Uber: Minimizing ETA

Anna Juchnicki          Sasha Sklyarenko
Christina Nguyen       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:

\[ P_m \mid r_j, d_j \mid \sum T_j \]
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 $L_{\text{max}}$ 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)

![Diagram of a tree network](image)
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($v_q,i; L_0(v_q,i), W(v_q,i), P(v_q,i), \pi^{DF}(v_q,i))$;

end;

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

$M_0(v_q,\psi(0)) \geq M_0(v_q,\psi(1)) \geq ... \geq M_0(v_q,\psi(k_q))$;
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
This is a greedy algorithm.
This is a greedy algorithm.