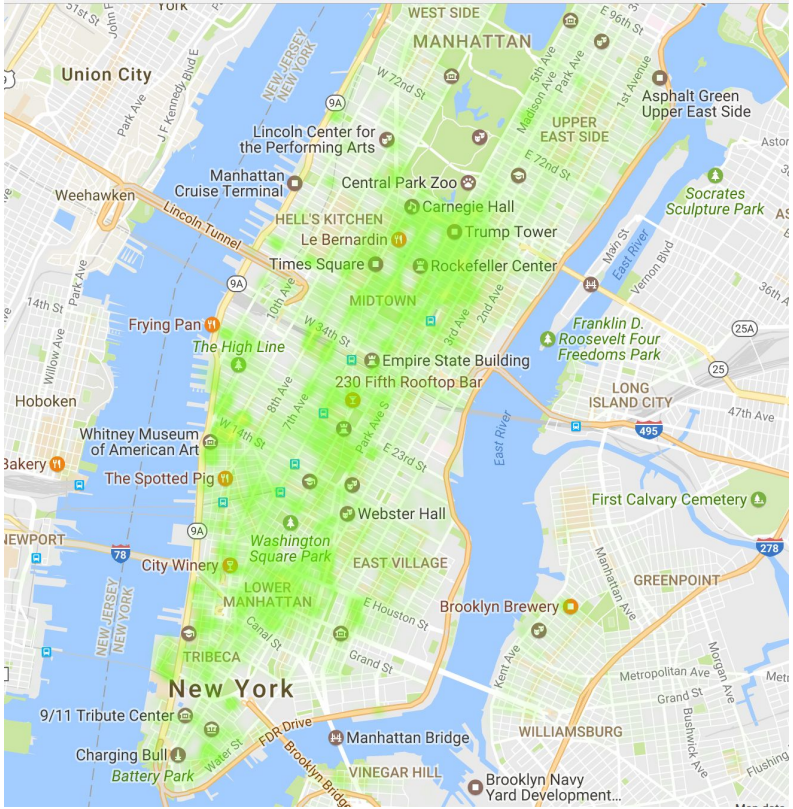
Uber: Minimizing ETA

Anna Juchnicki
Christina Nguyen
Sasha Sklyarenko
Faith Yu

Problem/Goal

- Dataset from uber contains times and coordinates of incoming requests in New York City

- We want to figure out how to schedule these requests and assign them to drivers



Formulation

Is it possible to formulate as a scheduling problem studied in class?

m	Assume we know number of cars at any certain moment
p_j	Time it takes to travel to pickup location
r_j	Expected release dates/times are calculated from historical data
d_j	Fake deadline of 5 minutes

Formulation

Objective: minimize sum of lateness of arrival to the pickup location

Matches a problem we studied in class:

$$P_m \mid r_j, d_j \mid \Sigma T_j$$

Our Assumptions

- Only focus on rides within Manhattan (our data was in the lower half)
- One customer per ride (no Uber Pool)
- Vehicles move at a constant speed on a strict grid
- If a vehicle is not serving a customer, it will either:
  - Become idle
  - Drive towards the nearest predicted high-demand area
  - Drive around randomly (Uber found this is the least optimal)
- Average ride distance is 1 mile, average car speed is 10mph -> approximately 6 minutes per ride
  - This is assuming there is no traffic and the driver can get directly to the destination
  - Taking into account Manhattan traffic, an average trip is probably around 12 minutes

Vehicle Scheduling on a Tree to Minimize Maximum Lateness:
Karuno, Nagamochi, Ibaraki

- Goal: Routing schedule that minimizes the maximum lateness L_{max} of a single vehicle traveling on a tree-shaped network T
  - Each vertex v has due date $d(v)$ and processing time $p(v)$, travel time $w(v, u)$ and $w(u, v)$ associated with each edge $(u, v) \in$ set of edges E
  - Vehicle starts at initial vertex v_0, visits all tasks **v** (w/o preemption) for their **p(v)** and returns to the initial vertex
  - Each edge ends up being traversed exactly two times (u to v, v to u)

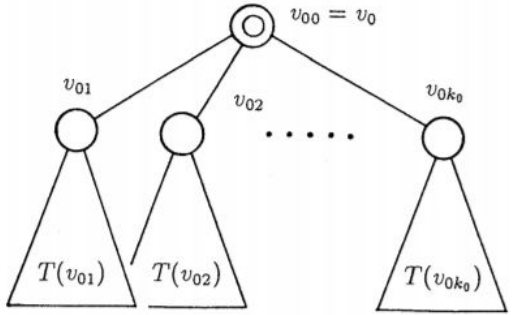


Figure.1 The subtrees rooted at v_0.

- Polynomial time algorithm w/ depth-first routing constraint

procedure SEARCH(v_q; $L_0(v_q), W(v_q), P(v_q), \pi^{DF}(v_q)$)

begin

k_q :=(the number of children of v_q in T);

v_p :=(the parent of v_q in T. Let $v_p = v_0$ if $v_q = v_0$.);

if $k_q = 0$ **then**

$L_0(v_q) := w(v_p, v_q) + p(v_q) - d(v_q)$;

$W(v_q) := w(v_p, v_q) + w(v_q, v_p)$; $P(v_q) := p(v_q)$;

$\pi^{DF}(v_q) := (v_q)$;

return

else

(Computation of optimal schedule at v_q)

$L_0(v_{q,0}) := p(v_q) - d(v_q)$; $W(v_{q,0}) := 0$; $P(v_{q,0}) := p(v_q)$;

$\pi^{DF}(v_{q,0}) := (v_q)$;

for $i := 1$ **to** k_q

begin

SEARCH($v_{q,i}$; $L_0(v_{q,i}), W(v_{q,i}), P(v_{q,i}), \pi^{DF}(v_{p,i})$);

end;

Compute a permutation ψ on $\{0, 1, .., k_q\}$ such that it satisfies

$M_0(v_{q,\psi(0)}) \geq M_0(v_{q,\psi(1)}) \geq \ldots \geq M_0(v_{q,\psi(k_q)})$;

$\pi^{DF}(v_q) := (\pi^{DF}(v_{q,\psi(0)}), \pi^{DF}(v_{q,\psi(1)}), ..., \pi^{DF}(v_{q,\psi(k_q)}))$;

(Computation of $L_0(v_q), W(v_q)$ and $P(v_q)$)

$LATE := 0$; $WEIGHT := w(v_p, v_q)$; $PROC := 0$;

for $i := 0$ **to** k_q

begin

$LATE := \max\{LATE, L_0(v_{q,\psi(i)}) + WEIGHT + PROC\}$;

$WEIGHT := WEIGHT + W(v_{q,\psi(i)})$;

$PROC := PROC + P(v_{q,\psi(i)})$

end;

$L_0(v_q) := LATE$; $W(v_q) := WEIGHT + w(v_q, v_p)$; $P(v_q) := PROC$;

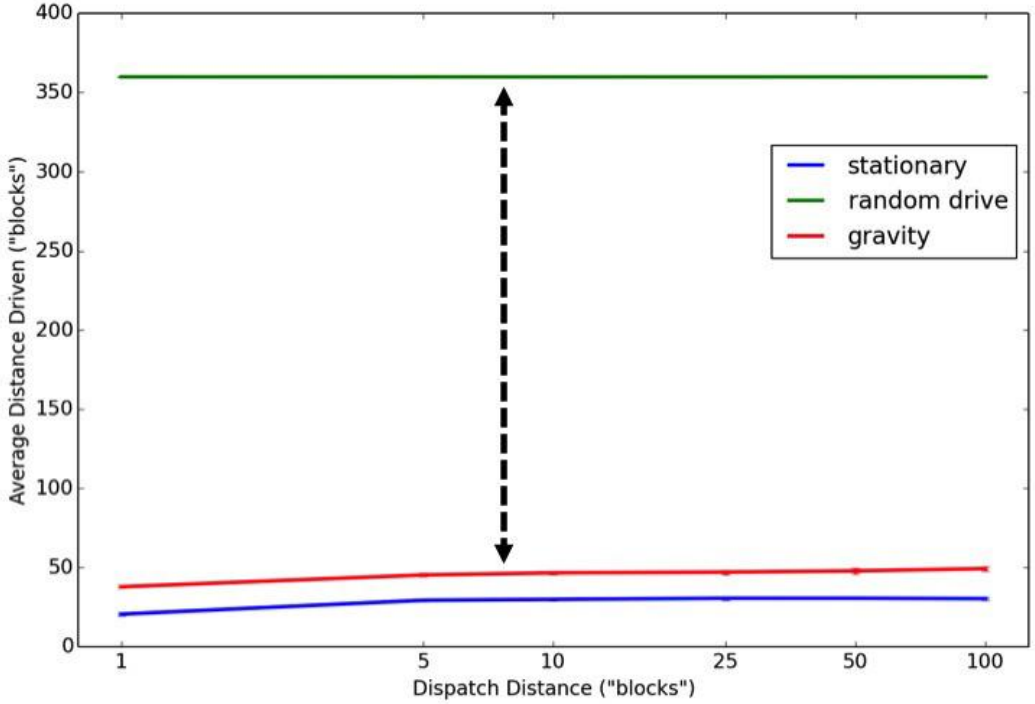
return

end.

Uber's Current Solution

In 2014, Uber did its own simulation to match drivers with riders and evaluated the following strategies:

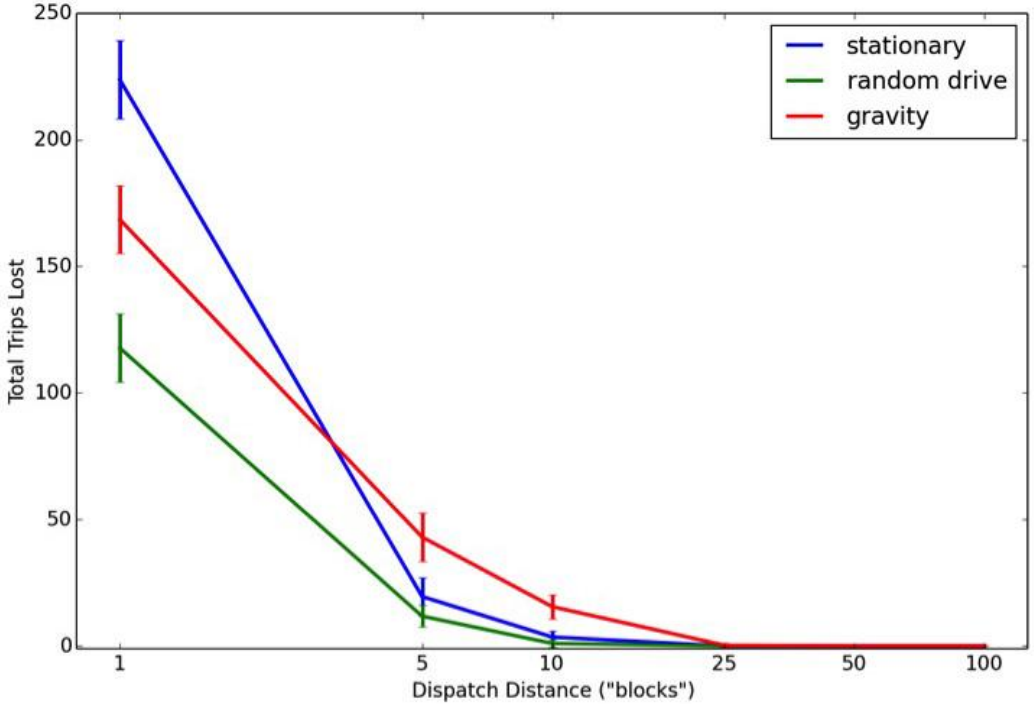
1. Stationary
2. Random Drive
3. Gravity



Uber's Current Solution

In 2014, Uber did its own simulation to match drivers with riders and evaluated the following strategies:

1. Stationary
2. Random Drive
3. Gravity



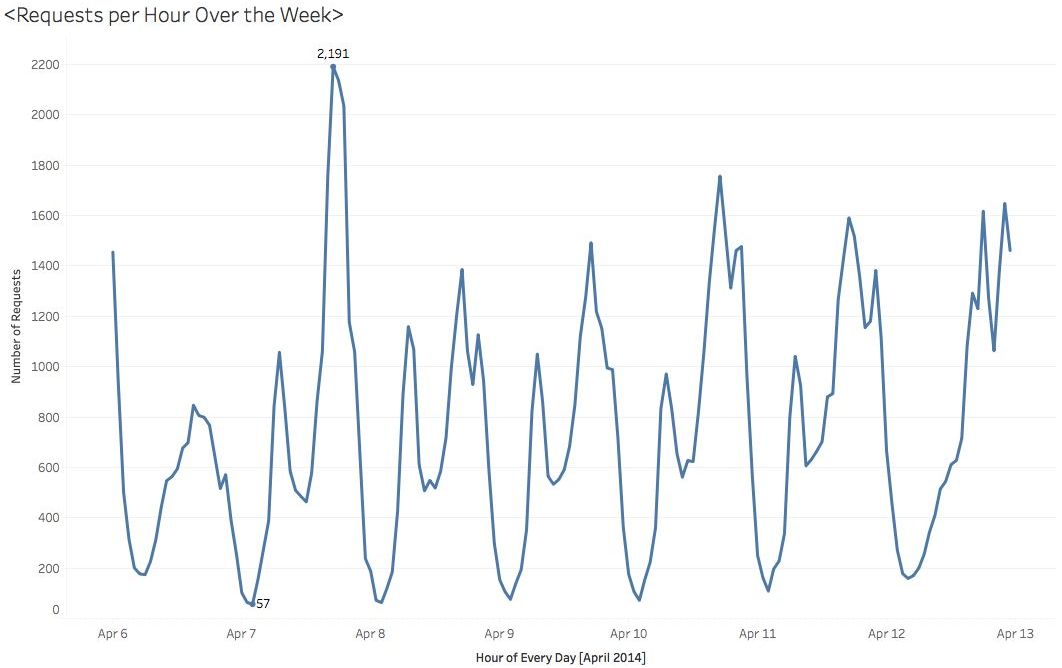
There are *optimal dispatch distances* for pairing a driver with a rider, and there are *optimal behaviors* for drivers to take in between trips. When dispatch distances are very short, drivers should navigate back toward *demand density*.

Our Algorithm

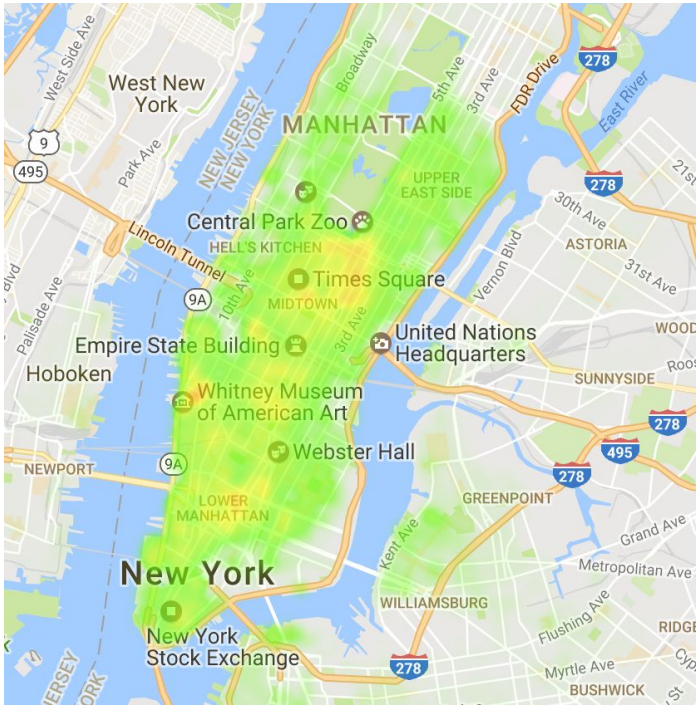
Since the problem is NP-hard, we formulate this as a matching problem and will attempt to apply heuristics.

Using the insights from the 2014 Uber study, we chose to limit the problem to minimizing tardiness when the driver and rider are at a distance of one mile within a high density area.

1. Find the optimal number of drivers to staff at each hour.



2. Determine high demand areas across Manhattan.



3. Formulate the scheduling problem as a matching problem and solve.

- We looked at a particular instance of 15 customers and 10 drivers in a high demand area
  - For each driver i, we calculated their tardiness for their arrival to customer j
  - Calculated by:
    - Generating a random number between 0 and 1 miles, for the distance between the customer and driver
    - Multiplying estimated travel time by a traffic factor of 2

	Driver	1	2	3	4	5	6	7	8	9	10
Customer	1	4.72811	0.77083	3.48739	2.328956	1.876902	4.65169	6.706	1.06989	1.25984	6.11689
	2	6.98597	0.71955	4.367	2.02556	4.92666	2.3762	3.28733	2.30126	6.81453	7.82015
	3	4.719613	2.654881	0.127684	0.557361	2.322281	3.632709	2.992729	3.096482	6.056556	1.783536
	4	2.9567	0.22236	1.256943	0.215963	3.61345	3.602407	7.13415	4.7132	3.87457	0.66398
	5	1.254896	0.31708	0	0.812475	2.235548	0.553639	4.2539	2.32659	2.214785	0.75896
	6	0	3.56521	3.997583	6.027927	1.965863	3.637329	5.110541	5.375085	4.691927	5.26589
	7	2.32548	4.719613	3.23569	4.516823	0.214589	3.441701	5.405519	0	0	5.314732
	8	0.494009	3.25486	0.254896	3.31589	2.33835	0.215369	5.215963	3.777284	1.985632	6.21586
	9	6.281092	6.223061	0.502998	1.025423	3.918776	4.920635	1.86596	0	2.41893	1.898743
	10	2.315332	0.332264	2.2549	0.526983	4.719613	5.236589	6.353748	2.959171	1.610007	0.554639
	11	4.637598	1.25486	4.688676	6.880204	6.982207	0.683816	4.520142	0	0.84596	1.353721
	12	4.864742	0.553639	1.245986	3.75762	2.944241	4.724397	2.25348	4.719613	1.254856	1.223591
	13	3.204046	0.2548	2.859447	3.21569	0.123569	2.344336	4.523868	0	2.359878	5.825452
	14	0	5.653938	2.067595	0.553639	0.232858	4.697663	4.794459	3.446455	0.812563	1.24895
	15	1.696489	2.056759	4.2596	1.25486	5.604655	0.094511	4.674944	1.585506	2.03258	1.45879

This is a greedy algorithm.

Total Tardiness	8.306097									
Driver	1	2	3	4	5	6	7	8	9	10
Customer 1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	1	0	0	0	0

This is a greedy algorithm.