On the Real-Time Vehicle Placement Problem

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Motivated by ride-sharing platforms' efforts to reduce their riders' wait times for a vehicle, this paper introduces a novel problem of placing vehicles to fulfill real-time pickup requests in a spatially and temporally changing environment. The real-time nature of this problem makes it fundamentally different from other placement and scheduling problems, as it requires not only real-time placement decisions but also handling real-time request dynamics, which are influenced by human mobility patterns. We use a dataset of ten million ride requests from four major U.S. cities to show that the requests exhibit significant self-similarity. We then propose distributed online learning algorithms for the real-time vehicle placement problem and bound their expected performance under this observed self-similarity.

1 Introduction

In the past five years, ride-sharing platforms like Uber and Lyft have become a significant means of transportation, accounting for nearly two billion rides in 2016. Given this popularity, ride-sharing companies have turned their attention towards the problem of optimizing rider experience. In particular, they have begun to examine ways to minimize rider wait times. Future vehicular technologies like autonomous cars can also benefit from algorithms to reduce wait times.

Reducing wait times to below two minutes is a challenging problem that requires real-time vehicle placement. The real-time nature of this problem then introduces two challenges: first, drivers should proactively predict where future pickup requests will be located and go to these locations in anticipation of future pickup requests, eliminating passenger waiting due to vehicle travel time. Yet to accurately predict future requests in the next few minutes, drivers must account for not just overall patterns in ride requests, but also real-time temporal and geo-spatial request fluctuations, which are influenced by human mobility dynamics. Second, these vehicle (driver) placement decisions must be made quickly, which precludes any significant coordination between different vehicles.

Existing work in this area has tended to focus on a coarser version of the problem that does not address one or both of these two challenges, e.g., [12] examines the placement of taxis based on predicted demand at a much coarser geo-spatial scale. The problem of load balancing by distributing vehicles around a geographical space is studied by [11]; the authors state that making ride request predictions to balance load at a small time granularity (less than thirty minutes) makes the problem hard. There has also been work using offline learning methods to optimally match riders and drivers under the assumption that all travel plans are known in advance [9], and [15] provides a recursion tree approach to maximize driver profit by recommending routes that are likely to have multiple pickup points. Probabilistic reasoning was applied by [17] to provide routing based on taxi drivers' collective intelligence. However, these latter approaches do not deliver real-time placements.

In this paper, we propose a distributed online learning approach to the problem of fine-grained (at a timescale of less than a few minutes and spatial scale of a few hundred meters) vehicle placement. An online learning algorithm gives us the flexibility to consider real-time ride request patterns in making our placement decisions. Historical information can help inform these decisions, but the utility of such historical information may vary a lot geo-spatially. We are not aware of any attempts to use online learning algorithms for real-time vehicle placement. Crucially, we discover a pattern which is exploited to analyze the performance of our proposed online real-time vehicle placement algorithms.

2 Problem Definition

We consider a geographical surface that is divided into equally sized cells of side length ϵ each, as shown in Figure 1a, to get an aggregate of $a \times b$ cells. We suppose a consecutive set of time snapshots, taken at intervals τ_ϵ, where each time snapshot t specifies a set of drop-offs at different cells, represented by the matrix $\boldsymbol{D}_t \in \mathbb{Z}^{a\times b}$, and pickups $\boldsymbol{P}_t \in \mathbb{Z}^{a\times b}$ of ride requests. Each (i, j) entry in $\boldsymbol{D}_t$ or $\boldsymbol{P}_t$ is the sum of all drop-offs or pickups that occurred at t, in cell (i, j). In this work, we have kept the length of each time snapshot to be a few minutes, reflecting our goal of keeping rider wait times under two minutes. The *vehicle placement* problem is to decide how to move vehicles from drop-off cells at time snapshot t to neighboring cells such that pickups happen at those cells at $t + 1$. Formally, our goal is to place vehicles in the "best" neighboring cells, i.e., so as to maximize a reward at time snapshot t that is defined as:

$$R_t(\boldsymbol{P}_t, \boldsymbol{\Gamma}_t) = \frac{1}{n_t}\sum_{i,j} min(\boldsymbol{P}_t[i, j], \boldsymbol{\Gamma}_t[i, j]) \tag{1}$$

where: $\boldsymbol{\Gamma}_t \in \mathbb{Z}^{a\times b}$; each entry represents the number of vehicles placed at each cell, or zero otherwise. $n_t = \sum_i^a \sum_j^b \boldsymbol{D}_{t-1}[i, j] = \sum_i^a \sum_j^b \boldsymbol{\Gamma}_t[i, j]$; the number of drop-offs at $t - 1$ or equivalently, the number of placements at t. For any t, $\boldsymbol{P}_t$ is known and the goal is to find $\boldsymbol{\Gamma}_t$ using $\boldsymbol{D}_{t-1}$ such that R_t is maximized. The reward R_t thus represents the average number of successful pick-ups per cell.

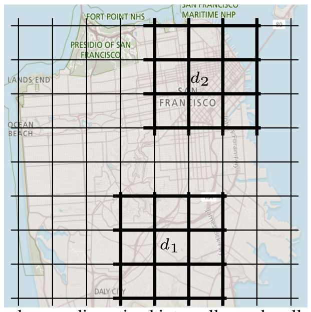
The algorithms decide the placement of each vehicle without knowing where future pick-ups will occur. Moreover, for every drop-off by a vehicle at time $t-1$ at cell (i, j), its corresponding placement at t may be either (i, j) itself or one of the neighbouring cells. This constraint is shown by the bolded cell outlines in Figure 1a. The number of neighbouring cells at which the vehicle can be placed is determined by the vehicle's ability to travel to the neighbouring cell within the time snapshot; it is thus determined by the length of the snapshot (τ_ϵ) and ϵ. This constraint also ensures that vehicles incur negligible cost (e.g., gasoline used) in moving to the exact pick-up location, as they only move to neighbouring cells. We denote the radius of the permitted neighbourhood by ϵ'. The goal of our online algorithm is then to choose the vehicle placements within any time interval $[t - 1, t)$ so as to maximize the reward (1), subject to the constraint that vehicles cannot move too far from the cell where they dropped off passengers at time $t - 1$.

The vehicle placement problem generalizes other known problems in computer science. For instance, in the k-server problem one must place k servers within a metric space so as to fulfill requests that can come in at any point in the space [10]. Viewing the servers as vehicles, our problem is a dynamic version of the k-server problem: we must adjust the vehicle placements in real time as new requests arrive, subject to constraints on the travel time of the vehicles.

3 Fractals and Ride Requests

In this section, we provide some background [3] on fractals, or self-similarity, and show that our ride request data exhibits self-similar patterns. It has been observed that real datasets of point-sets representing Montgomery county, Long Beach county, and CA road intersections exhibit properties of self-similarity [14]. In our problem we are also dealing with a natural phenomena involving human behaviors, i.e. human mobility in urban environments. Characterizing human mobility patterns using fractals provides a succinct description which could be leveraged in a variety of applications and their analyses, including our vehicle placement problem.

Fractal dimension provides a quantitative measure of self-similarity for a point-set. In the vehicle placement problem, the reward achieved by any algorithm depends on the underlying spatial patterns of pick-ups and drop-offs, which, as we observe, exhibits self-similarity. We shall focus on one specific measure of fractal dimension which relates to the estimation of spatial queries like how many points are present in a cell. Points in our context represent pickup or drop-off locations. Ride requests exhibit a biased distribution, i.e., they are more probable in high-density areas.



(a) Geographical space discretized into cells; each cell of side ϵ. For each drop-off d_i, there are 9 possible cell placements highlighted by a thick border.

City	D_2 min.	D_2 max.	D_2 mean	fractal range (m.)
Chicago	1.003	1.459	1.206	(600, 3000)
Los Angeles	0.828	1.482	1.074	(1500, 4000)
New York	1.250	1.668	1.457	(450, 2500)
San Francisco	1.049	1.686	1.343	(450, 2500)

(b) Summary of measured correlation fractal dimensions for four cities; computed over a week for every 3-minute time snapshot (3360 snapshots for a week). We exclude seven late-night hours of each day when frequency of ride requests is very low.

Figure 1: Illustration of vehicle pick-ups and drop-offs and an observed D_2 values.

Consider a high dimensional space divided into (hyper-)cubic grid cells of side ϵ. Let p_i denote the occupancy of the $i-$th cell which is the number of points which fall into the $i-$th cell of the grid.

Definition 1. *Correlation fractal dimension:* For a point-set that has self-similarity in the fractal range (ϵ_1, ϵ_2) the correlation fractal dimension D_2 is defined as:

$$D_2 \equiv \frac{\partial \log \sum_i p_i^2}{\partial \log \epsilon} = \text{constant} \qquad \epsilon \in (\epsilon_1, \epsilon_2) \tag{2}$$

For a point-set to be self-similar, in the range (ϵ_1, ϵ_2), we observe the plot of $\log \sum_i p_i^2$ versus $\log \epsilon$ must be a straight line, implying self-similarity in that range, i.e. the fractal range [3]. The slope of the line is equal to the correlation fractal dimension D_2.

Self Similarity in Ride Requests: Our dataset is a collection of real ride requests in a city over one week [8]. Each data item represents a ride request, i.e., a 4-tuple of: 1) time of request; 2) pickup location (latitude & longitude); 3) drop-off location (latitude & longitude); 4) time of drop-off.

To compute the correlation fractal dimension (D_2), our point-set comprises of pick-up locations of ride requests. We discretize time into short snapshots of 3-minute durations and divide the geographical space into square cells each with side $\epsilon = 100$ meters. Figure 1b presents the correlation fractal dimensions, D_2, from the ride request patterns of the four cities. For each city, there is a consistent weekly pattern of ride requests and a corresponding variation of its D_2 values. For each city, the values of D_2 range between D_2min and D_2max. The mean values for the four cities range from 1.0 to 1.5, indicating the degree of self-similarity of ride request patterns varies from city to city. The fractal range also varies across the cities; however, other than Los Angeles, the other cities all exhibit very similar fractal ranges, from 500 meters to 2500 meters.

The fractal dimension and the fractal range succinctly capture the characteristics of ride request patterns in a city. We believe these fractal dimensions and associated fractal ranges can be instrumental in revealing effective solutions to the real-time vehicle placement problem and facilitating the assessment of the effectiveness of specific algorithms for this problem.

4 Potential Algorithms

In this section, we examine in detail three potential algorithms for the real-time vehicle placement problem. We make use of the fractal analysis in the previous section to analyze their effectiveness. For each drop-off, the possible set of neighboring cells to which the vehicle can potentially be moved are restricted; see Figure 1a. If the drop-off location is in the (i, j) cell, then the set of neighboring cells within a square-shaped neighborhood centered at (i, j) and containing c^2 cells is denoted as $\eta(\square, (i, j), c)$. Figure 1a illustrates neighborhoods with c^2 = 3x3 cells.

Baseline Algorithm: Our simplest baseline (Algorithm 1) assumes a uniform weight across all neighbouring cells without considering any past history on the number of requests in those cells.

Poisson Process Based Algorithm: To capture the temporal pattern of ride requests, we consider a placement approach that models ride requests for every grid as a Poisson Process, as used in [16]

Algorithm 1 URand-NH (Uniform Random with No History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}_i, \boldsymbol{P}_i\ \forall i \in \{1, \ldots, k\}$.
 1: for each time snapshot $t = 1, 2, \ldots, k$ do
 2:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 3:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 4:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 5:         Choose a cell $\in N$ uniformly at random with coordinates $(i', j')$.
 6:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
 7:     end for
 8:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
 9: end for
```

(Algorithm 2). Numerous works [5, 13] have recommended against modeling human behavior as a Poisson process, due to the assumption made in the Poisson Process that consecutive events happen at regular time intervals. Contrarily, human activity tends to be bursty for short time intervals accompanied by long intervals of inactivity [2, 1]. For this reason, our maximum likelihood estimator (MLE) estimator is based on limited, and most recent history of inter arrival times of ride requests. Formally, let $N(t), t \geq 0$ be the number of ride requests (events) that occurred by time t.

Lemma 1. The probability of at least one event occurring in the time snapshot t is given by:

$$\Pr\{N(t) > 0|\lambda\} = 1 - e^{-\lambda t} \tag{3}$$

Proof. See [7]. □

Algorithm 2 PP-LH (Poisson Process with Limited History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$; $\boldsymbol{D}_i, \boldsymbol{P}_i \forall i \in \{1, \ldots, k\}$; history length $m$; min. samples $u$.
 1: for each time snapshot $t = (m+1), (m+2), \ldots, k$ do
 2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
 3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 4:     $\boldsymbol{M} = \sum_{i=t-m}^{m} \boldsymbol{D}_i + \boldsymbol{P}_i$
 5:     Estimate $\boldsymbol{\lambda}$ for each cell entry having a value $> u$ in $\boldsymbol{M}$ by computing MLE using inter-arrival times.
 6:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 7:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 8:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \Pr\{N(1) > 0|\boldsymbol{\lambda}[i, j]\}$ (see Equation 3).
 9:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
10:         $\boldsymbol{M}[i', j'] = 0$          ▷ Not discarding $(i', j')$, degrades performance since it might be considered again.
11:     end for
12:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
13: end for
```

For any instance when none of the neighbourhood cells have an estimator $\in \boldsymbol{\lambda}$, the algorithm falls back to uniform random strategy (like for URand-NH) from within the set of neighbors.

Follow The Leader (FTL) Algorithm: Follow The Leader is a sequential prediction strategy, which always puts all the weight on the cell with the highest reward so far. It has been applied to multiple online problems for time series predictions [6]. If there is a tie among the leaders, i.e., there are multiple leaders in the neighbourhood with equal reward values, a leader is chosen uniformly at random. If all cells are zero, FTL-CH also falls back to a uniform random strategy.

Algorithm 3 FTL-CH (Follow The Leader with Complete History)

```
Require: Search radius $\epsilon'$; cell length $\epsilon$ $\boldsymbol{D}$, $\boldsymbol{P}$; initial history length $m$.
 1: for each time snapshot $t = (m+1), (m+2), \ldots, k$ do
 2:     $\boldsymbol{M} = 0 \in \mathbb{R}^{a \times b}$
 3:     $\boldsymbol{\Gamma}_{t+1} = 0, \in \mathbb{R}^{a \times b}$
 4:     $\boldsymbol{M} = \sum_{i=1}^{t-1} \boldsymbol{D}_i + \boldsymbol{P}_i$
 5:     for each non-zero entry $(i, j) \in \boldsymbol{D}_t$ do
 6:         Pick a set of cells $N = \eta(\square, (i, j), \frac{2\epsilon'}{\epsilon})$.
 7:         Choose a cell $(i', j') = \text{argmax}_{(i,j) \in N} \boldsymbol{M}([i, j]$.
 8:         $\boldsymbol{\Gamma}_{t+1}[i', j'] = \boldsymbol{\Gamma}_{t+1}[i', j'] + 1$
 9:         $\boldsymbol{M}[i', j'] = \boldsymbol{M}[i', j'] - 1$          ▷ Update to history improves performance.
10:     end for
11:     Obtain rewards for $R_{t+1}(\boldsymbol{P}_{t+1}, \boldsymbol{\Gamma}_{t+1})$
12: end for
```

5 Analysis & Results

In this section we analyze the performance of our proposed algorithms. We use the fact that our ride request data exhibits self-similarity to provide an average bound on the algorithms' reward functions.

Reward Analysis: Given the observation of self-similarity, we are primarily interested in approximating queries such as: what is the average number of neighboring cells which are intersected by some neighborhood shape (like square or circle) of a certain length. Let $\overline{nb}(\epsilon')$ denote the average number of points within an enclosed square $\Box$ of radius ϵ' where $\epsilon' \in (\epsilon_1, \epsilon_2)$ for which self-similarity is observed. Note $\epsilon' > \epsilon$. An important consequence of this is:

Lemma 2. Given a set of points $\mathcal{P}$ with finite cardinality and its Correlation Dimension D_2, the average number of points within the shape $\Box$ with radius of ϵ' follows the power law: $\overline{nb}(\epsilon') \propto {\epsilon'}^{D_2}$.

Proof. See [3]. $\Box$

We now evaluate the performance of our algorithms under this self-similarity assumption. We assume throughout that our performance is determined by the placement strategy, i.e., the number of drop-offs is smaller than the number of pick-up requests in a given neighbourhood. Otherwise, if the number of drop-offs is too large, any strategy is likely to perform well. We show in our numerical analysis (Section 5.2) that this assumption holds in our dataset.

Theorem 1. Suppose that the total number of drop-offs in a given cell of radius ϵ' is no more than the expected number of pick-up requests in a cell. Then the expected performance of FTL-CH is strictly better than URand-NH for any pickup matrix $\boldsymbol{P}_t$ with $1 < D_2 < 2$: $\mathbb{E}_{\text{FTL-CH}}[R_t] > \mathbb{E}_{\text{URand-NH}}[R_t]$, and the expected number of fulfilled pick-ups per drop-off with FTL-CH exceeds that for URand-NH.

Proof sketch. In a given grid with radius ϵ', the expected reward for the Follow The Leader with Complete History (FTL-CH) algorithm at time t is given by the minimum of the number of drop-offs in that grid at time t, which we denote by D, and the number of pickups at time $t+1$ within the cell (i, j) chosen by the FTL-CH algorithm: all drop-offs at time t are placed in cell (i, j) at time $t+1$. The expected number of pickups in this cell is then proportional to ϵ^{D_2} from Lemma 2, and the expected reward is $\min\left\{D, C\epsilon^{D_2}\right\}$ for some constant $C > 0$.

If $D \leq C\epsilon^{D_2}$, then the expected reward with FTL-CH is D, which is also the maximum possible reward. Thus, the uniform random strategy cannot perform any better.

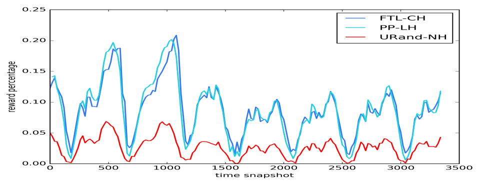
To show the second part of the theorem, we note that for the uniform random strategy, the expected fraction of total pick-ups in the grid that can be fulfilled by one drop-off is $\epsilon^2/{\epsilon'}^2$, i.e., the reciprocal of the number of cells in a grid. Then the expected number of pickups fulfilled is the expected total number of pickups multiplied by this fraction, i.e., $C{\epsilon'}^{D_2-2}\epsilon^2 = C\epsilon^{D_2}\left(\epsilon/\epsilon'\right)^{2-D_2} \leq C\epsilon^{D_2}$, the expected number of pickups under FTL-CH, since $1 < D_2 < 2$ and $\epsilon \leq \epsilon'$. $\Box$

Theorem 2. Suppose that the expected number of future pickup requests in each cell at time t is given by the number of requests experienced in the past interval of $[t-1, t)$, and the interarrival times of a sequence of pickups follow an Exponential distribution. Then the expected performance of FTL-CH is equivalent to that of PP-LH: $\mathbb{E}_{\text{FTL-CH}}[R_t] \equiv \mathbb{E}_{\text{PP-LH}}[R_t]$.

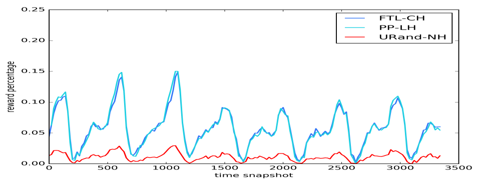
Proof sketch. Using Lemma 1 with $t = 1$, we claim that the placement cell chosen by PP-LH will on average have experienced higher pick-ups than any of its neighbouring cells for a fixed time interval. This is true because with the interval $[t-1, t)$, the maximum likelihood estimator for the Exponential distribution is the inverse of the mean inter-arrival times. The resulting λ estimate will be large for cells which have high rate of events, or equivalently cells which observed more events in the given time interval. From (3), we then see that PP-LH will choose the cell with the largest λ estimate, i.e., the cell that experienced the most number of events. Thus, the average reward obtained by PP-LH will be equivalent to that of FTL-CH [1]. $\Box$

Experimental Results: We apply the three algorithms to the ride request data sets from four cities and find the achieved rewards in each, indicating the effectiveness of the algorithms. Figure 2 plots the reward functions for the three algorithms for each 3-minute (τ_ϵ) time interval over a week. $\epsilon' = 500$ meters which effectively means there are 10^2 possible cells for any vehicle to be placed in. In Algorithms 2 and 3, $m = 20$ and 3 respectively.

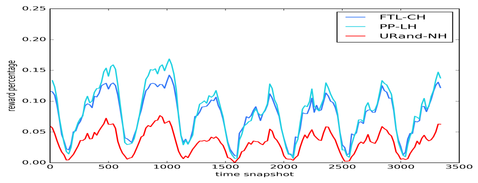
[1]For a more detailed proof refer to Lemma 1 in https://classes.soe.ucsc.edu/cmps290c/Spring09/lect/10/wmkalai-rewrite.pdf.



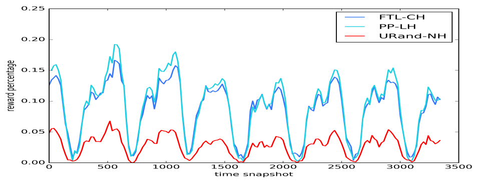
(a) Chicago



(b) Los Angeles

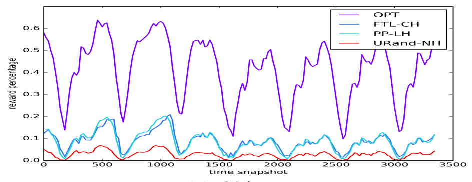


(c) New York

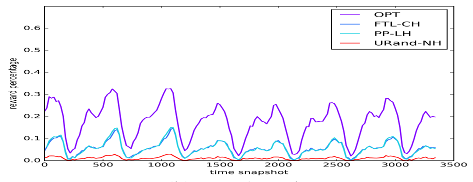


(d) San Francisco

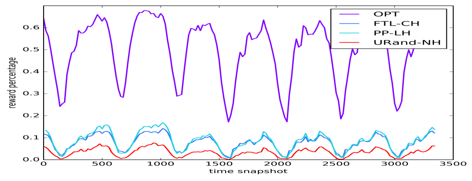
Figure 2: The PP-LH algorithm out-performs FTL-CH slightly and URand-NH significantly across all four cities in terms of the reward (Eq. 1) for a week with three minute time snapshots.



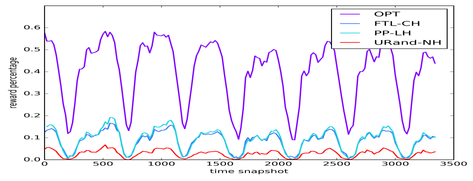
(a) Chicago



(b) Los Angeles



(c) New York



(d) San Francisco

Figure 3: Comparison of reward percentage plots for 3 algorithms along with optimal (OPT) reward.

The weekly 7-day pattern of these functions can be clearly seen. All three algorithms tend to be more effective when there are large numbers of ride requests. Supporting the claims in Theorems 1 and 2, PP-LH and FTL-CH perform similarly for all four cities. On average, the reward percentage for PP-LH is 0.75% better than FTL-CH. In San Francisco, PP-LH guarantees that 11.7% of pickups occur almost instantaneously, roughly within thirty seconds of the ride requests, since our cells are of length $\epsilon = 100$ meters. The average rewards using PP-LH for Chicago, Los Angeles, and New York yield similar results of $11.1\%, 7.2\%, \;\&\; 10.2\%$ respectively. Los Angeles has the lowest expected reward as well as the lowest average correlation fractal dimension D_2 (Figure 1b).

Discussion & Future Work: We considered three existing algorithms for the real-time vehicle placement problem. Even though PP-LH performed better than the other two algorithms, in comparison to OPT, as shown in Figure 3, there is still much room for further improvement. The reward function for OPT is computed using actual data; for every drop-off OPT assumes perfect knowledge of future pickups in the neighbouring cells.

There are several important insights from this work. 1) History of past pickups and drop-offs helps; although when accompanied with some sort of randomization does not necessarily help to boost the performance of the algorithm. For instance, uniform random selection of the leader as in the case of our FTL-CH, does as good as selecting a leader with the lowest index [4]. 2) We believe randomization is essential but some weighting mechanism needs to be considered which can be characterized with the help of self-similarity. 3) Having limited most recent history is almost as good as having complete history. One obvious extension is to consider vehicle placement actions not just in the next time snapshot but on a longer time scale.

References

[1] V. Alfi, G. Parisi, and L. Pietronero. Conference registration: how people react to a deadline. *Nature Physics*, 3(11):746–746, 2007.

[2] A.-L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[3] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using thecorrelation'fractal dimension. Technical report, 1998.

[4] A. Blum and Y. Monsour. Learning, regret minimization, and equilibria. 2007.

[5] C. Castellano, S. Fortunato, and V. Loreto. Statistical physics of social dynamics. *Reviews of modern physics*, 81(2):591, 2009.

[6] S. De Rooij, T. Van Erven, P. D. Grünwald, and W. M. Koolen. Follow the leader if you can, hedge if you must. *Journal of Machine Learning Research*, 15(1):1281–1316, 2014.

[7] M. Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[8] A. Jauhri, B. Foo, J. Berclaz, C. C. Hu, R. Grzeszczuk, V. Parameswaran, and J. P. Shen. Space-time graph modeling of ride requests based on real-world data. *arXiv preprint arXiv:1701.06635*, 2017.

[9] Y. Jia, W. Xu, and X. Liu. An optimization framework for online ride-sharing markets. *arXiv preprint arXiv:1612.03797*, 2016.

[10] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[11] F. Miao, S. Han, A. M. Hendawi, M. E. Khalefa, J. A. Stankovic, and G. J. Pappas. Data-driven distributionally robust vehicle balancing using dynamic region partitions.

[12] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering*, 13(2):463–478, 2016.

[13] J. G. Oliveira and A.-L. Barabási. Human dynamics: Darwin and einstein correspondence patterns. *Nature*, 437(7063):1251–1251, 2005.

[14] G. Proietti and C. Faloutsos. I/o complexity for range queries on region data stored using an r-tree. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 628–635. IEEE, 1999.

[15] M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong. A cost-effective recommender system for taxi drivers. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 45–54. ACM, 2014.

[16] D. Zhang, T. He, S. Lin, S. Munir, and J. A. Stankovic. Dmodel: Online taxicab demand model from big sensor data in a roving sensor network. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 152–159. IEEE, 2014.

[17] B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 322–331. ACM, 2008.