CS164 LBA

Gordon Ma, Gaurish Katlana, Leo Ware

November 20, 2021

1 Destinations [#audience]

We chose eight of our favorite coffee houese 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 coffeee spots in Berlin, in additional to classical Minervan spots. Bonanza Coffee Roasters removed wifi from the site after a previous class occupied the cafe. Milch and Zucher and The Visit has became Minervan's new swarming spots near the residence hall.

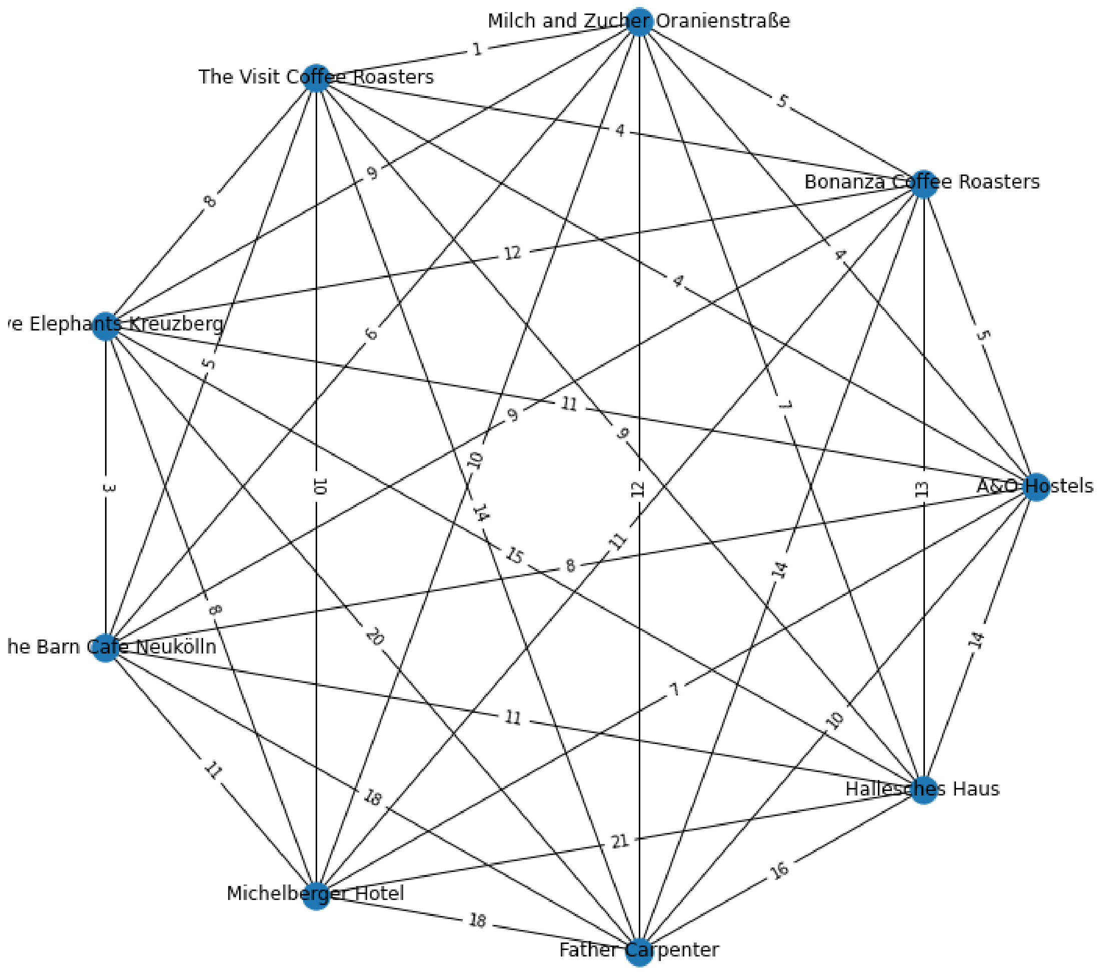
1. THE BARN Café Neukölln
2. Milch and Zucher
3. Bonanza Coffee Roasters
4. Five Elephants Kreuzberg
5. The Visit Cafe
6. Father Carpenter
7. Hallesches Haus
8. Michelberger Hotel

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*



3 TSP Formulation [#optimizationstrategy, #constrained, #algorithms]

Our goal with this assignment was the find the route from the residence hall through all 8 cafes that takes the minimum time to bike. To do this, we think about the problem as a graph traversal problem—the problem of finding the shortest path through the graph of cafes which visits every cafe. Then, we can formulate this problem as an integer programming problem with a number of constraints.

3.1 Constraints

To formulate this project as an integer programming problem, we need to describe possible routes in terms of integer valued variables. We define a design variables x_ij for each pairs of nodes i and j. This design variable takes on the value 1 if the route goes directly from i to i, and it is zero otherwise. We add the constraint:

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

We need to constrain the values of the funtion to ensure that every node it visited at least once. We can do this by adding these contraints:

$$\sum_{i=0}^{n} x_i j = 1$$

$$\sum_{j=0}^{n} x_i j = 1$$

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 constraints ensures that no there are no subtours in the final answer.

3.2 Objective

We want to find an expression for the cost of a route desribed 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 anlysis 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 enoough 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.

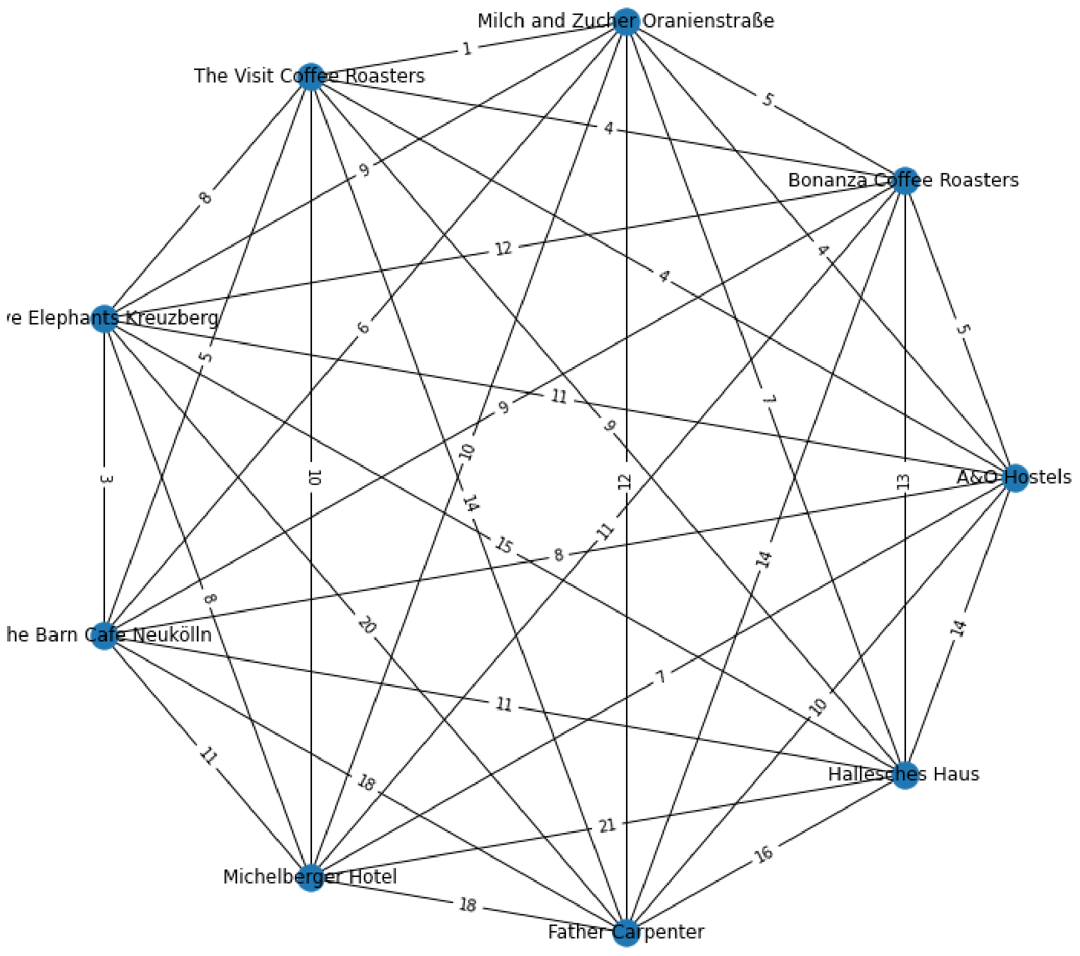
5.1 Appendix: Code

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

```
[2]: cafes = ['A&O Hostels', 'Bonanza Coffee Roasters', 'Milch and Zucher␣
      ↪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]
     ])
```

```
[3]: 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



```
[4]: 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)

         # surrograte 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)
```

```
---------------------------------------------------------------------------
SolverError                               Traceback (most recent call last)
/var/folders/7_/t6r6q2tn0mv_rnygf2k3bbtm0000gn/T/ipykernel_32009/215443331.py i
 ↪<module>
     62     return order
     63
---> 64 integer_programming(cafes, cost)

/var/folders/7_/t6r6q2tn0mv_rnygf2k3bbtm0000gn/T/ipykernel_32009/215443331.py i
 ↪integer_programming(names, cost)
     38     prob = cp.Problem(objective,
     39                       constraints)
---> 40     result = prob.solve()
     41
     42     # convert the result to a list of visited nodes

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
 ↪cvxpy/problems/problem.py in solve(self, *args, **kwargs)
    470         else:
    471             solve_func = Problem._solve
--> 472         return solve_func(self, *args, **kwargs)
    473
    474     @classmethod

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
 ↪cvxpy/problems/problem.py in _solve(self, solver, warm_start, verbose, gp,␣
 ↪qcp, requires_grad, enforce_dpp, **kwargs)
    963                 return self.value
    964
--> 965         data, solving_chain, inverse_data = self.get_problem_data(

    966             solver, gp, enforce_dpp, verbose)
    967

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
 ↪cvxpy/problems/problem.py in get_problem_data(self, solver, gp, enforce_dpp,␣
 ↪verbose)
    578         if key != self._cache.key:
    579             self._cache.invalidate()
--> 580             solving_chain = self._construct_chain(

    581                 solver=solver, gp=gp, enforce_dpp=enforce_dpp)
    582             self._cache.key = key

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
 ↪cvxpy/problems/problem.py in _construct_chain(self, solver, gp, enforce_dpp)
    804         A solving chain
    805         """
--> 806         candidate_solvers = self._find_candidate_solvers(solver=solver,␣
 ↪gp=gp)
    807         self._sort_candidate_solvers(candidate_solvers)
    808         return construct_solving_chain(self, candidate_solvers, gp=gp,

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
 ↪cvxpy/problems/problem.py in _find_candidate_solvers(self, solver, gp)
    727                     in incorrect solutions and is not recommended.
    728                 """
--> 729                 raise error.SolverError(msg)
    730             candidates['qp_solvers'] = [
    731                 s for s in candidates['qp_solvers']

SolverError:

                    You need a mixed-integer solver for this model. Refer to th␣
 ↪documentation
                        https://www.cvxpy.org/tutorial/advanced/index.
 ↪html#mixed-integer-programs
                    for discussion on this topic.

                    Quick fix 1: if you install the python package CVXOPT (pip␣
 ↪install cvxopt),
                    then CVXPY can use the open-source mixed-integer solver␣
 ↪`GLPK`.

                    Quick fix 2: you can explicitly specify solver='ECOS_BB'.␣
 ↪This may result
                    in incorrect solutions and is not recommended.
```

```
[ ]:
```

```
[ ]:
```