linear program (CLP) and proving their optimal performance under some conditions. Housni et al. (2021) address the impact of the matching policy on the future demand/supply composition by studying a two-stages matching

problem that aims at minimizing the cost of the first-stage matching (intended as the proportion on requests that remain unmatched) and of the worst-case scenario for the second-stage matching, given a set of predefined possibilities. The motivation comes from the sub-optimal performance of an approach which matches the current requests ignoring future uncertainty.

A second viewpoint of the problem relates to the optimization of the *pricing* strategies to mitigate the structural demand/supply imbalances in the geospatial network that the problem at stake presents. Such approach is undertaken by Banerjee, Freund, et al. (2017), who deal with the complex network externalities of shared vehicle systems optimization proposing a number of pricing and other control policies under a convex approximation, which they call "elevated flow relaxation", of the dynamic maximization of the long-run average performance, taking into account throughput (proportion of matched requests), social welfare and platform's revenue. Bimpikis et al. (2019) study the impact of demand pattern's spatial structure on platforms' profits and the resulting consumer surplus, focusing on the adequate pricing strategies to adopt depending on where the rides originate. Garg et al. (2021) consider the driver-side payment mechanisms for Ride-hailing platforms, investigating how dynamic (surge) pricing affects drivers' revenues and their strategies of repositioning to maximize such revenues. They focus on the theoretical grounds supporting the so-called additive surge (applied by Uber) and its advantages in terms of incentive compatibility with respect to the multiplicative surge. Besbes et al. (2021) consider the surge pricing technique and its implications on the reaction of strategic supply drivers (supply units) and price-sensitive riders (demand). They show how a platform may optimally set prices across the space in reaction to a localized demand shock to encourage drivers to relocate. Also Ong et al. (2021) study a recent shift in the way a ride sharing company creates incentives for drivers to relocate; in particular, they explain how Lyft has brought a significant change in their operation by introducing the "Personal Power Zone" (PPZ), a product which helps drivers discern the most profitable areas of the city in real time and suggests them the strategy to maximize their revenue. The authors give the mathematical explanation on how the new additive surge pricing creates a significantly higher incentive to relocate for drivers than the previously used multiplicative pricing (which Lyft used to call "Prime Time"). Their work also focuses on the paradigm shift in business model that Lyft underwent, depict-

ing a more centralized environment based on optimizing driver experience and revenues, so as to bring beneficial effects for the business in general (increased bookings) and under the drivers' and riders' experience viewpoint (e.g. reduction in pick-up times and dropped requests).

Similar to the approach of Banerjee, Freund, et al. (2017), other works concentrate upon different platform control mechanisms that can optimize various objective variables, such as consumer surplus, platform's or drivers' revenue. This is the case for Afeche et al. (2018), who face the problem of matching service supply (drivers) with demand (riders) over two-location spatial network, focusing on the impact of two platform control capabilities, namely "demand-side admission control" and "supply-side capacity repositioning". The former is the possibility for the platform to accept or reject requests based on some parameters, the latter involves redirecting drivers in different areas of the city to deal with demand/supply imbalances. Their findings show that it may be optimal for the platform to reject rider demand even in over-supplied areas, to encourage driver movement. Banerjee, Kanoria, et al. (2020) deal with the demand-supply asymmetry of ride-sharing systems (and other applications) through dynamic assignment control of a closed queueing network, namely with a fixed number of circulating supply units. They propose a class of state-dependent control policies which are proven to achieve optimal performance under some conditions. They incorporate travel delays as well and consider other potential applications of their model.

It might be important to specify that the aforementioned approaches are by no means to be considered disconnected from each other, because, ultimately, they all need to be implemented by ride sharing platforms in a multidimensional strategy aimed at maximizing their earnings, drivers' revenue and the amount of rides provided. In a rather theoretical fashion, this last point is addressed by Özkan (2020) by answering the following question: "Is matching optimization necessary?". Motivated by the wide presence of previous work where pricing policies are optimized under fixed matching policies, this paper shows that optimizing one decision (either pricing or matching) keeping the other fixed is not optimal, while a joint pricing and matching optimization strategy can lead to significant performance improvements.

3 Model

The objectives of this capstone project consist in understanding, reporting analyzing and replicating the content of the paper *"Empty-car Routing in Ridesharing Systems"* by Braverman et al. (2019). In this section we focus on the model employed by the authors, the tools needed to have a proper understanding of their research, and their main results.

3.1 General Problem

The setting involves a city divided in $r > 0$ different regions, where a fixed number of cars ($N > 0$) circulate waiting to serve riders and take them to their destination. Riders enter the system by requesting a ride from region i to region j or from a specific part of region i to another place in the same region. In this dynamic scenario, the main difficulty in matching as many requests as possible is the potential demand-supply spatial imbalances that might arise in cities where some areas are more populated or congested than others. In order to mitigate this issue, the solution proposed by the paper entails a *centralized mechanism* which moves empty cars to areas where more rides are requested so as to ensure the availability of service in regions where the demand is higher. Even though in most real-world ride sharing systems, such as Uber, Ride or Didi, the decision to relocate in another region is made independently and strategically by the drivers, this solution can be considered a good benchmark for decentralized systems where platforms create incentives for drivers to relocate to regions where they are most needed (through surge pricing or other strategies), namely it would be possible to analyze the efficiency of those strategies against the performance of a routing policy with which the platform decides where drivers should relocate and they have to abide by those instructions. The objective of Braverman et al. (2019) is to find an *static empty-car routing policy* that maximizes the revenue generation of the system (this is of course connected to the maximization of matched requests), to prove its asymptotic optimality against any other state dependent routing policy and to compare it with other widely known policies through a simulation with real-world data.

3.2 Analytical Tools

In this section, we introduce the mathematical concepts needed to get a thorough understanding of the model.

3.2.1 Continuous-Time Markov Chain

As it is described in Ross (1996), a continuous-time Markov chain is a stochastic process having the Markovian property that the conditional distribution of the future state, given the present state and all the past states, depends only on the present state and is independent of the past.

Consider a system where a *server station* provides a certain kind of service, and *clients* arrive at this station forming a queue. Passengers arrival rate is modelled as a Poisson process with rate λ. The service is provided with rate μ which is also exponentially distributed. *Interarrival and service time are assumed to be i.i.d.* (independent and identically distributed). A prerequisite for the system to be stable is $\lambda < \mu$, so that newcomers find the station free and are served in time; while in the case in which $\lambda \geq \mu$, the system would become unstable and queues will go to infinity. At each state in time, if k clients are in the system, two possible transition are involved:

$$k \Rightarrow k+1 \quad \text{at rate} \quad \lambda$$

when a new client arrives, or

$$k \Rightarrow k-1 \quad \text{at rate} \quad \mu$$

when a client is served. Moreover, the system addressed so far only entailed one single server station where clients show up to and get served, but also systems with multiple server stations can be anaylized; in fact, ride hailing systems deal with such kind of settings, since riders can request a ride from any part of the city.

3.2.2 Fluid Approximation

Analyzing Continuous-time Markov Chains can be extremely complex. To that aim, it is useful to resort to approximations of stochastic models through deterministic equations. One potential solution is the so-called *Fluid Approximation*, which considers the deterministic model as the limit of the stochastic process

for large populations/system size. What this concretely implies is the following: the clients arrival rate is sped up from λ to $N\lambda$; at the same time, though, the "weight" of one individual is scaled from 1 to $\frac{1}{N}$. The approximation considers the resulting scenario as $N \to \infty$. This model is fluid, no randomness is involved and it is possible to replace random variables with their expected values and the resulting system to analyze is a deterministic one composed by differential equations.

As will be shown later in the project, this technique is crucial to solve the optimization problem studied by Braverman et al. (2019).

3.3 Analytical Problem

In this section we describe in detail the specifications of the problem and analytical characteristics that help understand the results obtained by Braverman et al. (2019).

Let us remind that in this model N cars provide the service in r different regions. Passengers' arrivals at region i are modelled as a Poisson process with rate $N\lambda_i$, in order to ensure stability in the demand and supply levels in every region. The average time needed to travel from region i to region j is $1/\mu_{ij}$ and, as the Markov Chain setting suggests, travel times are assumed to be i.i.d. random variables. It is now important to understand what happens when a passenger shows up at a server station and, subsequently to the deployment of the service, what are the options for the drivers.

When a client arrives to region i, if at least one empty car is available, then the passenger is served and the full car travels from i to j with probability P_{ij}. A fact worth noticing is that trips within regions are allowed and happen with probability P_{ii}. In the case in which no empty car is available at a passenger's arrival, the client quits the system and uses another mean of transport. When a full car gets to destination j, drops the passenger, becomes empty and, at this point, can either stay at region j and wait for a new ride request with probability Q_{jj}, or redirect to region k with probability Q_{jk}. Notice that travel times for empty cars are modelled as equal to those with a passenger.

At any time t, $E_{ij}^{(N)}(t)$ is the amount of empty cars redirecting from region i to region j, while $E_{ii}^{(N)}(t)$ represents the number of empty cars waiting in region i for a passenger to request a ride. It is important to notice that, in this model, "empty routing from region i to region i" is considered the same as "waiting still

at the server station", which, in this case, is the whole region i. Furthermore, $F_{ij}^{(N)}(t)$ is the number of full cars, meaning the cars serving a customer at time t, driving from region i to region j, whereas $F_{ii}^{(N)}(t)$ involves those rides where the origin and destination regions of the service are the same.

We define

$$\begin{aligned} E^N &= \{E^N(t) \in \mathbb{Z}_+^{r\times r},\ t \geq 0\} \\ F^N &= \{F^N(t) \in \mathbb{Z}_+^{r\times r},\ t \geq 0\} \\ \bar{E}^N &= \left\{\frac{1}{N}E^N(t) \in \mathbb{R}_+^{r\times r},\ t \geq 0\right\} \\ \bar{F}^N &= \left\{\frac{1}{N}F^N(t) \in \mathbb{R}_+^{r\times r},\ t \geq 0\right\} \end{aligned}$$

where $E^N(t)$ and $F^N(t)$ are the $r\times r$ matrices whose (i,j)th elements are $E_{ij}^{(N)}(t)$ and $F_{ij}^{(N)}(t)$. Also, Braverman et al. (2019) define

$$\mathfrak{T} = \left\{(e,f) \in [0,1]^{r\times r} \times [0,1]^{r\times r} : \sum_{i=1}^{r}\sum_{j=1}^{r}(e_{ij}+f_{ij}) = 1\right\}$$

where e_{ij} and f_{ij} are placeholders for $\bar{E}_{ij}^{(N)}(t)$ and $\bar{F}_{ij}^{(N)}(t)$, respectively. The condition $(e,f) \in \mathfrak{T}$ is called "unit mass" condition, as the entries for those matrices sum up to 1.

The empty car routing matrix $Q^{(N)}(t)$, whose entries are $Q_{ij}\big(\bar{E}^N(t), \bar{F}^N(t)\big)$, is our decision variable, meaning that our aim is to build that matrix based on the result of the optimization problem. This implies that the process $\big(E^N(t), F^N(t)\big)$ is our *Continuous Time Markov Chain* and its transition rates, which are much more complex than those used to give the general idea of the mathematical concept in Section 3.2.1, are described in Table 1.

The first line of Table 1 shows the rate at which passengers arrive at region i, request a ride to region j and a driver picks her up to region j, thus the number of empty cars idling in region i decreases by 1 and the number of full cars driving from i to j increases by 1. This transition is conditional on the availability of empty cars in region i, otherwise the passenger simply abandons the system and E_{ii}^N and F_{ij}^N do not change. The second transition involves a passenger being dropped off at destination (region j), in this case, the number of full cars driving from i to j decreases by 1 and either the number of empty cars waiting at region j increases or, if the driver is relocated, the number of

Rate	Transition
$N\lambda_i P_{ij}$ if $E_{ii}^N(t) > 0$ 0 otherwise	$E_{ii}^N(t) - 1$; $F_{ij}^N(t) + 1$
$\mu_{ij} F_{ij}^N(t) Q_{jk}\big(\bar{E}^N(t), \bar{F}^N(t)\big)$	$F_{ij}^N(t) - 1$; $E_{jk}^N(t) + 1$
$\mu_{ij} E_{ij}^N(t)$ if $j \neq i$ 0 otherwise	$E_{ij}^N(t) - 1$; $E_{jj}^N(t) + 1$

Table 1: transition rates and subsequent changes in the network

empty cars driving from region j to region k increases by 1. Lastly, in the third line an empty car reaches the destination of its relocation, thus the number of empty cars driving from i to j decreases and the number of empty cars waiting in region j increases.

The availability at region i is defined as the amount of time that at least one empty car is available at region i, which is considered to be equal to the probability that a ride request from region i will be served. The scenario that Braverman et al. (2019) consider is defined *Large Market Regime*, namely they investigate the asymptotic behaviour when the number of cars N goes to ∞. Because of how the model is specified, this implies that, under that condition, the passenger arrival rate $N\lambda$ also goes to ∞. They define the long-term (asymptotic) availability at region i as $A_i^N = \mathbb{P}\big(\bar{E}_{ii}^N(\infty) > 0\big)$.

The analytical framework is now depicted, what is still missing is an explanation of the fluid model that Braverman et al. (2019) used to obtain ther main results.

3.3.1 Fluid System

In this section we describe the process that leads to the formulation of the final system of equations used by Braverman et al. (2019) to solve the Empty-cat Routing optimization problem.

In their work, the authors specify the set of equations representing the fluid

model, whose variables are the deterministic fluid analogs of our previously defined $\bar{E}^N$ and $\bar{F}^N$. The resulting new system of equations constitutes exactly the constraints of the optimization problem which will be specified later on.

The theorems proved in this part of their work are the bases, the building blocks of their results, because they allow us to replace random variables, stochastic metrics, with deterministic expected values through which a static empty-car routing policy can be defined. The next step is to make the final system of equations of the optimization problem explicit, describing the steps needed to solve it and report the theorems constituting the main results of the work by Braverman et al. (2019)

3.4 Results

In this section we describe the most critical results of Braverman et al. (2019) and their implications.

To specify the fluid-based optimization problem, it is necessary to define some new variables, which are as follows: firstly, c_{ij} is the rewards for ride completion from region i to region j; secondly, $q = \big(q_{ij}\big)$ is the $r \times r$ matrix representing the static empty-car routing policy Q; lastly, $\bar{e}$, $\bar{f}$ and $\bar{a}$ can be interpreted as substitutes for $\mathbb{E}\big(\bar{E}^N(\infty)\big)$, $\mathbb{E}\big(\bar{F}^N(\infty)\big)$ and $\bar{A}^N(\infty)$ and they are the variables we seek to find out to maximize platform's revenue. Here is the system to solve to get our policy:

$$\max_{q,\bar{e},\bar{f},\bar{a}} \sum_{i=1}^{r} \sum_{j=1}^{r} \bar{a}_i \lambda_i P_{ij} c_{ij} \tag{1}$$

subject to

$$\bar{a}\lambda_i P_{ij} = \bar{f}_{ij}\mu_{ij} \quad 1 \leq i,\ j \leq r \tag{2}$$

$$q_{ij}\sum_{k=1}^{r}\left(\mu_{ki}\bar{f}_{ki}\right) = \bar{e}_{ij}\mu_{ij} \quad 1 \leq i,\ j \leq r, \quad j \neq i \tag{3}$$

$$\lambda_i \bar{a}_i = \sum_{k=1;k\neq i}^{r}\left(\mu_{ki}\bar{e}_{ki}\right) + q_{ii}\sum_{k=1}^{r}\left(\mu_{ki}\bar{f}_{ki}\right) \quad 1 \leq i \leq r \tag{4}$$

$$\left(1\ -\ \bar{a}_i\right)\bar{e}_{ii} = 0 \quad 1 \leq i \leq r \tag{5}$$

$$\left(\bar{e}, \bar{f}\right)\ \in\ \mathfrak{T} \tag{6}$$

$$q_{ij} \geq 0 \qquad \sum_{j=1}^{r} q_{ij} = 1 \qquad 0 \leq \bar{a}_i \leq 1 \qquad 1 \leq i,\ j \leq r \tag{7}$$

To give some intuitive interpretation of the set of equations reported above, the quantity to maximize is composed by two components, namely c_{ij}, the reward for successful rides, and $\bar{a}_i\lambda_i P_{ij}$, which can be seen as the initialization rate for rides from region i to j; meaning that what we are maximizing here is a metric for *revenue generation*. The first constraint, equation (2), states that the aforementioned initialization rate from i to j must be equal to $\bar{f}_{ij}\left(1/\mu_{ij}\right)$, the fluid mass of full cars driving from i to j over the average travel time between the two regions at stake. Likewise, equation (3) implies that, $q_{ij}\sum_{k=1}^{r}\left(\mu_{ki}\bar{f}_{ki}\right)$, which can be considered the rate at which empty cars are off, must be equal to $\bar{e}_{ij}\mu_{ij}$, namely the mass of empty cars going from i to j over the related travel time. Equation (4) is called "car flow balance" by Braverman et al. (2019), as it states that the total rate of outflow from region i, namely $\lambda_i\bar{a}_i$, is required to be equal to the total inflow into the same region, which is computed as $\sum_{k=1}^{r}\left(\mu_{ki}\bar{e}_{ki}\right) + q_{ii}\sum_{k=1}^{r}\left(\mu_{ki}\bar{f}_{ki}\right)$. Equation (5) establishes an important condition to be respected in order for the system to be stable, which is as follows: undersupply at region i, namely having $1 - \bar{a}_i > 0$, only occurs when there is no empty car waiting in that server station ($\bar{e}_{ii} = 0$). On the other hand, if that proved wrong and $\bar{e}_{ii} > 0$, it would ensure that ride requests are picked up and served in that region, thus $\bar{a}_i = 1$.

3.4.1 Theorem 1

This first theorem links the random variables representing the number of full and empty cars driving across regions, and also the empty cars waiting still in one region, with the a feasible solution of the system of equations $(1)-(7)$ representing the fluid-based optimization problem. More specifically, Theorem 1 by Braverman et al. (2019) states that, given a solution of equations $(1)-(7)$ $\big(q, \bar{e}, \bar{f}, \bar{a}\big)$, as $N \to \infty$ we have that

$$F_{ij}^N(\infty) \Rightarrow \bar{f} \tag{8}$$

$$E_{ij}^N(\infty) \Rightarrow \bar{e}_{ij} \quad 1 \leq i \neq j \leq r \tag{9}$$

$$E_{ii}^N(\infty) \Rightarrow 0 \; for \; any \; i \; such \; that \; \bar{a}_i < 1 \tag{10}$$

$$\sum_i E_{ii}^N(\infty) \Rightarrow \sum_i \bar{e}_{ii} \; for \; any \; i \; such \; that \; \bar{a}_i = 1 \tag{11}$$

$$\mathbb{P}\big(\bar{E}_{ii}^N(\infty) > 0\big) \to \bar{a}_i \quad 1 \leq i \leq r \tag{12}$$

This theorem states that, by setting our decision variable to q (the routing policy stemming from the fluid-based optimization problem), random variables representing the distributions of full and empty cars idling around the city will asymptotically converge to the expected values of such quantities, which are obtained by solving the aforementioned optimization problem.

3.4.2 Theorem 2

Theorem 2 by Braverman et al. (2019) is divided in two parts (a) and (b). Part (a) states the following: Let $\big(q^*, a^*, e^*, f^*\big)$ be an optimal solution for the optimization problem defined by equations $(1)-(7)$. Then,

$$\sum_{i=1}^{r}\sum_{j=1}^{r} A_t^N(\infty)\lambda_i P_{ij} c_{ij} < \sum_{i=1}^{r}\sum_{j=1}^{r} \bar{a}_i^* \lambda_i P_{ij} c_{ij} \; with \; N > 0$$

Part (b) affirms that, letting $\big(\bar{E}^{(N)*}(t), \bar{F}^{(N)*}(t)\big)$ be the CTMC under the static routing policy q^*, if

$$P_{ij} > 0 \quad 1 \leq i, j \leq r$$

$$q_{ii}^* > 0 \quad 1 \leq i \leq r$$

then,

$$\lim_{N\to\infty} \sum_{i=1}^{r}\sum_{j=1}^{r} A_t^{(N)*}(\infty)\lambda_i P_{ij} c_{ij} = \sum_{i=1}^{r}\sum_{j=1}^{r} \bar{a}_i^* \lambda_i P_{ij} c_{ij}$$

The second theorem by Braverman et al. (2019) gives two main contributions to their work. The first part states that, given an optimal solution of the fluid-based optimization problem, the resulting utility is proved to be an "upper bound on the expected system utility of the system with N cars under any state-dependent routing policy"(Braverman et al. (2019)). Moreover, the second part states that it is actually possible to attain such result as $N \to \infty$ by enforcing the resulting optimal static empty-car routing policy q^*.

3.4.3 Solving Procedure

In order to solve the system of equations (1)-(7), it is necessary to handle the *bilinear constraint* they exhibit. There are many methods available to transform such systems into optimization problems with *linear constraints* only and in the next lines we are going to see the reasoning adopted by Braverman et al. (2019) in this respect.

Here is the transformed set of constraints:

$$\bar{a}\lambda_i P_{ij} = \bar{f}_{ij}\mu_{ij} \quad 1 \leq i,\ j \leq r \tag{13}$$

$$\mu_{ij}\bar{e}_{ij} \leq \sum_{k=1}^{r} \mu_{ki}\bar{f}_{ki} \quad 1 \leq i,\ j \leq r, \quad j \neq i \tag{14}$$

$$\sum_{k=1;k\neq i}^{r} \left(\mu_{ki}\bar{e}_{ki}\right) \leq \lambda_i \bar{a}_i \leq \sum_{k=1;k\neq i}^{r} \left(\mu_{ki}\bar{e}_{ki}\right) + \sum_{k=1}^{r} \left(\mu_{ki}\bar{f}_{ki}\right) \quad 1 \leq i \leq r \tag{15}$$

$$\lambda_i \bar{a}_i + \sum_{j=1;j\neq i}^{r} \left(\mu_{ji}\bar{e}_{ji}\right) = \sum_{k=1;k\neq i}^{r} \left(\mu_{ki}\bar{e}_{ki}\right) + \sum_{k=1}^{r} \left(\mu_{ki}\bar{f}_{ki}\right) \quad 1 \leq i \leq r \tag{16}$$

$$\left(\bar{e}, \bar{f}\right) \ \in \ \mathfrak{T} \tag{17}$$

$$0 \leq \bar{a}_i \leq 1 \qquad 1 \leq i \leq r \tag{18}$$

$$\left(1 - \bar{a}_i\right)\bar{e}_{ii} = 0 \qquad 1 \leq i \tag{19}$$

Braverman et al. (2019) prove that if the solution $\left(\bar{e}, \bar{f}, \bar{a}\right)$ and q satisfy the equations (1)-(7), then it will be a solution of $(13) - (19)$ as well. On the other hand, if this solution holds for $(13) - (19)$, then it will satisfy (1)-(7), provided that the decision matrix Q is defined as follows:

$$\begin{aligned} q_{ij} &= \frac{\mu_{ij}\bar{e}_{ij}}{\sum_{k=1}^{r}(\mu_{ki}\bar{f}_{ki})} && 1 \leq i,\ j \leq r, \quad j \neq i \\ q_{ii} &= \frac{\lambda_i \bar{a}_i - \sum_{k=1}^{r}(\mu_{ki}\bar{e}_{ki})}{\sum_{k=1}^{r}(\mu_{ki}\bar{f}_{ki})} && 1 \leq i \leq r \end{aligned} \tag{20}$$

It can be noticed that in the new system $(13) - (19)$, one bilinear constraint is still present, namely equation (19). Braverman et al. (2019) deal with the bilinearity of this equation proving that it can be easily overlooked, since any optimal solution of the system $(13) - (18)$ will satisfy this constraint.

Hence, the new target optimization problem is determined by $(13) - (18)$, which can be easily solved with the aid of a standard software. After that, all is needed is to plug the resulting variables into equation (20) and find the optimal empty-car routing policy we have been looking for.

3.5 Experimental Results

This section will be devoted to the description of the real-world data study carried out by Braverman et al. (2019) to test the optimal routing policy they obtained from their model. The dataset they relied on is from *Didi Chuxing* (2019) ("Didi"), one of the largest transportation platforms operating in Asia and Latin America.

Their tests were aimed at building on the theoretical results from Theorem 2 and verify how the static empty-car optimal policy would perform in a *finite-cars setting* against two widely used state-dependent policies. Based on revenue generation function (equation 1) maximized by the fluid-based system, they started off by defining a *Utility function* as follows:

$$\mathcal{U}(\bar{e}, \bar{f}, \bar{a}) = \mathcal{U}(\bar{a}) = \frac{\sum_{i=1}^{r} \bar{a}_i \lambda_i}{\sum_{i=1}^{r} \lambda_i} \tag{21}$$

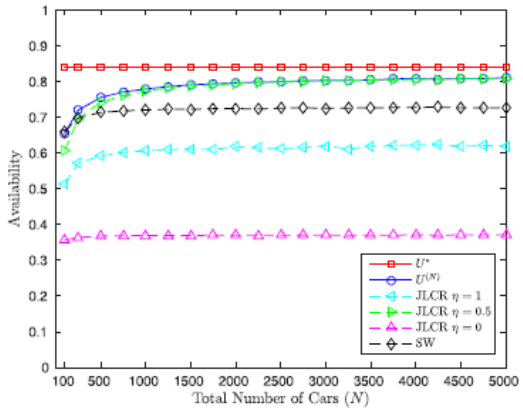
with the reward function for completing a ride equal to $c_{ij} = \frac{1}{\sum_{i=1}^{r} \lambda_i}$. This formula is to be interpreted as the ratio between the rate of fulfilled requests, given by the numerator $\sum_{i=1}^{r} \bar{a}_i \lambda_i$, and the total rate of requests, given by the denominator $\sum_{i=1}^{r} \lambda_i$. This was used as a metric of policy performance on the given dataset.

Let us understand how those two state-dependent policies work. The first one is named *Join-the-Least-Congested-Region with Threshold* η (JLCR-η) and works exactly as its name suggests, meaning a driver, after completing a ride, relocates empty in a different region if and only if the difference in congestion (mass of empty cars both waiting and driving towards region i) between the two regions exceeds η times a measure of congestion, otherwise the driver waits still in the region she currently is. The other policy used in the test is called *Shortest*

Wait and, as also in this case the name suggests, it is aimed at minimizing the time that drivers have to wait to get their next passenger. Indeed, a driver will wait in region i if the expected time to get the next ride request is the minimum across all the regions, also taking into account the time it would take to get to each destination and the number of cars that are expected to leave the region in the meanwhile.

Their main result is shown by Figure 1, which compares the performance of all the policies, plotting the Availability against the number of cars in the system. Among the lines in the plot, U^* is the optimal utility attained in the fluid limit with the estimated parameters, while, concerning $U(N)$, they run a simulation increasing the number of cars in the system keeping the lambdas constant. Let us remember that the process $(E^{(N)}, F^{(N)})$ is the CTMC, and, moreover,the availability in each region is $A_i^N = \mathbb{P}\big(\bar{E}_{ii}^N(\infty) > 0\big)$ Thus, what Braverman et al. (2019) did is the following: they simulated the Markov Chain, let it run until it achieved a stationary distribution, then they replaced the availability in the limit with the fraction of times in which $E_{ii}^N > 0$. This

Figure 1: Plot extracted by Braverman et al.(2019)



provided an estimate for $\bar{a}_i$, which was plugged in equation 21. This procedure was then repeated for many values of N to obtain the blue curve in Figure 1. The plot exhibits great performance by the optimal static policy Q^* resulting from the fluid-based optimization model, as it outperforms both JLCR-η and the Shortest Wait policy. Nonetheless, it is important to specify that the amount of parameters needed to compute Q^* is far higher than those required by JLCR-η,

as the latter only needs knowledge of λ, whereas the former make use of λ, μ and the matrix P.

4 Simulation on Chicago Transportation Network Data

4.1 Overview

In this section we describe the practical implementation of the approach by Braverman et al. (2019) on a different real-world dataset and aim to depict the differences in performance of the model. Specifically, we estimated the parameters λ,μ and P_{ij} from the data, solved the fluid-based optimization problem specified by equations (1) and (13) − (18) and computed the achieved Utility given by equation (21) so as to make a comparison of the results across datasets.

4.2 The Dataset

The dataset we employed for our analysis contains the records of trips from the "Transportation Network Providers", namely ridehailing systems, of Chicago (US) and is available on the *Chicago Data Portal* (2021). On this platform, developers can find any sort of Open Data recorded in the city and use them to carry out analyses and simulation. The available dataset contains approximately 195 millions observations, where each observation is one trip, and all the orderd recorded were fulfilled. The dataset is updated once every three months and includes records starting from November 2018. The city is partitioned in a number of distinct regions.

We started off connecting to an API to get the data from the portal and retrieving observations from April 2021 (roughly 200000 rides). This process required a large amount of time and computational power and that is why we restricted the scraping to such limited data.

```
client = Socrata("data.cityofchicago.org",
                 app_token= "my_app_token",
                 username="my_username",
                 password="my_password",
```

```
                timeout=10000)
results = client.get("wrvz-psew", limit=200000)
results_df = pd.DataFrame.from_records(results)
```

Next, we see how the dataset was manipulated in order to perform the needed estimations.

4.3 Data Cleaning & Preparation

The raw data needed to be preprocessed before we could dive into the estimation of the parameters. In this section, we describe all the steps of the preprocessing phase that eventually lead to the polished dataset used in the simulations.

The dataset originally had 21 columns, which are showed here below:

```
Input:
chicago_raw.columns

Output:
Index(['Unnamed: 0', 'trip_id', 'taxi_id',
 ↪  'trip_start_timestamp',
       'trip_end_timestamp', 'trip_seconds', 'trip_miles',
       'pickup_community_area', 'dropoff_community_area',
        ↪  'fare', 'tips',
       'tolls', 'extras', 'trip_total', 'payment_type',
        ↪  'company',
       'pickup_centroid_latitude', 'pickup_centroid_longitude',
       'pickup_centroid_location', 'dropoff_centroid_latitude',
       'dropoff_centroid_longitude',
        ↪  'dropoff_centroid_location',
       ':@computed_region_vrxf_vc4k', 'pickup_census_tract',
       'dropoff_census_tract'],
      dtype='object')
```

Not all this information was needed to carry out our analysis, thus we immediately removed the unnecessary columns.

```
#drop not needed columns
chicago_raw.drop('Unnamed: 0', axis=1, inplace=True)
chicago_raw.drop('fare', axis=1, inplace=True)
chicago_raw.drop('tips', axis=1, inplace=True)
chicago_raw.drop('tolls', axis=1, inplace=True)
chicago_raw.drop('extras', axis=1, inplace=True)
chicago_raw.drop('payment_type', axis=1, inplace=True)
chicago_raw.drop('company', axis=1, inplace=True)
chicago_raw.drop('pickup_centroid_latitude', axis=1,
↪   inplace=True)
chicago_raw.drop('pickup_centroid_longitude', axis=1,
↪   inplace=True)
chicago_raw.drop('pickup_centroid_location', axis=1,
↪   inplace=True)
chicago_raw.drop('dropoff_centroid_latitude', axis=1,
↪   inplace=True)
chicago_raw.drop('dropoff_centroid_longitude', axis=1,
↪   inplace=True)
chicago_raw.drop('dropoff_centroid_location', axis=1,
↪   inplace=True)
chicago_raw.drop(':@computed_region_vrxf_vc4k', axis=1,
↪   inplace=True)
chicago_raw.drop('pickup_census_tract', axis=1, inplace=True)
chicago_raw.drop('dropoff_census_tract', axis=1, inplace=True)
```

The resulting dataset only constained 8 columns and Table 2 reports the description of each variable that made it to the final version of the dataset.

Table 2: Variable name and description in the final version of the dataset

Variable Name	**Description**
trip id	Unique id of the ride
taxi id	Unique id of the car
trip start timestamp	Timestamp (format yy-mm-dd H-M-S) of the moment the ride started
trip end timestamp	Timestamp (format yy-mm-dd H-M-S) of the moment the ride ended
trip seconds	Total duration of the ride (in seconds)
trip miles	Total length of the ride (in miles)
pickup community area	Region of the city where the passenger was picked up (from 1 to 77)
dropoff community area	Region of the city where the passenger was dropped off (from 1 to 77)
trip total	Total fare for the trip

We proceeded by dropping the rows that contained Null Values, since the dataset only contained records for completed rides, which means that those could be classified as misrecorded observations.

```
#Check number of null values in each column
chicago_raw.isnull().sum()
#drop all observations with NaN values
chicago_raw = chicago_raw.dropna(how='any',axis=0)
zero_values = chicago_raw[chicago_raw.eq(0).any(1)]
chicago_raw = chicago_raw.drop(zero_values.index)
```

Discrepancies in the data may come in many forms. In the next lines, we describe how we handled the various types of badly recorded data.

We first checked the maximum values for "trip miles" and "trip total", so as to verify whether any unexpectedly high value occurred in those columns.

```
Input:
chicago_raw[['trip_miles','trip_total']].max()
Output:
trip_miles     994.50
trip_total    8260.56
dtype: float64
```

We then investigated on the other columns' values of those observation.

```
Input A:
chicago_raw.loc[chicago_raw['trip_miles'].idxmax()]
Input B:
chicago_raw.loc[chicago_raw['trip_total'].idxmax()]

Output A:
trip_id
↪  bdedfa0e22bb43b26988e9c77bafce825a9dc0d4
taxi_id
↪  279e6ef4129260b19e953938f5ca14fba369cbd6db1f06...
trip_start_timestamp                                2021-04-11
↪  18:30:00.000
trip_end_timestamp                                  2021-04-11
↪  18:45:00.000
trip_seconds
↪  1200
trip_miles
↪  994.5
pickup_community_area
↪  56
dropoff_community_area
↪  8
```

```
trip_total
↪  30.25
Name: 150665, dtype: object
Output B:
trip_id
↪  41a6bd4d878d8b54aaa4cd1526252e978f54051c
taxi_id
↪  10b2fc771259a5d8a802c85b6a1d0f5efe023f55bbd504...
trip_start_timestamp                                2021-04-17
↪  13:45:00.000
trip_end_timestamp                                  2021-04-17
↪  13:45:00.000
trip_seconds
↪  180
trip_miles
↪  0.6
pickup_community_area
↪  39
dropoff_community_area
↪  39
trip_total
↪  8260.56
Name: 105119, dtype: object
```

It was clear that those were discrepancies in the data, since, for instance, 994.5 miles is inconsistent with the total fare and the duration of the ride. Therefore, we deleted all the observation exhibiting such characteristics.

```
discrepancies = chicago_raw[ (chicago_raw['trip_miles'] >
↪  chicago_raw['trip_total']) & (chicago_raw['trip_miles'] >
↪  100)]
chicago_raw = chicago_raw.drop(discrepancies.index)
```

At this point, the dataset was almost suitable for the extrapolation of the param-

eters. As we aimed at following the same procedure as Braverman et al. (2019), we selected the same time window of the day and filtered the data in order to obtain only rides occurred between 5 *pm* and 6 *pm*.

```
chicago_raw['trip_start_timestamp']
 ↪   =chicago_raw['trip_start_timestamp'].str.slice(0, 18)
chicago_raw['trip_end_timestamp']
 ↪   =chicago_raw['trip_end_timestamp'].str.slice(0, 18)
chicago_raw['trip_start_timestamp'] =
 ↪   pd.to_datetime(chicago_raw['trip_start_timestamp'],
 ↪   format='%Y-%m-%dT%H:%M:%S' )
chicago_raw['trip_end_timestamp'] =
 ↪   pd.to_datetime(chicago_raw['trip_end_timestamp'],
 ↪   format='%Y-%m-%dT%H:%M:%S' )
chicago_raw.index = chicago_raw['trip_start_timestamp']
chicago_raw = chicago_raw.drop('trip_start_timestamp', axis=1)
chicago_raw=chicago_raw.between_time('17:00:00', '18:00:00')
```

In the next section, we present the procedure we followed to estimates the parameters of our *5-regions Network* which were subsequently used in the optimization problem.

4.4 Parameters Estimation

In this section, we illustrate how we extrapolated the parameters λ,μ and P_{ij} from our data following the same procedure Braverman et al. (2019) also used.

One last passage before estimating our variables was to define our network of regions, namely we selected the 5 regions among which the highest number of rides occurred. Instead of opting for 9 regions, as in Braverman et al. (2019), we selected a more restricted network and this decision was due to the limited number of observations (and thus rides occurred) at our disposal.

```
trips_per_region=
 ↪   chicago.groupby("pickup_community_area")["trip_id"].count()
trips_per_region = trips_per_region.sort_values()
```

```
most_requested_regions=[]
for i in trips_per_region.index[-5:]:
  most_requested_regions.append(i)
most_requested_regions.sort()

mrr=most_requested_regions

chicago=chicago[chicago['pickup_community_area'].isin(mrr)]
chicago=chicago[chicago['dropoff_community_area'].isin(mrr)]
```

The final version of our dataset contained 5182 rides originating and ending in the following regions:

```
Input A:
chicago.shape
Output A:
(5182, 8)

Input B:
most_requested_regions
Output B:
[6.0, 8.0, 28.0, 32.0, 76.0]
```

At this point, we were ready to estimate the parameters of the network.

Similarly to the rest of the simulation, we carried out the same procedure as Braverman et al. (2019), we computed the matrix μ, namely the service rate, using the *average travel time* among the regions in the network. In order to do that, we first built a multi-index series with lists of all the records for travel times among regions.

```
Input:
chicago_restricted =
↪  chicago[['pickup_community_area','dropoff_community_area',
'trip_seconds']]
```

```
#create a multi-index series
#with the list of all travel times among regions i and j
new=chicago_restricted.groupby(['pickup_community_area',
    'dropoff_community_area']).trip_seconds.apply(list)
#example of the output for region 8
new[8.0]
Output:
dropoff
6.0           [1380.0, 1080.0, 1080.0, 1260.0, 1270.0, 979.0...
8.0           [180.0, 326.0, 360.0, 42.0, 384.0, 180.0, 720....
28.0          [720.0, 1042.0, 720.0, 720.0, 978.0, 540.0, 48...
32.0          [480.0, 245.0, 300.0, 720.0, 531.0, 1380.0, 41...
76.0          [3780.0, 2358.0, 3980.0, 2880.0, 3600.0, 2040....
Name: trip_seconds, dtype: object
```

Only then we were able to compute the average travel time per region and insert the values into a matrix.

```
mrr = most_requested_regions
mu =[[0 for x in range(len(mrr))] for x in range(len(mrr))]
for row in range(len(mrr)):
  for col in range(len(mrr)):
    mu[row][col] = 1/(statistics.mean(new[mrr[row]][mrr[col]]))
```

The resulting matrix μ is

$$\mu = \begin{pmatrix} 0.00201 & 0.00115 & 0.00062 & 0.00079 & 0.00041 \\ 0.00097 & 0.00227 & 0.00138 & 0.00184 & 0.00034 \\ 0.00069 & 0.00142 & 0.00152 & 0.00177 & 0.00036 \\ 0.00099 & 0.00196 & 0.00257 & 0.00298 & 0.00016 \\ 0.00039 & 0.00040 & 0.00040 & 0.00037 & 0.00058 \end{pmatrix}$$

As in Braverman et al. (2019), to compute P_{ij}, namely the probability of picking up a passenger and driving her from region i to j, we consider the number of rides from region i to region j, and divide by the total number of orders originating at i.

```
P_ij=[[0 for x in range(len(mrr))] for x in range(len(mrr))]
for row in range(len(mrr)):
  from_i=0
  for i in range(len(mrr)):
    from_i += len(new[mrr[row]][mrr[i]])
  for col in range(len(mrr)):
    P_ij[row][col] = len(new[mrr[row]][mrr[col]])/from_i
```

The output is the following matrix:

$$P_{ij} = \begin{pmatrix} 0.4900 & 0.3346 & 0.0796 & 0.0876 & 0.0079 \\ 0.1021 & 0.4762 & 0.1806 & 0.2248 & 0.0161 \\ 0.0563 & 0.3848 & 0.3140 & 0.2254 & 0.0193 \\ 0.0566 & 0.4457 & 0.2524 & 0.2330 & 0.0121 \\ 0.1618 & 0.4422 & 0.1051 & 0.1567 & 0.1340 \end{pmatrix}$$

To determine the arrival rate at region i, we computed the *average number of orders originating at region i per time slot*. Now, differently from Braverman et al. (2019), our dataset did not divide time in slots, thus we further processed the data to achieve that outcome manually.

```
timeslot= [0 for x in range(7)]
#creating timeslots through dictionary
dct= {}
dct['17:00']= 1
dct['17:15']= 2
dct['17:30']= 4
```

```
dct['17:45']= 5
dct['18:00']= 7
lambdas=[]
for i in range(len(chicago.index)):
    a = str(chicago.index[i])[-8:-3]
    timeslot[dct[a]-1]+=1

for j in mrr:
  region_timeslot = [0 for x in range(7)]
  for i in range(len(chicago.index)):
    a = str(chicago.index[i])[-8:-3]
    row = chicago.values[i]
    if row[5]==j:
      region_timeslot[dct[a]-1]+=1
  sum=0
  for i in range(len(timeslot)):
    if timeslot[i] != 0: sum+=region_timeslot[i]/timeslot[i]
  lambdas.append(sum)
```

The output is our estimate for $N\lambda$. Given that our dataset provided the unique IDs for the taxis that completed each ride ("taxi id" column), we estimated the number of cars N in the system simply by counting the number of distinct IDs in our dataset.

```
Input:
N = len(set(chicago['taxi_id']))
N
Output:
673
```

Thus, the vector λ resulting from our estimation is

$$\lambda_i = \begin{pmatrix} 0.00036 & 0.00301 & 0.00089 & 0.00177 & 0.00139 \end{pmatrix}$$

The whole set of parameters is now ready, it is time to see how we set up and

solved the optimization problem.

4.5 Optimization Problem

In this section we describe the main tools used to solve the optimization problem and the main results. Let us remind that the system we solved is specified by

$$\max_{q,\bar{e},\bar{f},\bar{a}} \sum_{i=1}^{r} \sum_{j=1}^{r} \bar{a}_i \lambda_i P_{ij} c_{ij}$$

subject to Equations (13)-(18)

In order to find the solution to the linear program we used the "PuLP library, an open source package that allows mathematical programs to be described in the Python computer programming language."(Mitchell et al. (2011)). The natural language supported by this library makes the code easy to write and understand.

```
#Creating problem object
prob = LpProblem("FLUID_LIMIT", LpMaximize)
#Defining optimizers
e= LpVariable.dicts("e", [(i, j) for i in range(5) for j in
↪  range(5)], 0, 1)
f= LpVariable.dicts("f", [(i, j) for i in range(5) for j in
↪  range(5)], 0, 1)
a= LpVariable.dicts("a", [i for i in range(5)], 0, 1)
#Objective Function
obj=0
for j in range(5):
  obj += lpSum(a[i]*lambda_i[i]*P_ij[i][j]*payoff for i in
  ↪  range(5))
prob+=obj
#Constraints
#Equation (13)
for i in range(5):
  for j in range(5):
    prob+=lambda_i[i]*a[i]*P_ij[i][j]==mu[i][j]*f[(i,j)]
#Equation (14)
```

```
for i in range(5):
  for j in range(5):
    if j==i:continue
    prob+= mu[i][j]*e[(i,j)] <= lpSum(mu[k][i]*f[(k,i)] for k in
    ↪  range(5))
#Equation (15)
for i in range(5):
  right_sum_ki=lpSum(mu[k][i]*f[(k,i)] for k in range(5))
  for k in range(5):
    if k==i:continue
    mid_sum_ki=lpSum(mu[k][i]*e[(k,i)])
    left_sum_ki=lpSum(mu[k][i]*e[(k,i)])
  prob+= left_sum_ki <= lambda_i[i]*a[i]
  prob+= lambda_i[i]*a[i] <= mid_sum_ki + right_sum_ki
#Equation (16)
for i in range(5):
  for j in range(5):
    if i==j: continue
    left_sum_ij=lpSum(mu[i][j]*e[(i,j)])
  for k in range(5):
    if k==i:continue
    mid_sum_ki=lpSum(mu[k][i]*e[(k,i)])
  prob+= lambda_i[i]*a[i] + left_sum_ij == mid_sum_ki +
  ↪  lpSum(mu[k][i]*f[(k,i)] for k in range(5))
#Equation (17)
for i in range(5):
  prob+=lpSum(e[(i,j)] for j in range(5))+lpSum(f[(i,j)] for j
  ↪  in range(5))==1
#Solving System
prob.solve()
```

We proceeded by computing the achieved utility U, given by Equation (21).

```
Input:
U=0
for i in range(5):
    l_i=lambda_i[i]
    a_i=a[i].varValue
    U+=a_i*l_i*payoff
U
Output:
0.4995716209914585
```

4.6 Simulation

In this section we illustrate the methodology followed for our simulation, provide explanation for the differences with the one by Braverman et al. (2019) and analyze the attained results.

As we previously specified in Section 3.5, Braverman et al. (2019) simulated a Markov Chain for multiple values of N in order to plot $U^{(N)}$. This was not possible for us as the computational power needed far exceeded ours, thus we decided to opt for another tactic. Instead of keeping the lambdas constant, we iteratively computed the matrix λ for different values of N (let us remember that, as in Braverman et al. (2019), what we directly estimated from the data is $N\lambda$, thus each entry of the matrix had to be divided by N), consequently changing also the payoff $c_{ij} = 1/\sum_{i=1}^{r} \lambda_i$. After that, we solved the optimization problem many times utilizing different matrices λ and payoffs c_{ij}.

```
a_solution=[]
payoff_solution=[]
lambda_solution=[]
for n in range(1,5000):
  N=n
  lambda_i =[lambdas[i]/N for i in range(len(lambdas))]
  lambda_i_arr = np.array(lambda_i)
  payoff=1/(np.sum(lambda_i_arr))
  prob = LpProblem("my_data", LpMaximize)
```

```
e= LpVariable.dicts("e", [(i, j) for i in range(5) for j in
↪  range(5)], 0, 1)
f= LpVariable.dicts("f", [(i, j) for i in range(5) for j in
↪  range(5)], 0, 1)
a= LpVariable.dicts("a", [i for i in range(5)], 0, 1)
#objective function
obj=0
for j in range(5):
  obj += lpSum(a[i]*lambda_i[i]*P_ij[i][j]*payoff for i in
  ↪  range(5))
prob+=obj
#constraints
for i in range(5):
  for j in range(5):
    prob+=lambda_i[i]*a[i]*P_ij[i][j]==mu[i][j]*f[(i,j)]

for i in range(5):
  for j in range(5):
    if j==i:continue
    prob+= mu[i][j]*e[(i,j)] <= lpSum(mu[k][i]*f[(k,i)] for k
    ↪  in range(5))

for i in range(5):
  right_sum_ki=lpSum(mu[k][i]*f[(k,i)] for k in range(5))
  for k in range(5):
    if k==i:continue
    mid_sum_ki=lpSum(mu[k][i]*e[(k,i)])
    left_sum_ki=lpSum(mu[k][i]*e[(k,i)])
  prob+= left_sum_ki <= lambda_i[i]*a[i]
  prob+= lambda_i[i]*a[i] <= mid_sum_ki + right_sum_ki

for i in range(5):
  for j in range(5):
    if i==j: continue
```

```
      left_sum_ij=lpSum(mu[i][j]*e[(i,j)])
    for k in range(5):
      if k==i:continue
      mid_sum_ki=lpSum(mu[k][i]*e[(k,i)])
    prob+= lambda_i[i]*a[i] + left_sum_ij == mid_sum_ki +
    ↪  lpSum(mu[k][i]*f[(k,i)] for k in range(5))

  for i in range(5):
    prob+=lpSum(e[(i,j)] for j in range(5))+lpSum(f[(i,j)] for j
    ↪  in range(5))==1
  #solving problem
  prob.solve()
  #creating lists with results
  temp=[]
  for key, value in a.items():
    temp.append(value.varValue)
  a_solution.append(temp)
  lambda_solution.append(lambda_i)
  payoff_solution.append(payoff)
```

Moreover, we stored multiple values of the resulting utility as N changed.

```
U=[]
for N in range(1,4999):
  sum=0
  payoff_solution_i=payoff_solution[N]
  for i in range(5):
    l_solution_i=lambda_solution[N][i]
    a_solution_i=a_solution[N][i]
    sum+=(l_solution_i*a_solution_i)
  U.append(sum*payoff_solution_i)
```

Lastly, we plotted it against the number of cars in the system.

```
figure(figsize=(10, 8), dpi=80)
plt.plot(x_points, y_points)
plt.xlabel('Number of cars (N)')
plt.ylabel('Avalability (U)')
plt.xticks(np.arange(0, 5000, step=300))
plt.yticks(np.arange(0, 1.05, step=0.05))
ax = plt.axes()
ax.set(xlim=(0, 5000), ylim=(0, 1))
plt.axes(ax)
plt.show()
```

Figure 2 shows the resulting plot. Firstly, it is important to notice that the graph plotted does not correspond specifically to $U^{(N)}$ which Braverman et al. (2019) obtained. The main differences are that, not only we did not keep the

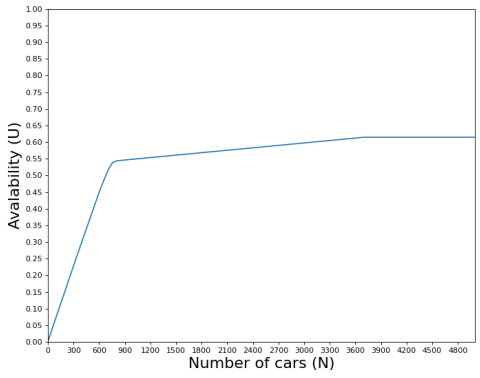


Figure 2: Availability against the number of cars in the system

parameters constant, but also did we use the availability a resulting from the optimization problem in the fluid limit, instead of simulating an estimate based on the stationary distribution of a Markov Chain. Indeed, what we plotted is the *Optimal Utility in the limit*, against the number of cars in the system, thus the corresponding value for $U^{(N)}$ would have most likely been lower. Also, this explains the steep increase in availability on the left-hand side of the graph. Secondly, the attained result is suboptimal with respect to the one on the analysis by Braverman et al. (2019), as our availability peaks at around 0.61, while theirs

is around 0.85. In this case, it can be quite confidently stated that this disparity is due to the structural and qualitative differences in the datasets used, as the methodology followed to manipulate the data and extrapolate the parameters is the same. Moreover, the accuracy of the code for the optimization problem and the absence of bugs in it is confirmed by the fact that, testing it with the parameters provided by Braverman et al. (2019) in their "Supplementary Material", we obtained the optimal utility equal to the one plotted in Figure 1.

Bibliography

Afeche, Philipp, Liu Zhe, and Costis Maglaras (2018). "Ride-Hailing Networks with Strategic Drivers: The Impact of Platform Control Capabilities on Performance". In: DOI: http://dx.doi.org/10.2139/ssrn.3120544.

Banerjee, Siddhartha, Daniel Freund, and Thodoris Lykouris (2017). *Pricing and Optimization in Shared Vehicle Systems: An Approximation Framework*. arXiv: 1608.06819 [cs.GT].

Banerjee, Siddhartha, Yash Kanoria, and Pengyu Qian (2020). *Dynamic Assignment Control of a Closed Queueing Network under Complete Resource Pooling*. arXiv: 1803.04959 [math.PR].

Besbes, Omar, Francisco Castro, and Ilan Lobel (2021). "Surge Pricing and Its Spatial Supply Response". In: *Management Science* 67(3), pp. 1350–1367. DOI: 10.1287/mnsc.2020.3622.

Bimpikis, Kostas, Ozan Candogan, and Daniela Saban (2019). "Spatial Pricing in Ride-Sharing Networks". In: *Operations Research* 67(3), pp. 744–769. DOI: 10.1287/opre.2018.1800.

Braverman, Anton, J. G. Dai, Xin Liu, and Lei Ying (2019). "Empty-Car Routing in Ridesharing Systems". In: *Operations Research* 67(5), pp. 1437–1452. DOI: 10.1287/opre.2018.1822.

Chicago Data Portal (2021). URL: https://data.cityofchicago.org/Transportation/Transportation-Network-Providers-Trips/m6dm-c72p (visited on 05/24/2021).

Curry, Michael J., John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, Yuhao Wan, and Pan Xu (2019). *Mix and Match: Markov Chains and Mixing Times for Matching in Rideshare*. arXiv: 1912.00225 [cs.DS].

Didi Chuxing (2019). URL: https://www.didiglobal.com/about-didi/about-us (visited on 05/19/2021).

Garg, Nikhil and Hamid Nazerzadeh (2021). *Driver Surge Pricing*. arXiv: 1905.07544 [cs.GT].

Housni, Omar El, Vineet Goyal, Oussama Hanguir, and Clifford Stein (2021). *Matching Drivers to Riders: A Two-stage Robust Approach*. arXiv: 2011.03624 [math.OC].

Mitchell, Stuart, Michael OSullivan, and Iain Dunning (2011). "PuLP: a linear programming toolkit for python". In: *The University of Auckland, Auckland, New Zealand*, p. 65.

Ong, Hao Yi, Daniel Freund, and Davide Crapis (2021). *Driver Positioning and Incentive Budgeting with an Escrow Mechanism for Ridesharing Platforms*. arXiv: 2104.14740 [cs.GT].

Özkan, Erhun (2020). "Joint pricing and matching in ride-sharing systems". In: *European Journal of Operational Research* 287(3), pp. 1149–1160. DOI: https://doi.org/10.1016/j.ejor.2020.05.028.

Özkan, Erhun and Amy R. Ward (2020). "Dynamic Matching for Real-Time Ride Sharing". In: *Stochastic Systems* 10(1), pp. 29–70. DOI: 10.1287/stsy.2019.0037.

Ross, S.M. (1996). *Stochastic processes*. Wiley. ISBN: 9780471120629.

statista.com (2021). URL: https://www.statista.com/statistics/1155981/ride-sharing-market-size-worldwide/ (visited on 06/08/2021).