USING REINFORCEMENT LEARNING TO MINIMIZE TAXI IDLE TIMES

Kevin O’Keeffe
Senseable City Lab, Massachusetts Institute of Technology
Cambridge, MA 02139
kkeeffe@mit.edu

Sam Anklesaria
Senseable City Lab, Massachusetts Institute of Technology
Cambridge, MA 02139

Paolo Santi
Senseable City Lab, Massachusetts Institute of Technology
Cambridge, MA 02139

Carlo Ratti
Senseable City Lab, Massachusetts Institute of Technology
Cambridge, MA 02139

ABSTRACT

Taxis spend a significant amount of time idle, searching for passengers. The routes
vacant taxis should follow in order to minimize their idle times are hard to calcu-
late; they depend on complex quantities like passenger demand, traffic conditions,
and inter-taxi competition. Here we explore if reinforcement learning (RL) can
be used for this purpose. Using real-world data to characterize passenger demand,
we show RL-taxis indeed learn to how to reduce their idle time in many environ-
ments. In particular, a single RL-taxi operating in a population of regular taxis
learns to out-perform its rivals by a significant margin.

1 INTRODUCTION

Even during peak hours, taxis in Manhattan spend one third of their driving time idle [Schaller 2017],
looking for passengers. Estimates of idle times for taxis in other cities are as high as 60% [Powell
et al. 2011]. This inefficiency bears much cost: it worsens congestion for commuters, reduces
revenue for taxi companies, and increases pollution to the environment. A reduction in taxi idle
times would thus confer much benefit.

Reducing taxi inefficiency amounts to an optimal path finding problem: when vacant, what path
should a taxi follow to minimize its idle time, the total time it spends looking for passengers? Most
previous research found step-wise optimal solutions to this problem [Zhang et al. 2014; Qu et al.
2014; Dong et al. 2014; Huang et al. 2014] – ‘greedy’ paths which minimize the time until the
next passenger pick-up only. But because the idle time is a cumulative quantity calculated over a
given reference period – and therefore depends on the full trajectory a taxi traces out in both serving
and searching for multiple passengers – these greedy paths are not necessarily globally optimal.
A recent work [Yu et al. 2019] has improved upon this situation by formulating the problem as
a Markov Decision Process (MDP), the natural framework to optimize cumulative, and not just
instantaneous, quantities / objectives. Using taxi data from Shanghai, they showed this formalism
performed better than common baselines.

In order for MDPs to be effective, however, an accurate model of the environment must be provided.
For taxis moving in cities, this is a challenging task. The relevant features, such as passenger de-
mand, traffic conditions, and inter-taxi competition, are complicated, hard-to model quantities. The
ability of MDPs to minimize the idle times of real-world taxis could thus be limited.

Reinforcement learning (RL) could overcome this limitation. It is a type of MDP where, ideal for
taxis moving in cities (and other complex systems), a model of the environment is not required.
Instead, the optimal behavior for the (unspecified) environment is learned through trial and error.
When combined with modern techniques from deep learning, so called deep-RL algorithms have
been shown to be dramatically powerful. They have been used to train autonomous helicopters [Ng
et al. 2006; Abbeel et al. 2007], to achieve super human performance in the board game Go [Silver
et al. 2017], and to train a robotic hand to solve a Rubix cube [Andrychowicz et al. 2018].
It’s natural to wonder if these successes could be translated to taxi research. Several researchers have started to explore this possibility by applying deep-RL in various contexts, such as optimizing taxi carpooling (Jindal et al., 2018), dispatching (Oda and Joe-Wong, 2018), and route scheduling (Shi et al., 2018). In this work, we continue this endeavour by studying if deep-RL could be useful in minimizing taxi idle times. A first step in this direction was taken in Han et al. (2016), where a single RL-taxi on an otherwise empty street network from Singapore was trained using tabular Q-learning (a basic form of RL which doesn’t rely on deep learning). Here we extend this work by using more powerful RL algorithms in new environments (i) a single RL-taxi moving on street networks from several real world cities, and (ii) a single RL-taxi moving in a population of regular (non-RL) taxis. We find RL-taxis can indeed learn to appreciably lower their idle times, as compared to other baseline taxi models. Our results are potentially relevant to taxi companies, intelligent transport systems, and other settings in which vehicular efficiency is important.

2 RESULTS

2.1 BACKGROUND MATERIALS

Mathematical preliminaries. We begin with a definition of an MDP, which models an agent interacting with an environment. The agent’s interactions leads to rewards, which the agents tries to maximize. Mathematically, an MDP is defined by a 4-tuple \((\mathcal{S}, \mathcal{A}, P, R)\), where \(\mathcal{S}\) is a set of states which characterize the environments, \(\mathcal{A}\) is a set of actions available to the agent, \(P_{a}(s,s')\) is a transition function, which describes the probability of the environment going from state \(s\) to \(s'\) when the agent takes action \(a\), and \(R_{a}(s,s')\) is a function which generates a reward \(r\) during this transition from \(s\) to \(s'\). The goal of an MDP problem is for an agent to learn a policy \(\pi(a,s)\) which describes the probability of selecting action \(a\) when in state \(s\). More specifically, the goal is to learn an optimal policy \(\pi^*\) which maximizes the cumulative reward (sometimes called the return) \(G = \sum_{t}^{T} r_{t}\), where \(T\) is the length of the ‘episode’ (the total number of steps or time units considered).

Having defined an MDP we can now define RL: As mentioned in the Introduction, it is simply ‘model-free’ MDP. In other words, an MDP where the system transition function \(P_{a}\) is unknown. Put broadly, then, the purpose of RL is to approximate the optimal policy \(\pi^*\) when \(P_{a}\) is either unknown or too complicated to describe analytically. Many algorithms to find \(\pi^*\) exists, and finding new algorithms is an active area of research. Generally speaking, the algorithms fall into two classes (a) policy gradient methods and (b) value based methods. In (a) the policy \(\pi_{\theta}(s,a)\) is modeled directly by some function approximator \(f_{\theta}(s,a)\) (in deep-RL this is a neural network) with parameters \(\theta\). This is updated / improved directly via gradient descent methods; that is, by relating \(\nabla_{\theta} J\) to \(\nabla_{G}\) (intuitively, by moving in the direction in \(\theta\)-space which maximizes the return \(G\)). In (b), the policy is defined indirectly. First, a \(Q\)-function is learned, which is defined as the expected return of taking action \(a\) and when in state \(s\): \(Q(s, a) = E(G) = E(\sum_{t}^{T} r_{t})\). The optimal policy is then given by \(\pi(s,a) = \arg\max_{a} Q(s,a')\). In deep-RL the \(Q\)-function is modeled with a neural network. In contrast to policy gradients methods (a), the \(Q\) function is updated recursively via the Bellman equation: \(Q(s,a) = r + \max_{a'}Q(s,a')\), which can be derived by conditioning on the expectation in the definition of \(Q(s,a)\). For a more detailed explanation of these methods, and RL generally, see Sutton and Barto (2018).

Problem formulation. We are now ready to formulate the taxi idle time minimization problem as an MDP. We consider taxis moving in a street network \(\mathcal{S}\) whose edges represent street segments, and whose nodes represent intersections. We assume each street segment has unit length and taxis travel with constant speed equal to one segment per unit time. The state space \(\mathcal{S}\) are the set of positions the taxis can take, or the nodes in the graph. As is common in RL studies, we represent each state by a one-hot vector \(s = (0, 1, 0, 0, \ldots, N_{\text{nodes}})\) where \(N_{\text{nodes}}\) is the number of nodes in the graph and \(s_{j} = 1\) if the taxi is at node \(j\) (although for some environments we consider, different state vectors will be chosen). The action space is \(\mathcal{A} = \{1, \ldots, k_{\text{max}}\}\), \(k_{\text{max}}\) is the maximum node degree in the street network. Selecting action \(a = j\) means the taxi moves to the \(j\)’th neighbour of the current node. Because the number of neighbour a given node is variable, not all action will be ‘legal’ – selecting \(a = 5\) on a node with only two neighbours will be meaningless. In this case, the taxi stays at the current positions and receives a negative reward \(r = -r_{\text{penalty}} = -2\), which will discourage the agent taking this action in the future. In the case where a legal action is taken, the taxi moves to the next node. If a passenger trip is generated there (which happens with a probability specified
Figure 1: Histogram of idle times \( \tau \) for a taxi with policies \( \pi^*, \pi_{\text{greedy}}, \) and \( \pi_{\text{random}} \) on four street networks studied. The optimal policy \( \pi^* \) gives significantly lower idles times than the other policies. The ensemble size for each plot was 1000.

later), a positive reward \( r_{\text{success}} = L \), is generated where \( L \) is the duration of the trip. If no trip is generated, a reward \( r = 0 \) is generated. Finally, as discussed the taxi moves according to a policy \( \pi(s,a) \), whose optimization (in terms of minimizing the idle time), is the goal of the paper.

We next consider two specific environments.

2.2 Environment 1: Single Taxi

Model environment \( \mathcal{M} \). We begin with the simplest environment \( \mathcal{M} \): a single taxi moving on an otherwise empty street network (i.e. no other taxis present). We make the further simplification of there being a constant probability \( p_i \) of a passenger being generated at the \( i \)-th node. This amounts to specifying the transition function \( P_a \) – to providing a model for the environment – which in turn reduces the problem to a classic MDP. So strictly speaking RL is not needed in this case. Our reason for studying this simple scenario is that the optimal \( \pi^* \) can be derived explicitly in this case, and thus can be used as a baseline to test the RL algorithms against, as well as for the more complicated \( N \) taxi case. Finally, we also assume street segments have unit length and that passengers persist for precisely one time unit at each node (in Appendix B we show how these simplifications can be relaxed).

We now derive \( \pi^* \). Let the random variable \( X_i \) be the time taken for an empty taxi to find the next passenger having started at node \( i \). Let the indicator random variable \( I_i \) be 1 if our taxi picks up a passenger at street \( i \) within one time-step. If all streets take 1 time-step to traverse and we continue driving until we find a passenger

\[
X_i = I_i + (1 - I_i)(1 + \min_{j \in N(i)} X_j)
\]

(1)

where \( N(i) \) gives the outgoing neighbors of street \( i \). Applying the expectation operator to the above gives

\[
x_i = p_i + (1 - p_i) \left( 1 + \min_{j \in N(i)} x_j \right)
\]

(2)

where \( x_i \) is the expected waiting time for a taxi on street \( i \). Eq. (2) can be interpreted as a relaxation algorithm for \( x_t \), in other words, can be solved recursively to find \( x_t \). Let \( x_{i,t} \) be our approximation of \( x_i \) at time \( t \). Then we can find a better approximation via \( x_{i,t+1} = p_i + (1 - p_i) \left( 1 + \min_{j \in N(i)} x_{j,t} \right) \).

Starting from arbitrary values for \( x_{i,0} \), we continue iterating until we reach a fixed point. Once these \( x_i \) are known, the optimal route is found by moving to minimize waiting time: when at node \( i \), node \( j^* = \arg \min_{N, x_j} \) is selected. The optimal policy then corresponds to the optimal route: \( \pi^*(j|i) = 1 \) which implies \( \pi(j|i) = 0 \) for all other \( j \in N_i \). Note we have abused notation slightly here by using \( \pi(j|i) \) to denote the probability of moving to node \( j \) when at node \( i \) (this is abusive since we earlier defined the policy is term of a state \( s \) and action \( a \), \( \pi(s,a) \), so by \( \pi(j|i) \) we mean \( \pi(s = i, a') \) where \( a' \) denotes to node \( j \) is selected when at node \( i \)).

We tested \( \pi^* \) using real-world taxi data from three cities: NYC (confined to Manhattan), San Francisco, and Singapore. We also considered a toy ‘grid’ street network consisting of a square grid of length \( n = 5 \) (so 25 nodes total), making four cities in total. In Appendix A we describe the datasets and how they were used to approximate the \( p_i \).

The street networks themselves were found using the python package ‘osmnx’. Figure 1 shows the distribution of idle times of taxis following \( \pi^* \). Two baselines are also shown: the idle times
for taxis following a random policy $\pi_{\text{random}}$, in which a taxi moves randomly on the graph, and a greedy policy $\pi_{\text{greedy}}$, in which when at node $i$, the taxi moves to neighbouring node with highest $p_i$. As can be seen, the optimal policy gives substantially lower idles times.

Next we explored how well $\pi^*$ can be approximated by an RL-taxi. We use three popular deep-RL algorithms: deep Q-networks (DQN) [Mnih et al., 2013], Advantage-Actor-Critic (A2C) [Sutton and Barto, 2018], and proximal policy optimization (PPO) [Schulman et al., 2017]. DQN were first used to achieve superhuman performance an Atari games [Mnih et al., 2013], one of the first big successes of deep-RL. A2C and PPO came later, and are popular choices in contemporary research, having strong performance and relatively easy implementations. Figure 2 shows the training curves of these algorithms when used on a taxi moving in $\mathcal{M}$ along with the baseline optimal $\tau$. The results are encouraging: the RL-taxi learn policies that give close to optimal $\tau$ on all cities. A2C performs best on all cities, but takes the longest to train. PPO on the other hand trains quickest, while the DQN is slow and unstable. The hyperparameters used for the algorithms are described in Appendix C.

2.3 N TAXIS

Next we study the $N > 1$ taxi regime. We first explore if $\pi^*$ – which, recall, the optimal policy when a single taxi moves on the street network $S$ – is useful in this case. We suspect that in the low density regime, when the probability of two taxis being at the same node is small, that $N$ taxis moving according to $\pi^*$ might still lead to low $\tau$. In Figure 3, we test this by showing the mean idle time of a population of taxis moving in $\mathcal{M}$ according to $\pi^*$ versus $N_{\text{taxis}}/N_{\text{nodes}}$ – a measure of density. We also show $\langle \tau \rangle$ for three other populations: a population of random taxis, a population of greedy taxis, and a mixed population of equal numbers of $\pi^*$-taxis, greedy taxis, and random taxis. Interestingly, for the grid graph $\pi^*$-taxis and greedy taxis perform worse than random taxis when the density is large. In Singapore, the $\pi^*$ population operates worse than the other populations for all densities, while for the remaining cities all taxi types plateau at large inefficiencies at large densities. These results show there is much room for improvement for an RL-taxi.

For simplicity, we restrict our analysis to the easier case of just one RL-taxi in a population of regular taxis (taxis following a deterministic policy). This lets us avoid the much more difficult multi-agent RL problem, in which game-theoretic complexities arise. For example, because intelligent agents can change their behavior, the idea of a static optimal policy becomes ill-defined (the policy which beats opponent $A$ at some stage during training might later become ineffective as $A$ changes its policy during learning, and so on). More sophisticated algorithms are needed to train agents capable of handling these complications, which we leave for future work.

Figure 4 shows how a single RL-taxi out performs in a population of $\pi^*$-taxis for all the cities when the density is $N_{\text{taxis}}/N_{\text{nodes}} = 2$. Similar results are found when the RL-taxi operates in a population of greedy taxis and random taxis, and for other values of the density. We were curious what strategy the RL-taxi followed to beat the other taxis. To this end, in Figure 5, we plotted the relative amount of times the $\text{RL}$-taxi (trained using PPO) spends at each node, $C_{i}^{\text{RL}}$, as well as that of the optimal cab $C_{i}^{\text{optimal}}$ versus the passenger generation rate $p_i$ for each city (ideally we would plot the $C_i$ over the street networks too, but it was challenging to plot both the networks along with $C_i$ and the passenger generation rates $p_i$). For all but the Singapore street network, we see that the RL-taxi learns to exploit a few nodes that are not very popular in terms of $p_i$, but are visited less than the other cabs. For Singapore however, we see the $\text{RL}$-taxi concentrates on a few popular nodes.
Figure 3: Mean idle time of a population of taxis following deterministic policies discussed in the main text. Note the mean is with respect to the population; just one simulation of length $T = 8640$ (this $T$ is the same for all simulations run in the paper) was run for each data point. The population size $N_{taxis}$ is plotted relative to the number of nodes in the graph $N_{nodes}$ (values for $N_{nodes}$ are reported in Appendix A), which is a measure of the density of taxis on the graph. For low densities, the optimal (optimal for the single agent case) $\pi^*$ policy has low idle times. But this performance degrades at higher densities.

Figure 4: Training curves for all a single RL-taxi operating in a population of $N_{taxis} = 2N_{nodes}$ taxis following the $\pi^*$ policy. The RL-taxis learns to significantly outperform the other taxis.

Figure 5: Fraction of time taxis spend at each node $C_i$ versus that node’s passenger generation rate $p_i$ for all cities. The RL-taxi was trained using the PPO algorithm with hyper parameters given in Appendix C. Only the 10 largest $C_i$ are shown.
3 DISCUSSION

Our main goal was to explore if RL could be useful for minimizing taxi idle times. To achieve this, we studied taxis moving in two toy environments: a single RL-taxi moving on real-world street networks with realistic passengers demand, and a single RL-taxi moving in a population of regular taxis moving with simple rules. In these idealized settings, we found that indeed RL-taxis learned to minimize their idle times; in the first setting, the PPO algorithms achieved close to optimal behavior in each city studied, and in the second it achieved significantly lower $\tau$ (as compared with the other non-RL taxis). Previous work [Han et al. (2016)] studied the first case, but with basic Q-learning (non-deep) RL and on only one city. And to our knowledge, no other work has studied our multi-agent set-up. Taken together, our findings indicate that RL could be useful in improving the efficiency of real-world taxis. They are also potentially useful to other aspects of taxi research, such as route planning when serving passengers [Li et al. (2009); Koeners et al. (2011)] (as opposed to route planning when searching for passengers) or mobile sensing [O’Keeffe et al. (2019); O’Keeffe et al. (2019); Skordylis and Trigoni (2011); Piran et al. (2011)].

The most pertinent direction for future work would be relax the idealizations we made in our study. For example, a more realistic model for the non-RL taxis, such as the Intelligent Driver Model [Treiber et al. (2000)] could be used. The same is the true for the passenger demand, which we characterized with a simple Poisson process; real passenger demand most likely has temporal variations – a complication which could potentially be tackled with distributional RL [Bellemare et al. (2017)]. Better models for pricing (which we ignored here), and for inter-taxi competition (which we modeled as a simple coin toss) would also be interesting. Finally, a fully multi-agent RL study could be executed, in which a fleet of RL-taxis are trained simultaneously. Effective algorithms for multi-agent RL have recently been developed, which could prove fruitful for this purpose [Lowe et al. (2017); Rashid et al. (2018); Sunehag et al. (2017)].

REFERENCES

B. Schaller, Schaller Consulting, December 1 (2017).

J. W. Powell, Y. Huang, F. Bastani, and M. Ji, in International Symposium on Spatial and Temporal Databases (Springer, 2011), pp. 242–260.

D. Zhang, T. He, S. Lin, S. Munir, and J. A. Stankovic, IEEE Transactions on Parallel and Distributed Systems 26, 3122 (2014).

M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong, in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (ACM, 2014), pp. 45–54.

H. Dong, X. Zhang, Y. Dong, C. Chen, and F. Rao, in 17th International IEEE Conference on Intelligent Transportation Systems (ITSC) (IEEE, 2014), pp. 2003–2008.

J. Huang, X. Huangfu, H. Sun, H. Li, P. Zhao, H. Cheng, and Q. Song, IEEE transactions on Knowledge and Data Engineering 27, 46 (2014).

X. Yu, S. Gao, X. Hu, and H. Park, Transportation Research Part B: Methodological 121, 114 (2019).

A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, in Experimental robotics IX (Springer, 2006), pp. 363–372.

P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, in Advances in neural information processing systems (2007), pp. 1–8.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Nature 550, 354 (2017).

M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., arXiv preprint arXiv:1808.00177 (2018).

I. Jindal, Z. T. Qin, X. Chen, M. Nokleby, and J. Ye, in 2018 IEEE International Conference on Big Data (Big Data) (IEEE, 2018), pp. 1417–1426.
T. Oda and C. Joe-Wong, in IEEE INFOCOM 2018-IEEE Conference on Computer Communications (IEEE, 2018), pp. 2708–2716.

D. Shi, J. Ding, S. M. Errapotu, H. Yue, W. Xu, X. Zhou, and M. Pan, in 2018 IEEE Global Communications Conference (GLOBECOM) (IEEE, 2018), pp. 1–7.

M. Han, P. Senellart, S. Bressan, and H. Wu, in Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (ACM, 2016), pp. 2421–2424.

R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction (MIT press, 2018).

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, arXiv preprint arXiv:1312.5602 (2013).

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, arXiv preprint arXiv:1707.06347 (2017).

Q. Li, Z. Zeng, B. Yang, and T. Zhang, in 2009 17th International Conference on Geoinformatics (IEEE, 2009), pp. 1–5.

G. Koeners, E. Stout, and R. Rademaker, in 2011 IEEE/AIAA 30th Digital Avionics Systems Conference (IEEE, 2011), pp. 2C3–1.

K. P. O'Keeffe, A. Anjomshoaa, S. H. Strogatz, P. Santi, and C. Ratti, Proceedings of the National Academy of Sciences 116, 12752 (2019).

K. O’Keeffe, P. Santi, B. Wang, and C. Ratti, arXiv preprint arXiv:1901.08678 (2019).

A. Skordylis and N. Trigoni, IEEE Transactions on Intelligent Transportation Systems 12, 680 (2011).

M. J. Piran, G. R. Murthy, G. P. Babu, and E. Ahvar, in 2011 Third International Conference on Computational Intelligence, Modelling & Simulation (IEEE, 2011), pp. 388–393.

M. Treiber, A. Hennecke, and D. Helbing, Physical review E 62, 1805 (2000).

M. G. Bellemare, W. Dabney, and R. Munos, in Proceedings of the 34th International Conference on Machine Learning-Volume 70 (JMLR. org, 2017), pp. 449–458.

R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, in Advances in Neural Information Processing Systems (2017), pp. 6379–6390.

T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, arXiv preprint arXiv:1803.11485 (2018).

P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sannerat, J. Z. Leibo, K. Tuyls, et al., arXiv preprint arXiv:1706.05296 (2017).

san fran dataset, http://crawdad.org/dartmouth/campus/20090909 accessed: 01/04/2016.

P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, Proceedings of the National Academy of Sciences 111, 13290 (2014).
A Datasets

Data sets

Description of datasets We used 3 real-world data sets from: New York City (confined to the borough of Manhattan), San Francisco, and Singapore. The New York data set has been obtained from the New York Taxi and Limousine Commission for the year 2011 via a Freedom of Information Act request. The Singapore data set were provided to the MIT SENSEable City Lab by AIT and the Singapore government, respectively. The San Francisco data sets was publicly available. We note these are the same datasets used in previous studies Santi et al. (2014); O’Keeffe et al. (2019) (and so the descriptions of these datasets overlap with those given in those works).

Each data set consists of a set of taxis trips. Each trip \( i \) is represented by a GPS coordinate of pickup location \( O_i \) and dropoff location \( D_i \) (as well as the pickup times and dropoff times). We snap these GPS coordinates to the nearest street segments using OpenStreetMap. We do not however have details on the trajectory of each taxi – that is, on the intermediary path taken by the taxi when bringing the passenger from \( O_i \) to \( D_i \). So we need to approximate trajectories. We used two methods for this, one sophisticated, one simple. The sophisticated method was for the Manhattan data set. Here, as was done in Santi et al. (2014), we generated 24 travel time matrices, one for each hour of the day. An element of the matrix \( (i,j) \) contains the travel time from intersection \( i \) to intersection \( j \). Given these matrices, for a particular starting time of the trip, you pick the right matrix for travel time estimation, and compute the shortest time route between origin and destination; that gives an estimation of the trajectory taken for the trip. For the remaining cities, we used the simple method of finding the weighted shortest path between \( O_i \) and \( D_i \) (where segments were weighted by their length).

The temporal range of the data sets was not uniform. NYC was the most comprehensive, consisting of all the taxi trips in Manhattan starting on 2011. The remaining data sets were for one week. One day’s worth of data was also used in the paper: 01/18/11, 05/21/08, 02/21/11, for NYC, San Francisco, Singapore. The sizes of the street networks were \( N_{nodes} = (3564, 8014, 16289) \) in the same order. Training RL-taxis on the entire street networks was prohibitively costly. So we sub-sampled the street networks by selecting a point at the city center and taking all nodes within a distance \( R \) of this point. The python commands to find the networks is given in the code block below. The result sub-networks were also smaller that \( N_{nodes} = 1000 \).

```python
import osmnx as ox

def load_sub_graph(city):
    if city == 'nyc':
        point, R = (40.7896239, -73.9598939), 2*10**3
        G = ox.graph_from_point(point, distance=R, network_type='drive')
    elif city == 'sanfran':
        R = 1.5*10**3
        G = ox.graph_from_point((37.756986, -122.439705), distance=R, network_type='drive')
    elif city == 'singapore':
        R = 6.0*10**3
        G = ox.graph_from_point((1.223439, 103.819684), distance=R, network_type='drive')
    return G
```

Estimating trip probabilities \( p_i \). We assume passenger trips were generated via a poisson process

\[
P(N_i = n, t) = \frac{(\lambda_i t)^n}{n!} e^{-\lambda_i t}
\]  

(3)
The probability of at the probability of there being at least one trip generated at node \( i \) per time unit is then
\[
p_i = \mathbb{P}(N_i > 0, t = 1) = 1 - \mathbb{P}(N_i = 0, t = 1) \quad (4)
\]
\[
p_i = 1 - e^{-\lambda_i t} \quad (5)
\]
We estimated the rate \( \lambda_i \) for the \( i \)-th node by counting the number of trips which start at that node, \( C_i \), and divide by the number of times units in a day, \( T \) (we take a day as our reference period). The average time taken for a taxi to traverse a segment is 10 s. So for convenience we measured time in deci-seconds so that taxis move at unit speed. This means \( T = 8640 \). Figure 6 shows the estimated \( p_i \).

B GENERALIZATIONS TO ENVIRONMENT \( \mathcal{M} \)

When studying model \( \mathcal{M} \) is the main text, we made three simplifications: (i) Passengers trip were generated at each time step with probability \( p_i \), and then disappear afterwards (i.e., passenger do not wait at nodes) (ii) Segments were all unit length. We here show these are easily relaxed.

Passenger waiting times. This is easily handled. Assume passengers wait for \( \delta \) time-steps after generation. If there is no competition among taxi drivers, this is simply equivalent to increasing the probabilities at each node by a factor of \( \delta \), \( p_i \rightarrow \delta p_i \). Everything else remains the same.

Variable street lengths. Non-unit street lengths means that trip generation must be modeled as a Poisson random variables, as opposed to Bernoulli random variables. This is because a passenger can be picked up at any point along the street segments, whereas before, street segments were treated as one dimensional points. Let \( \lambda_i \) be the rate of trip generation. We need to know the probability that the exponential variable is greater than the street length \( L \). This is given by complementary CDF of the exponential distribution: \( 1 - e^{\lambda x} \). This new form of the fixed point equation is this
\[
x_i = (1 - e^{-\lambda_i L_i})(\lambda_i^{-1} - l(e^{\lambda_i L_i} - 1)^{-1}) + e^{-\lambda_i L_i} \left( L_i + \min_{j \in N(i)} x_j \right) \quad (6)
\]

C HYPERPARAMETERS

We here discuss the hyperparameters for each RL algorithm used.

DQN. The neural network used had three Dense layers of dimension 64. The first two layers had ReLU activations. The final layer had a linear activation. A mean square loss was used with an Adam’s optimizer with learning rate 0.01. The reward discount factor was 0.99. Following Mnih et al. (2013), we used both a behavior and target network. The target network was updated ‘softly’ every learning step according to \( \theta_{\text{target}} = \tau \theta_{\text{behavior}} + (1 - \tau)\theta_{\text{target}} \) with \( \tau = 0.01 \) where \( \theta_i \) denotes the parameters in neural network \( i \). The DQN-agent had memory size \( 10^4 \) and batch size 32 was used. During training, actions were selected according to an \( \epsilon \)-greedy policy with \( \epsilon = 0.5 \). This value was decayed by factor \( \delta = 0.999 \) every episode, to a minimum value of 0.05. Learning was performed every timestep. These values of hyperparameters were found to be optimal (in the sense they produced the lower idle times) on the NYC network. Experiments showed these values were approximately optimal on the other cities too, so these values were frozen throughout the work. See Mnih et al. (2013) for a review of DQN’s and the meaning of these hyperparameters.
A2C. The actor network had three Dense layers of dimension 32. The first two layers had ReLU activations, while the final layer had softmax. The critic network had the same architecture, except the final layer had dimension 1 and a linear activation (to return the $Q$-value). Both networks used a Adam’s optimizer with learning rate 0.01. The discount factor was $\gamma = 0.99$. Learning was applied at the end of every episode (in contrast to the DQN, which was done at every timestep in each episode). As for the DQN these values of hyperparameters were found to be optimal (in the sense they produced the lower idle times) on the NYC network and used throughout the work. See Sutton and Barto (2018) for a description of the A2C algorithm.

PPO. The actor network had three Dense layers of dimension 20. The first two layers had $tanh$ activations while the last had a softmax. The critic network is the same as the actor network with a linear activation in the final layer of dimension 1. An Adam’s optimizer was used for both networks. In contrast to the DQN and A2C, different hyperparameter values were used in each city (because experiments showed significantly different results at different values). The hyperparameters were: the learning rate $lr$, loss clipping values $lc$, entropy loss term $c_1$, batch size, and number of epochs used during each learning step (PPO uses more than one gradient step / epoch per learning step). These hyperparameters were found by sampling 20 times from the following hypergrid:

```plaintext
lrs = [0.1, 0.01, 0.001]
gammas = [0.1, 0.5, 0.9, 0.99]
taus = [0.1]
loss_clips = [0.5, 0.2, 0.1, 0.05, 0.01]
c1s = [0.1, 0.01, 0.001, 0.0001]
num_epochs = [2, 4, 8, 16]
batchsize = [16, 32, 64]
```

The optimal values for the grid, NYC, San Francisco, and Singapore street networks were:

- $lr, gamma, loss_clipping, c1, batchsize, num epochs = 0.001, 0.9, 0.1, 0.1, 64, 16$
- $lr, gamma, loss_clipping, c1, batchsize, num epochs = 0.001, 0.9, 0.1, 0.1, 64, 16$
- $lr, gamma, loss_clipping, c1, batchsize, num epochs = 0.001, 0.9, 0.1, 0.1, 64, 16$
- $lr, gamma, loss_clipping, c1, batchsize, num epochs = 0.001, 0.99, 0.2, 0.01, 16, 2$