Data Council, Barcelona, Oct 2, 2019

Rethinking transportation in cities: Making smarter traffic through Optimization and Location Intelligence

Miguel Alvarez
Data Scientist, CARTO
malvarez@carto.com

CARTO

CARTO is the platform to build powerful Location Intelligence apps with the best data streams available.


TAXI

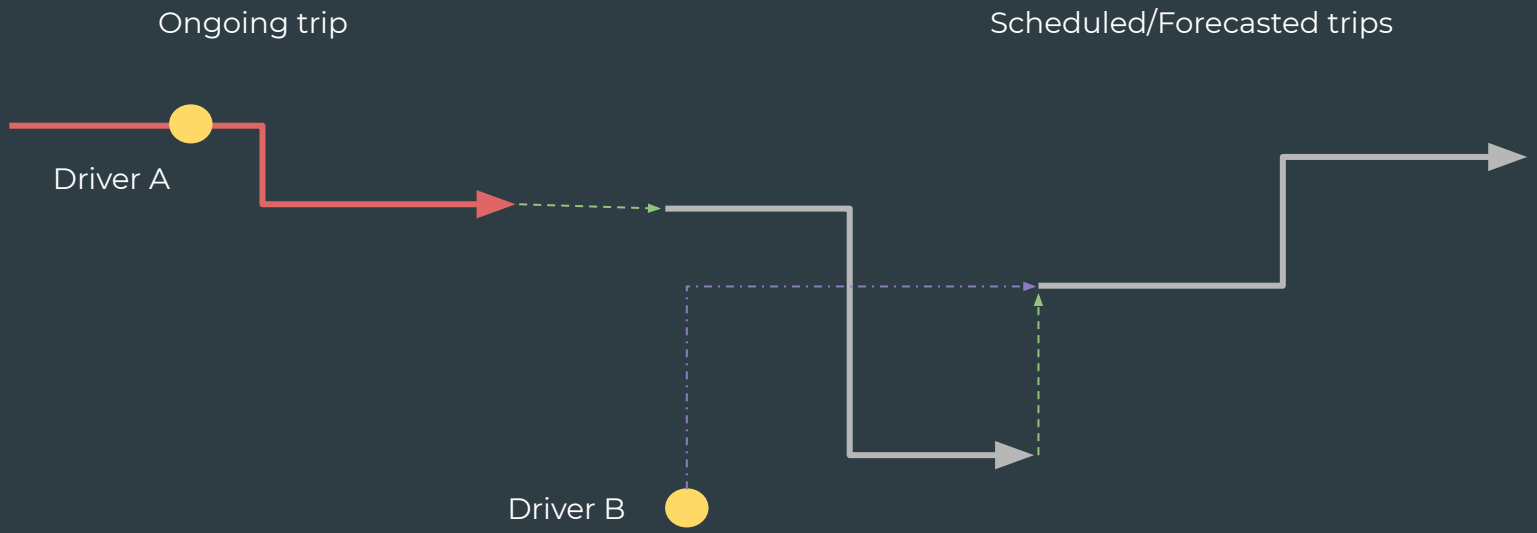

PERFUMERIA

What if we could rethink the way this service is provided to make more livable cities?

New legislation + Optimization + Location Intelligence

Goal: Quick response

Routing problem



Traditional vs on-demand last mile transportation problems

Traditional

All the information about orders and driver availability is known beforehand.

Normally a solution is not needed immediately.

Vehicle Routing Problem (VRP)

Main challenge: Finding a near-optimal solution

On-demand

Narrow vision problem: Much less information available

Orders have to be processed and assigned in (almost) real time.

Classification depends on characteristics of the service and efficiency required: From **Assignment Problem** to VRP

Main challenges:

- Overcome narrow vision problem
- Instant solution required

Visualizing on-demand orders with CARTOFrames

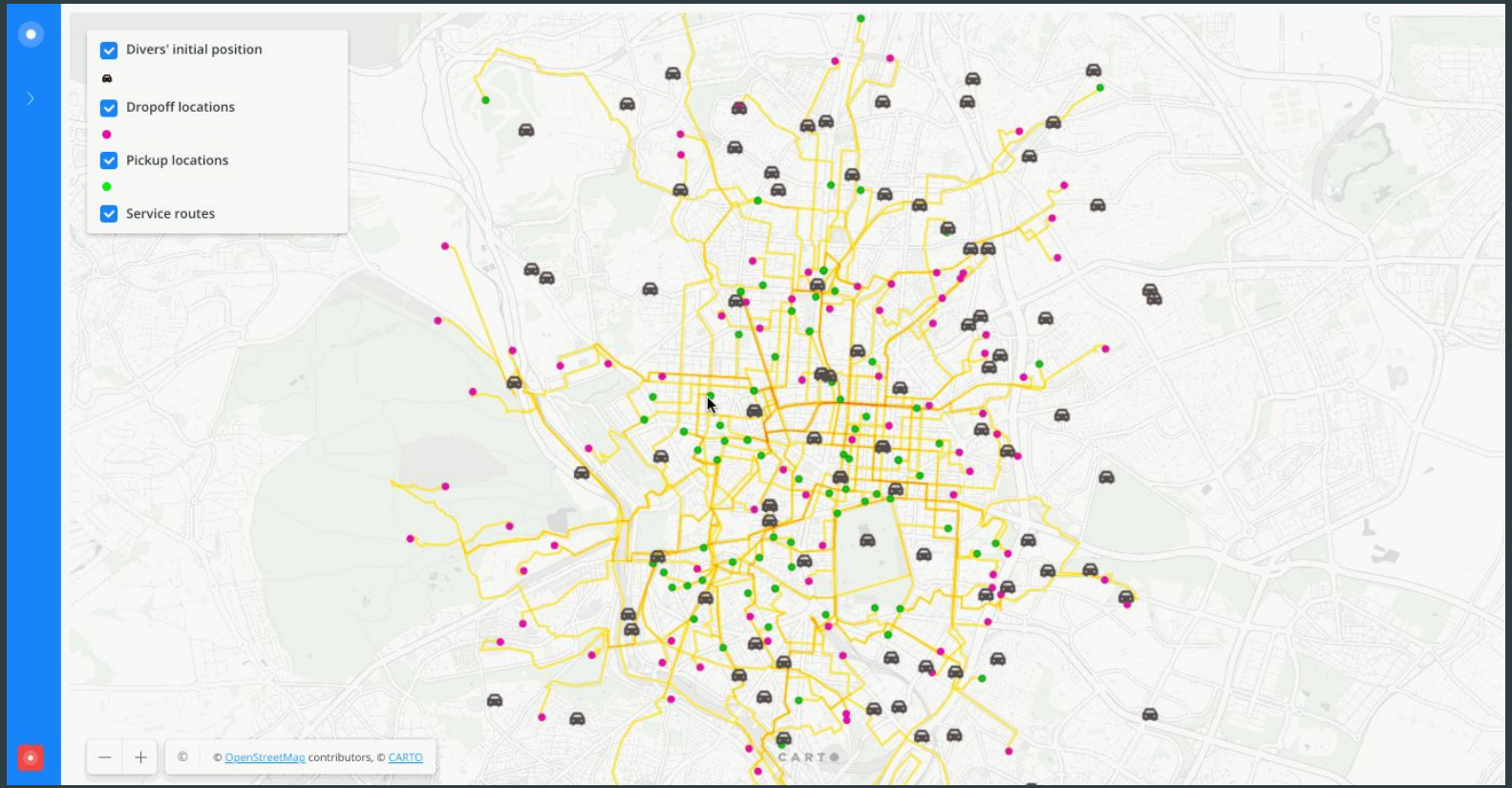
```
Map(
    Layer(
        'services_animated_',
        '''filter: animation($order_place_minute, 60, fade(0.5, 1))
        color:opacity(turquoise, 0.8)''',
    widgets=[
        animation_widget(
            title='Hour',
            description= 'Play, pause, move forward or backwards'
        )]),
    basemap = basemaps.darkmatter
)
```

Designing and improving an optimization algorithm

Assumptions made

- The goal will be to minimize distances, and our main metric will be the accumulated distance traveled by all drivers.
- Every trip is independent of each trip, i.e., they cannot be combined. A driver will not be able to start a trip until they finish the one they are currently doing unless they are idle.
- Drivers will not be assigned to a new trip until they are idle.
- The type of fleet is the same for every driver.
- We will always have enough idle drivers to assign to trips at each iteration of the algorithm.

Visualizing trips received from 7:00 pm to 7:05 pm



Step 1. Greedy algorithm

Usually the first approach followed when solving this problem.

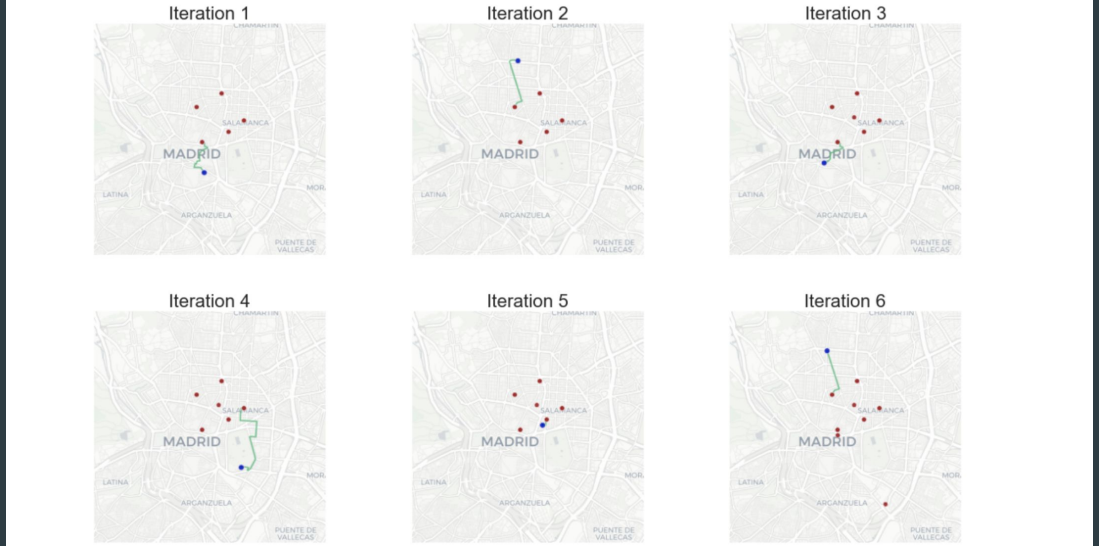
Algorithm activated every time an order is received and it searches the nearest idle driver.

Very easy to implement, and to understand and analyze its results.

Solution: Distance **181.67** km

```
[51]: maps = [Map(layers=[Layer(query_assignment_path.format(3*i+j), "color:turquoise"),
                          Layer(query_avail_riders.format(3*i+j), "color:red"),
                          Layer(query_pickup_point.format(3*i+j), "color:blue")],
                  title='Iteration {}'.format(3*i+j+1),
                  viewport={
                      'zoom': 11,
                      'lat': 40.421597,
                      'lng': -3.695756
                          })
              for i in range(2) for j in range(3)]

      Layout(maps, 3, 2)
```



Step 2. Batch assignment algorithm

Increase information available and flexibility by postponing decisions.

Postpone assignments, running the algorithms every *x* minutes.

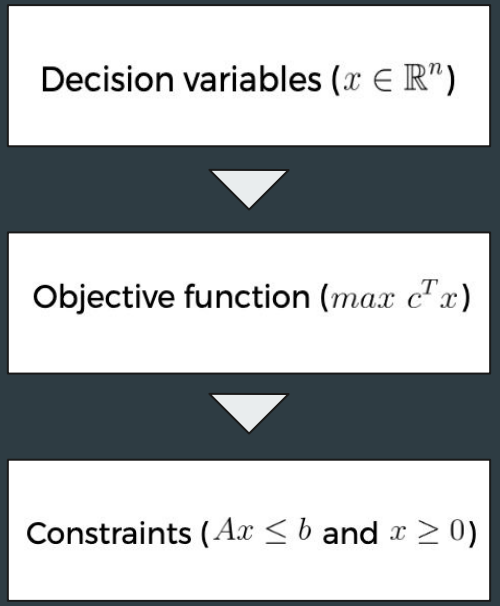
m trips, n drivers ⇒ Assignment problem

- Problem-specific techniques: **Hungarian algorithm**
- General optimization techniques: **Linear Programming**

Linear programming

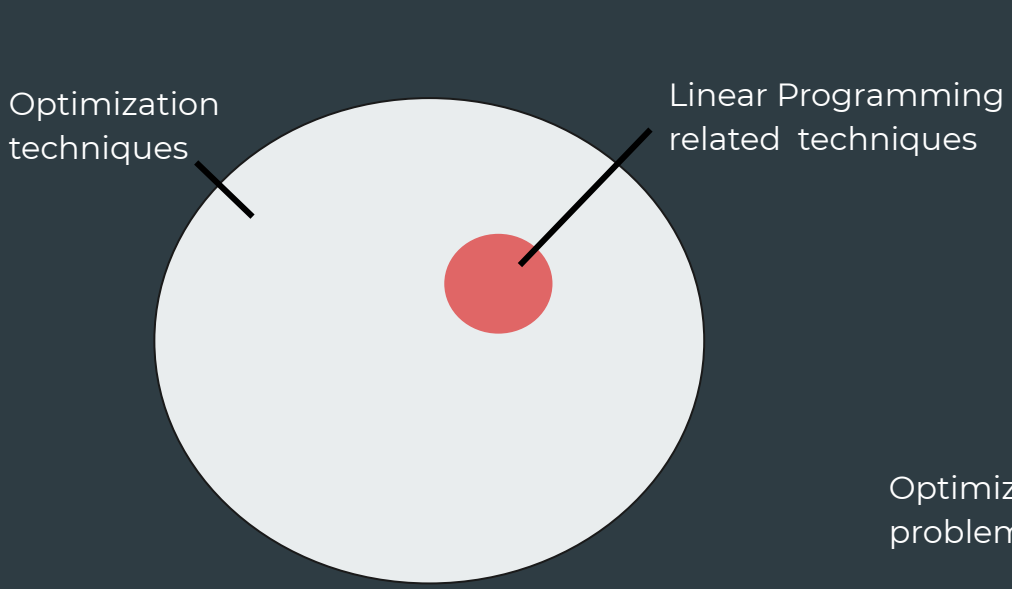
Linear programming (aka linear optimization) is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear (in)equations

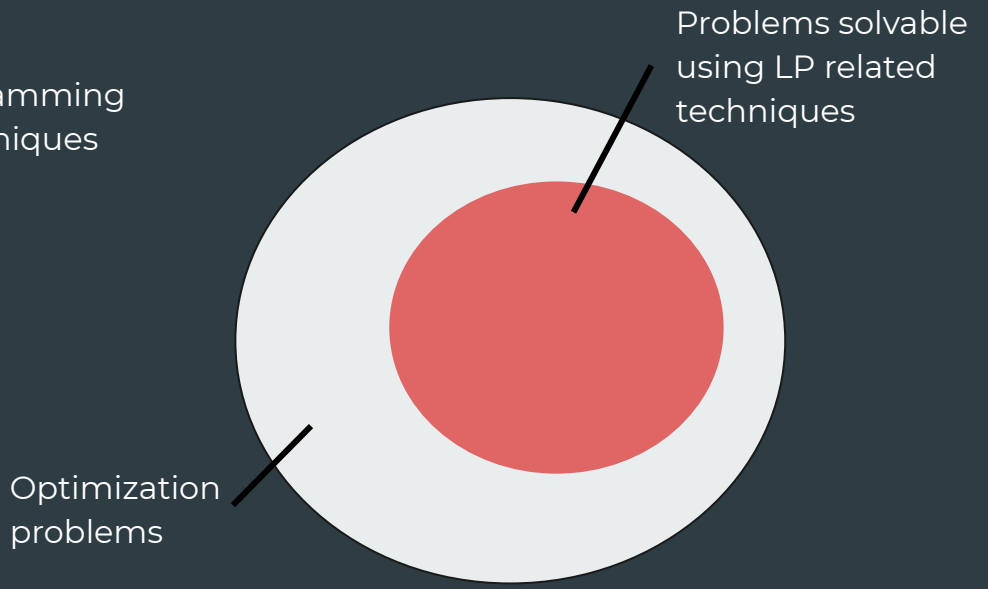
$$max\ c^T x$$
s.t.
$$Ax \leq b$$
$$x \geq 0$$
$$A \in \mathbb{R}^{mxn}, x \in \mathbb{R}^n, c \in \mathbb{R}^n, b \in \mathbb{R}^m$$



Why so powerful?

- Exact modeling. Ensures finding the *optimal* solution.
- Common mathematical language in Optimization





Assignment Problem. Modeling using OR-Tools*

```
[1]: from ortools.linear_solver import pywraplp

     # Instantiate solver. We use CBC solver (COIN-OR project)
     solver = pywraplp.Solver('MySolver', pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)
```

Indices

$i = \{1, 2, ..., I\}$ orders

$j = \{1, 2, ..., J\}$ riders

Parameters

d_{ij} : distance from driver j's starting location to trip i's pickup location

```
[6]: no_servs, no_drivers = np.shape(cost_matrix)
```

* https://developers.google.com/optimization/

Variables

X_{ij} : binary variable indicating whether trip i gets assigned to driver j

[7]:
```
x = [solver.NumVar(0.0, 1.0, 'x_{}_{}'.format(i//no_drivers, i%no_drivers))
     for i in range(no_servs*no_drivers)]
```

Objective function

$$min \sum_{ij} d_{ij} X_{ij}$$

[8]:
```
objective = solver.Objective()
for i in range(no_servs):
    for j in range(no_drivers):
        cost = int(cost_matrix[i, j] * 60)
        objective.SetCoefficient(x[no_drivers*i+j], cost)
objective.SetMinimization()
```

(1) Every driver is assigned to, at most, one trip

$$\sum_i X_{ij} \leq 1, \forall j$$

[9]:
```
for i in range(no_drivers):
    ct1 = solver.Constraint(0.0, 1.0)
    for j in range(no_servs):
        ct1.SetCoefficient(x[no_drivers*j+i], 1)
```

(2) Every trip is assigned to, at most, one driver

$$\sum_j X_{ij} \leq 1, \forall i$$

[10]:
```
for i in range(no_servs):
    ct2 = solver.Constraint(0.0, 1.0)
    for j in range(no_driver):
        ct2.SetCoefficient(x[no_riders*i+j], 1)
```

(3) Assign as many trips as possible

$$\sum_{ij} X_{ij} = min\{I, J\}$$

[11]:
```
min_assignments = float(np.minimum(no_drivers, no_servs))
ct3 = solver.Constraint(min_assignments, min_assignments)
for k in range(no_drivers*no_servs):
    ct3.SetCoefficient(x[k], 1)
```

Solving our problem. The Simplex Algorithm

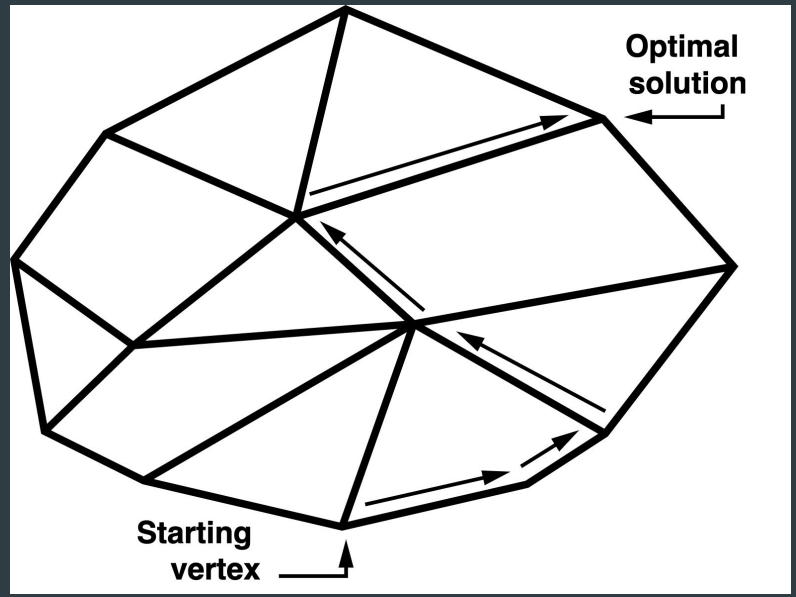
```
[12]: result_status = solver.Solve()
      assert result_status == pywraplp.Solver.OPTIMAL
```

Theorem: If it exists, the optimal solution of a linear program is at an extreme point (vertex) of the polytope defined by the constraints.

$$x \in \mathbb{R}^n$$
$$x = [x_B, x_N];\ \text{with}\ x_B \in \mathbb{R}^m \ \text{and}\ x_N \in \mathbb{R}^{(n-m)}$$

Algorithm:

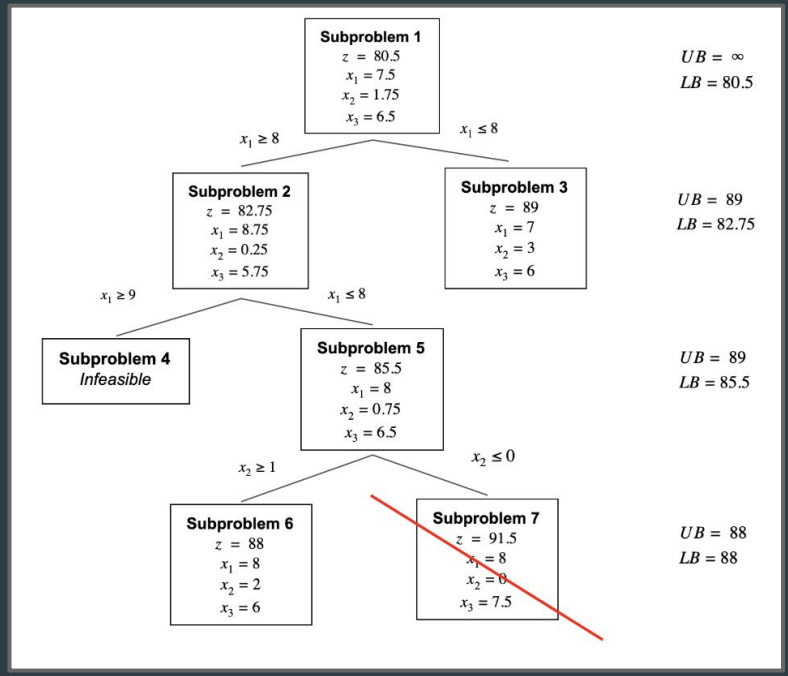
1. Find initial feasible basic solution
2. Repeat until no new entering non-basic variable is found:
   2.1. Find entering non-basic variable
   2.2. Find leaving basic variable



What if my variables are discrete?

Branch and bound algorithm

1. Solve relaxed problem
2. Repeat until no better integer solution can be found:
   2. 1. - If integer solution found: Update best integer bound
      - Else, can we prune this branch?
      - Else, update best solution bound ,and pick one integer defined variable with continuous value and branch
   2. 2. Pick one branch and solve relaxed



What makes a good solver?

1. Presolve
2. Heuristics
3. Parallel communication

Cool property thanks to totally unimodular matrices

$$
\begin{aligned}
& max\ c^T x \\
& \text{s.t.} \\
& Ax \leq b \\
& x \geq 0 \\
& A \in \mathbb{R}^{mxn}, x \in \mathbb{R}^n, c \in \mathbb{R}^n, b \in \mathbb{R}^m
\end{aligned}
$$

If *A* and *b* are integer, then all basic feasible solutions are integer regardless of how we define the variables *x* because the matrix *A* is **totally unimodular** (i.e, every square submatrix has determinant 0, +1 or −1)

We can define our variables as continuous!

Solution

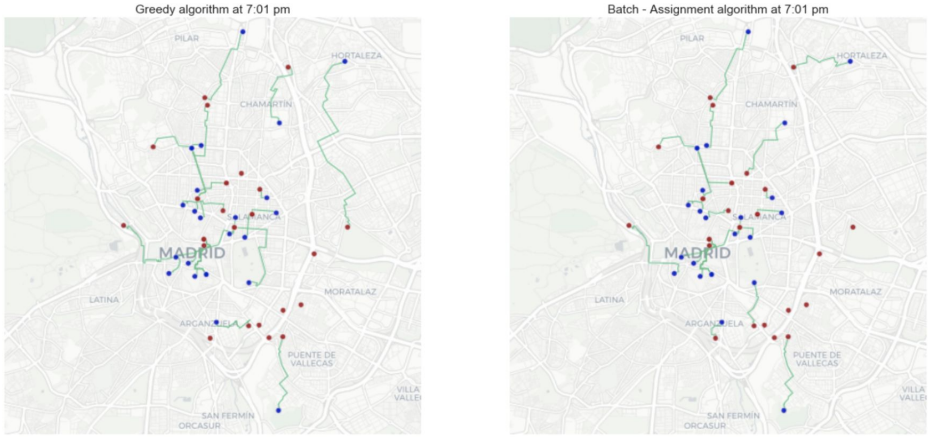
Greedy: 181.67 km

Assignment: 159.10 km

13% improvement!

```
[65]: Layout([
          Map(layers=[Layer(query_assignment_path_m.format(iterations[0], iterations[1]), 'color: turquoise'),
                      Layer(query_avail_riders_m.format(iterations[1], iterations[0]), 'color: red'),
                      Layer(query_pickup_point_m.format(iterations[0], iterations[1]), 'color: blue')],
              title="Greedy algorithm at 7:01 pm",
              viewport={
               'zoom': 11,
               'lat': 40.421597,
               'lng': -3.695756
                      }),
          Map(layers=[Layer(query_assignment_path_batch.format(iterations[0], iterations[1]), 'color: turquoise'),
                      Layer(query_avail_riders_batch.format(iterations[1], iterations[1]), 'color: red'),
                      Layer(query_pickup_point_m.format(iterations[0], iterations[1]), 'color: blue')],
              title="Batch - Assignment algorithm at 7:01 pm",
              viewport={
               'zoom': 11,
               'lat': 40.421597,
               'lng': -3.695756
                     })
      ], 2, 1)
```

```
[65]: Text(0.5, 1.0, 'Batch - Assignment algorithm at 7:01 pm')
```



Step 3. Supply - demand matching

Forecasting demand to make smarter assignments

Build a reference grid + Collect historical data + Enrich your data

↓

Forecast demand

↓

Broaden vision of the problem

Minimize empty driving time

Reduce driving distances / waiting times

Data enrichment. CARTO DATA OBSERVATORY

Financial

Merchant and ATM transaction data from leading banks and credit card companies

Human Mobility

Mobile device and GPS data provide insight into human movement patterns

Demographics

The most recent census data including: age, income, household types and more

Housing

Property statistics, prices, and history to drive decisions in investment portfolios

Road Traffic

Data from routing apps and GPS to analyse traffic patterns and commuter behaviour

Points of Interest

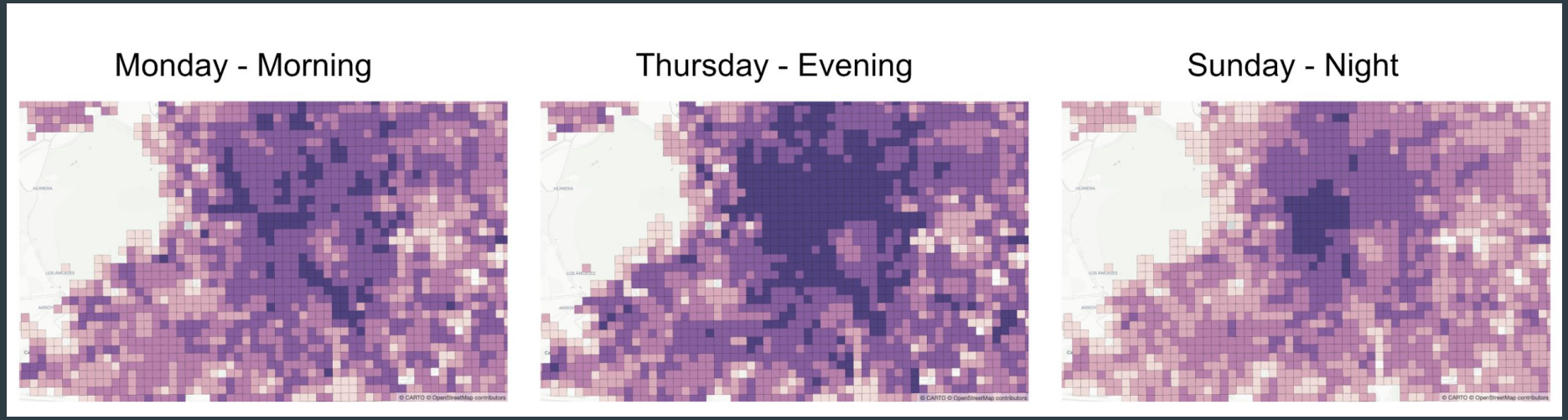
Location data for business establishments, restaurants, schools, attractions, and more

Data enrichment. Footfall and OD matrix to avoid bias

We know the trips we have made, but we don't know what our competitors are doing. We don't have a complete version of the demand.

Footfall

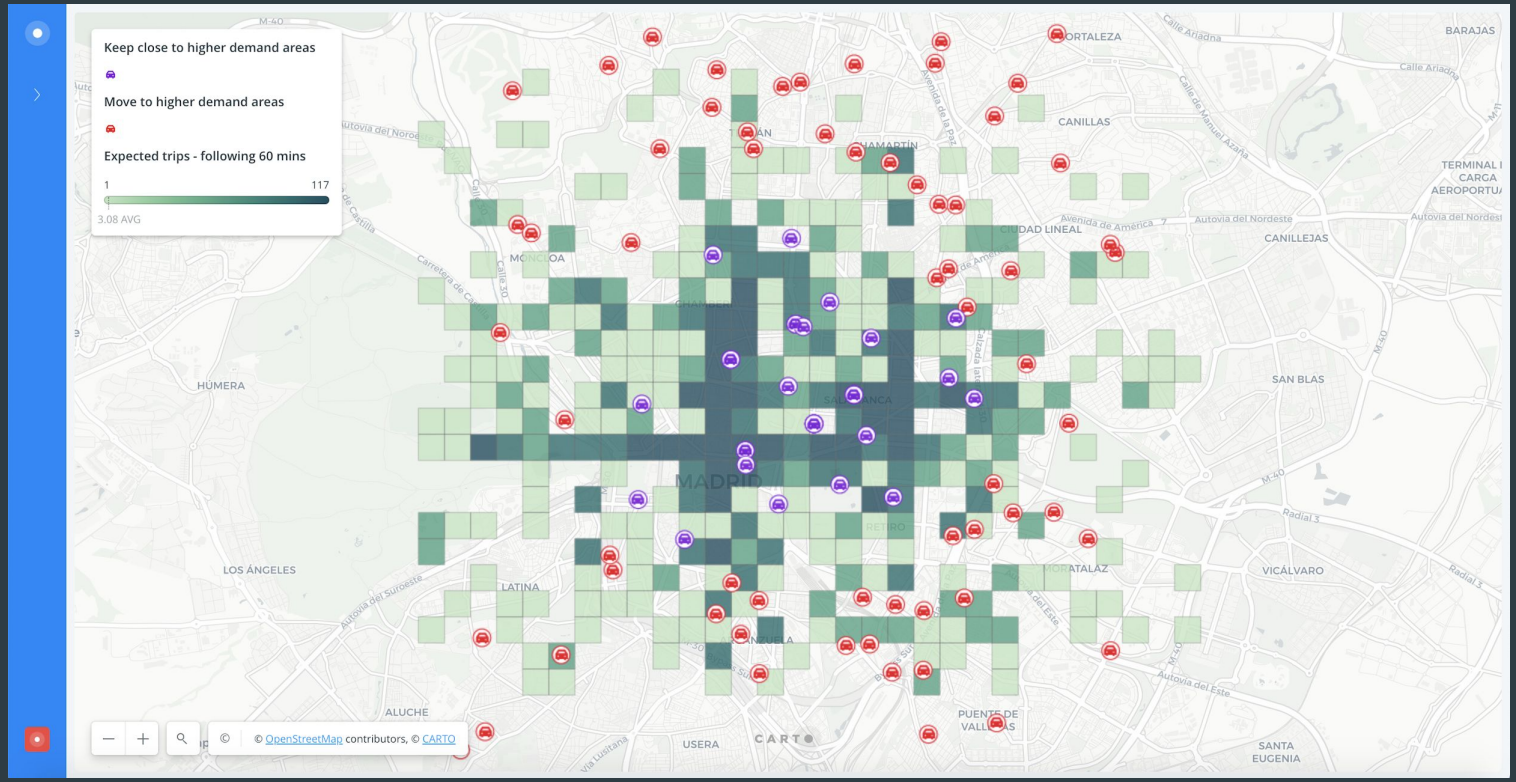
We can know the number of people visiting different parts of the city at different days of the weeks, and different hours of the day with a very high precision (250x250m grid)



OD matrix

We can know where people visiting a specific cell live or work.
This is very powerful information to identify potential customers.

Forecasting result



Logical constraints

Drivers already in higher expected demand zones, can only be assigned to trips if at least 75% of the other drivers are assigned to trips.

Indices

$k = \{k_1, k_2..., k_K\} \subset \{1, 2..., J\}$ drivers to keep close to higher demand areas

$m = \{m_1, m_2, ..., m_M\} \subset \{1, 2..., J\}$ drivers to move to higher demand areas

Variables

δ : binary auxiliary variable for linearization of logical constraint

Constraints

$$(4) \sum_{i,k} X_{ik} > 0 \implies \sum_{i,m} X_{im} \geq 0.75 \cdot I$$

Linearization:

$$\sum_{i,k} X_{ik} > 0 \implies \delta = 1 \implies \sum_{i,m} X_{im} \geq 0.8 \cdot I$$

$$\sum_{i,k} X_{ik} > 0 \implies \delta = 1 :$$

$$\sum_{i,k} X_{ik} \leq K \cdot \delta$$

$$\delta = 1 \implies \sum_{i,m} X_{im} \geq 0.75 \cdot I :$$

$$\sum_{i,m} X_{im} \geq 0.75 \cdot I \cdot \delta$$

```
[17]: # New variables
delta = solver.IntVar(0.0, 1.0, 'delta')
# New constraints
# 4.1
ct41 = solver.Constraint(0.0, 0.0)
ct41 = solver.SetCoefficient(delta, -float(len(keep_drivers_list)))
for i in range(no_servs):
    for k in keep_drivers_list:
        ct41.SetCoefficient(x[no_riders*i+k], 1)
#4.2
ct42 = solver.Constraint(0.0, solver.infinity())
ct42 = solver.SetCoefficient(delta, -0.75*no_servs)
for i in range(no_servs):
    for m in move_drivers_list:
        ct42.SetCoefficient(x[no_riders*i+m], 1)
```

Next steps

Apart from this, there are other improvements that would lead to more efficient and higher quality assignments. Some examples of this are:

- Optimization criteria. In our example we took distance as the metric to be optimized. However, we might add extra criteria, always bearing in mind that costs have to be calculated at every iteration of the algorithm. Some examples could be:
  - ETA
  - Utilization: Minimum fleet
  - Priority to urgent trips
- Fair distribution of trips to drivers
- Combining trips (e.g. Uber pool)
- Consider drivers for future assignments before they finish their current trip

Takeaways

Linear Programming is a traditional Optimization technique widely used because of its strength. In order to make the most of it, it is very important to understand how it works and what the different solvers have to offer.

Visualization is essential to easily analyze spatial patterns and the performance of our algorithms.

Data enrichment helps avoid bias of using only our own data.

Thanks for listening!
Any questions?

Miguel Alvarez
malvarez@carto.com

CARTO