

CS164 LBA

Gordon Ma, Gaurish Katlana, Leo Ware

November 20, 2021

1 Destinations [#audience]

We chose eight of our favorite coffee houses in the Berlin. We chose these because, as Minervans, we spend a lot of time working in different cafes throughout the city, but it can sometimes be hard to find cafe's with good coffee and reliable wifi. So, we thought we could do a service to future generations of Minervans in Berlin by preparing this list and finding the optimal path to visit all of them. We call this the cafe tour.

Berlin is one of the centers in the world's third wave coffee culture. As such, there are many cafes in Berlin that is known for its specialty coffee. The following 8 sites are some of Gordon's favorite coffee spots in Berlin, in addition to classical Minervan spots. Bonanza Coffee Roasters removed wifi from the site after a previous class occupied the cafe. Milch and Zucker and The Visit has become Minervan's new swarming spots near the residence hall.

1. THE BARN Café Neukölln
2. Milch and Zucker
3. Bonanza Coffee Roasters
4. Five Elephants Kreuzberg
5. The Visit Cafe
6. Father Carpenter
7. Hallesches Haus
8. Michelberger Hotel

2 Travel Times [#evidencebased]

To fully formulate the traveling salesman problem, we needed data on the time required to bike between each pair of cafes. Due to the recent surge in COVID cases, we decided the safest way to collect this data was using Google maps. Many Minervans own bikes in Berlin. So, we thought bike times were probably most relevant to future classes.

Fig. 1 displays a graph of the cafes along with the estimated biking time between each pair.

Figure 1. *Cafes with estimated biking times in minutes*



$$x_{ij} \in \{0, 1\}$$

We need to constrain the values of the function to ensure that every node is visited at least once. We can do this by adding these constraints:

$$\sum_{i=0}^n x_{ij} = 1$$

$$\sum_{j=0}^n x_{ij} = 1$$

These say that the route must enter and leave each node exactly once.

This ensures that the route visits every node exactly once, but it allows for the possibility of subtours. To prevent this, we use black magic. We introduce a new set of integer valued design variables u_i for $i \in \{1..n\}$. Then, we add the following two constraints:

$$u_i > 0$$

And, for all i, j where $i \neq j$:

$$u_i - u_j + nx_{ij} \leq n - 1$$

This constraint ensures that there are no subtours in the final answer.

3.2 Objective

We want to find an expression for the cost of a route described by the design variables x_{ij} . Let c_{ij} represent the cost of traveling from node i to node j in the network. Then, the cost of the route will be:

$$cost = \sum_{i,j \in \{1..n\}} x_{ij} c_{ij}$$

This expression looks at every edge which we could travel and asks if we travel it as part of the route. It adds up the costs of all the routes we take.

4 Optimal Solution [#optimization]

5 Alternative Formulations & Comparison

In order to check the accuracy of our solution, we compare this result to an analysis of the problem using two other optimization methods: ant colony optimization, and brute force search. Since we are solving the traveling salesman for a case with only 8 cafes, the problem is actually small enough to brute force. There are $8! = 40,302$ distinct paths through all eight cafes, starting and ending at the res hall. So, we can simply iterate over these and see which is cheapest.

5.1 Appendix: Code

```
[21]: import numpy as np
import networkx as nx
import cvxpy as cp
import matplotlib.pyplot as plt
import random
```

```
[16]: cafes = ['A&O Hostels', 'Bonanza Coffee Roasters', 'Milch and Zucker_
↳Oranienstraße', 'The Visit Coffee Roasters', 'Five Elephants Kreuzberg',_
↳'The Barn Cafe Neukölln', 'Michelberger Hotel', 'Father Carpenter',_
↳'Hallesches Haus']
```

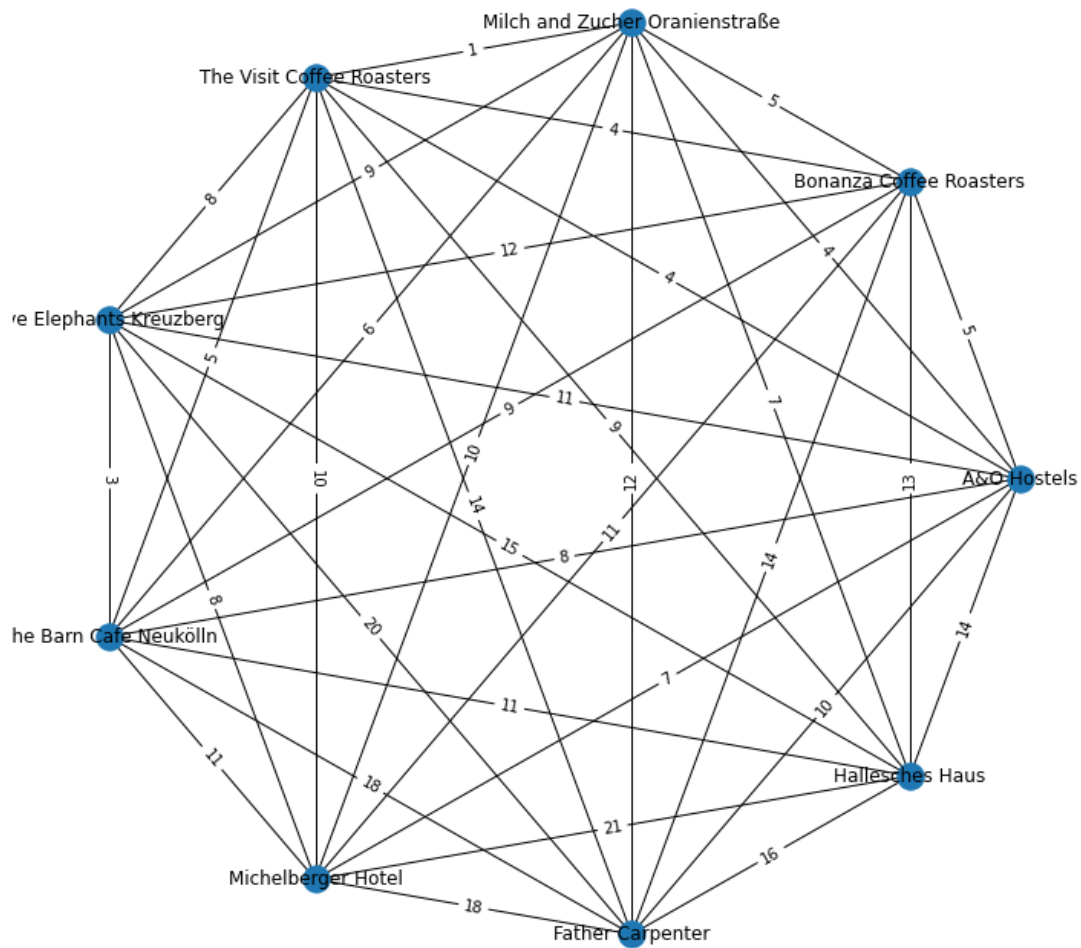
```
cost = np.array([
    [ 0,  4,  4,  4, 11,  9, 11, 11, 13],
    [ 5,  0,  4,  4, 12, 10, 15, 16, 14],
    [ 4,  5,  0,  1,  8,  6, 14, 13, 10],
    [ 4,  4,  1,  0,  8,  5, 13, 14,  9],
    [11, 12,  9,  8,  0,  3, 10, 21, 15],
    [ 8,  9,  6,  5,  3,  0, 13, 20, 12],
    [ 7, 11, 10, 10,  8, 11,  0, 15, 18],
    [10, 14, 12, 14, 20, 18, 18,  0, 18],
    [14, 13,  7,  9, 15, 11, 21, 16,  0]
])
```

```
[17]: g = nx.from_numpy_array(cost)
g = nx.relabel_nodes(g, dict(enumerate(cafes)))

plt.figure(figsize=(10,10))
pos = nx.circular_layout(g)
nx.draw(g, pos=pos, with_labels=True)
nx.draw_networkx_edge_labels(g, pos=pos, edge_labels=nx.get_edge_attributes(g,_
↳'weight'))

plt.title("Biking Times between Cafes")
plt.savefig("imgs/graph.png")
```

Biking Times between Cafes



```
[18]: def integer_programming(names, cost):
        """Solves the traveling salesman problem using CVXPY and integer_
        programming."""

        n = cost.shape[0]

        # route matrix
        X = cp.Variable(cost.shape, boolean = True)

        # surrogate variables
        u = cp.Variable(n, integer = True)
```

```

# objective function
objective = cp.Minimize(cp.sum(cp.multiply(cost, X)))

# define constraints
ones = np.ones(n)
constraints = [
    # each node must be visited exactly once
    X @ ones == ones,
    X.T @ ones == ones,

    # no self-loops
    cp.diag(X) == 0,

    # The first variable is the starting point (A&O Hostels)
    u[0] == 1,
    u[1:] >= 2,
    u[1:] <= n,
]

# subtour elimination constraints
for i in range(1,n):
    for j in range(1,n):
        if i != j:
            constraints.append(u[i] - u[j] + n*X[i,j] <= n-1)

# Define and solve the CVXPY problem.
prob = cp.Problem(objective,
                   constraints)
result = prob.solve()

# convert the result to a list of visited nodes
one_steps = []
for i in range(0,9):
    for j in range(0,9):
        if solution.value[i,j] == 1:
            one_steps.append([i,j])

# turn matrix into list of cafes
pos = 0
order = [names[0]]
while True:
    for edge in one_steps:
        if edge[0] == pos:
            pos = edge[1]
            break
    order.append(names[pos])

```

```
        if pos == 0:
            break

    return order

integer_programming(cafes, cost)
```

```
[18]: ['A&O Hostels',
       'Bonanza Coffee Roasters',
       'Milch and Zucker Oranienstraße',
       'The Visit Coffee Roasters',
       'Hallesches Haus',
       'The Barn Cafe Neukölln',
       'Five Elephants Kreuzberg',
       'Michelberger Hotel',
       'Father Carpenter',
       'A&O Hostels']
```

Long-step dual simplex will be used

```
[38]:
```

7

```
[ ]:
```