THE BEST STREET ART IN PALERMO TOUR

Location-Based Assignment

Minerva University

CS164 - Optimization Methods

Prof. Levitt

November 19, 2022

Table of Contents

THE BEST STREET ART IN PALERMO TOUR

Eight Must-Sees and Why [#audience]

Staying in Buenos Aires, our team enjoys leaving in the neighbourhood that is famous for more than 100 murals worth visiting. In this tour, we will show our 7 favourite graffiti and a perfect place to have breakfast with friends. The tour is perfect for the first-day city exploration, because it will introduce you to the city context and will not take you too long (when you follow our directions), so you have enough time to get over jet lag!

The start point is a residence hall (Gorriti 6066, location 0). Then, you go to the right and walk several blocks to see the first art on Avenida Córdoba 5496 – all coordinates can be found below. This is a creative mayo advertisement, and Buenos Aires is the only city in the world where you find brands paying artists to do graffiti for their marking campaigns (Figure 1). Since you already cannot stop thinking about the food, it is a good time to grab the sandwich in a really popular bagel place – location 2, Uriarte 1376 (Figure 2).

Figure 1. *Creative mayo advertisement.*

Figure 2. *Bagel and desserts cafe.*

After some great food, it is time to remember social responsibility and visit pieces of art created specifically for the local ecological project (Figure 3, Figure 4).

Figure 3. *Jungle fantasy.*

Figure 4. *Nature-mother.*

The following art is a local Minerva joke that we will not disclose now, but the guy on the graffiti reminds someone (hint: Minerva's founder) (Figure 5).

Figure 5. *Ben Nelson art.*

You did not even notice, but already made a circle and are close to home, so just two last pieces of graffiti to remind you how creative and talented people are here (Figure 6, Figure 7).

Figure 6. *Just see it. It's 3D!*

Figure 7. *Caricature of garage's owner*

Mean Travel Time

To calculate the distance between two locations, we use the trip route planner on Google Maps to calculate the walking distance (in meters) between both locations. Since this is a walking tour, factors such as one-way roads do not come into play here; the distance matrix is symmetric. Calculating Mean Distances can be found in Appendix A.

$$C = \begin{bmatrix} 0 & 216 & 248 & 350 & 702 & 1117 & 1680 & 1230 & 1254 \\ 216 & 0 & 31 & 134 & 486 & 901 & 1464 & 1015 & 1040 \\ 248 & 31 & 0 & 102 & 455 & 870 & 1433 & 984 & 1009 \\ 350 & 134 & 102 & 0 & 352 & 767 & 1330 & 881 & 907 \\ 702 & 486 & 455 & 352 & 0 & 416 & 979 & 530 & 565 \\ 1117 & 901 & 870 & 767 & 416 & 0 & 596 & 658 & 624 \\ 1680 & 1464 & 1433 & 1330 & 979 & 596 & 0 & 534 & 500 \\ 1230 & 1015 & 984 & 881 & 530 & 658 & 534 & 0 & 35 \\ 1254 & 1040 & 1009 & 907 & 565 & 624 & 500 & 35 & 0 \end{bmatrix}$$

TSP formulation

An interactive, runnable version of the math described here can be found at this Colab link. For this problem, we can use an indicator variable x where $x_{ij} \in \{0, 1\}$, which equals one whenever a tour goes from point i to point j. We can then look at the cost as being a separate variable c_{ij} which is equivalent to the symmetric distance between the two points, which would mean that the total cost of any tour would be equal to the sum across all costs for every individual tour that was made which in this case, gives us our objective function as being:

$$minimize \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

To look at the constraint of visiting each city only once, we can show that by ensuring that we enter and leave each city only once with the constraints:

$\sum_{\substack{1 \le i \le n \\ i \neq j}} x_{ij} = 1$ where we're looking at point i going out to all j outbound points and $\sum_{\substack{1 \le i \le n \\ i \neq j}} x_{ij} = 1$ where we're looking at all points i coming inbound to j.

However, suppose we just do the optimization with these constraints. In that case, we'll get the answer [(0, 1, 2), (3, 4), (5, 6), (8, 7)] where each sub-list in the list describes a sub-tour as the algorithm optimizes for just the constraints that each point has to be visited once and not that the path has to be connected all the way through.

The easiest solution is manually adding constraints within each sub-tour to ensure you're visiting a place outside the sub-tour. This constraint can be added algorithmically by making sure that the first sub-tour has at least one path that branches out of it with the equation:

$\sum_{i \in S} \sum_{j \notin S} x_{ij} >= 1$ where we're looking across the sum of all the possible combinations of elements in the Sub-tour with the parts not in the sub-tour to be greater than or equal to 1,

implying that there should be at least one connection leading outside. Solving the TSP with CVXPY can be found in Appendix B.

Optimal Solution

The process above increments until the termination condition, which is that there's only a single sub-tour (i.e. a full tour) left, which in this case turned out to be [(0, 7, 8, 6, 5, 4, 3, 2, 1)] which was pretty interesting since it tells us to go in the other direction than what we did in reality which should technically be the same with the only inefficiency being that we had to backtrack on our route to get to Sheikob's bagels at our 8th place which it accounts for by going to 7 first and then 8. Our initial route can be found on the following Google map (Figure 9).

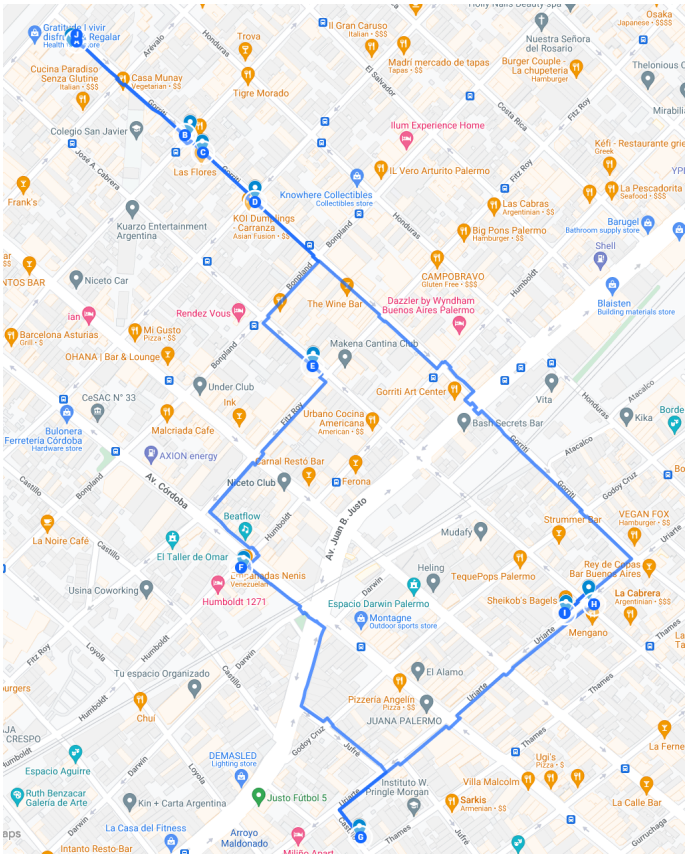

Figure 9. *Our initial path that was not optimal - surprise!*

Contribution

Kseniia: Kseniia helped design the original tour (that we actually undertook) and the written description of the tour, data collection processes and report writing.

Saad: Saad wrote the CVXPY code needed to solve the problem, including the lazy sub-tour elimination. He also wrote the formulation of this problem as a Travelling Salesman Problem.

Irhum: Irhum performed the estimation with Google Maps to produce the distance matrix and cleaned up the code for the neat formatting of appendixes.

Word count: 1117.

A Calculating Mean Distances

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cvxpy as cp

n = 9
C = np.array(
    [
        [
            0,
            216,
            248,
            350,
            474 + 123 + 105,
            605 + 403 + 92 + 17,
            605 + 403 + 233 + 140 + 155 + 89 + 55,
            721 + 120 + 389,
            1120 + 134,
        ],
        [
            216,
            0,
            31,
            134,
            258 + 123 + 105,
            389 + 403 + 92 + 17,
            389 + 403 + 233 + 140 + 155 + 89 + 55,
            389 + 121 + 505,
            906 + 134,
        ],
        [
            248,
            31,
            0,
            102,
            227 + 123 + 105,
            358 + 403 + 92 + 17,
            358 + 403 + 233 + 140 + 155 + 89 + 55,
            358 + 121 + 505,
            875 + 134,
        ],
        [
            350,
            134,
            102,
            0,
            124 + 123 + 105,
            255 + 403 + 92 + 17,
            255 + 403 + 233 + 140 + 155 + 89 + 55,
            255 + 121 + 505,
            773 + 134,
        ],
        [
            474 + 123 + 105,
            258 + 123 + 105,
            227 + 123 + 105,
            124 + 123 + 105,
```

```
            0,
            55 + 479,
            55 + 445,
        ],
        [
            721 + 120 + 389,
            389 + 121 + 505,
            358 + 121 + 505,
            255 + 121 + 505,
            530,
            17 + 377 + 264,
            55 + 479,
            0,
            35,
        ],
        [
            1120 + 134,
            906 + 134,
            875 + 134,
            773 + 134,
            530 + 35,
            17 + 377 + 230,
            55 + 445,
            35,
            0,
        ],
    ]
)
```

B Solving the TSP with CVXPY

```
X = cp.Variable(C.shape, boolean=True)
ones = np.ones((n, 1))

# Defining the objective function
objective = cp.Minimize(cp.sum(cp.multiply(C, X)))

# Defining the constraints
constraints = []
constraints += [X @ ones == ones]
constraints += [X.T @ ones == ones]
constraints += [cp.diag(X) == 0]

```

```python

def lazy_constraint_addition(constraints, X, frm, to):
    every = []
    for i in frm:
        for j in to:
            every.append(X[i] @ [1 if k == j else 0 for k in range(9)])
    constraints += [sum(every) >= 1]

    return constraints


# Solving the problem
def solve(objective, constraints, X, n):
    prob = cp.Problem(objective, constraints)
    prob.solve(verbose=False)
    ans = X.value
    paths = []
    for i in range(n):
        flag = False
        for path in paths:
            if i in path:
                flag = True
        if flag:
            continue

        cur_path = [i]
        next = list(ans[i]).index(1)
        while next not in cur_path:
            cur_path.append(next)
            next = list(ans[next]).index(1)
        paths.append(cur_path)

    return paths


solution = solve(objective, constraints, X, n)

while len(solution) > 1:
    cur_tours = solution
    cur_tours = [list(i) for i in cur_tours]

    cur = cur_tours[0]
    not_cur = [i for i in range(9) if i not in cur]

    constraints = lazy_constraint_addition(constraints, X, cur, not_cur)

    solution = solve(objective, constraints, X, n)
    print(solution)
```

```
# [[0, 1], [2, 3], [4, 5], [6, 7, 8]]
# [[0, 3, 2, 1], [4, 5], [6, 7, 8]]
# [[0, 4, 3, 2, 1], [5, 6], [7, 8]]
# [[0, 5, 4, 3, 2, 1], [6, 7, 8]]
# [[0, 6, 5, 4, 3, 2, 1], [7, 8]]
# [[0, 7, 8, 6, 5, 4, 3, 2, 1]]
```

C Photos!

In keeping up with the spirit of the times, here's BeReal-style front *and* back camera photos of the team at all 8 locations: