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.

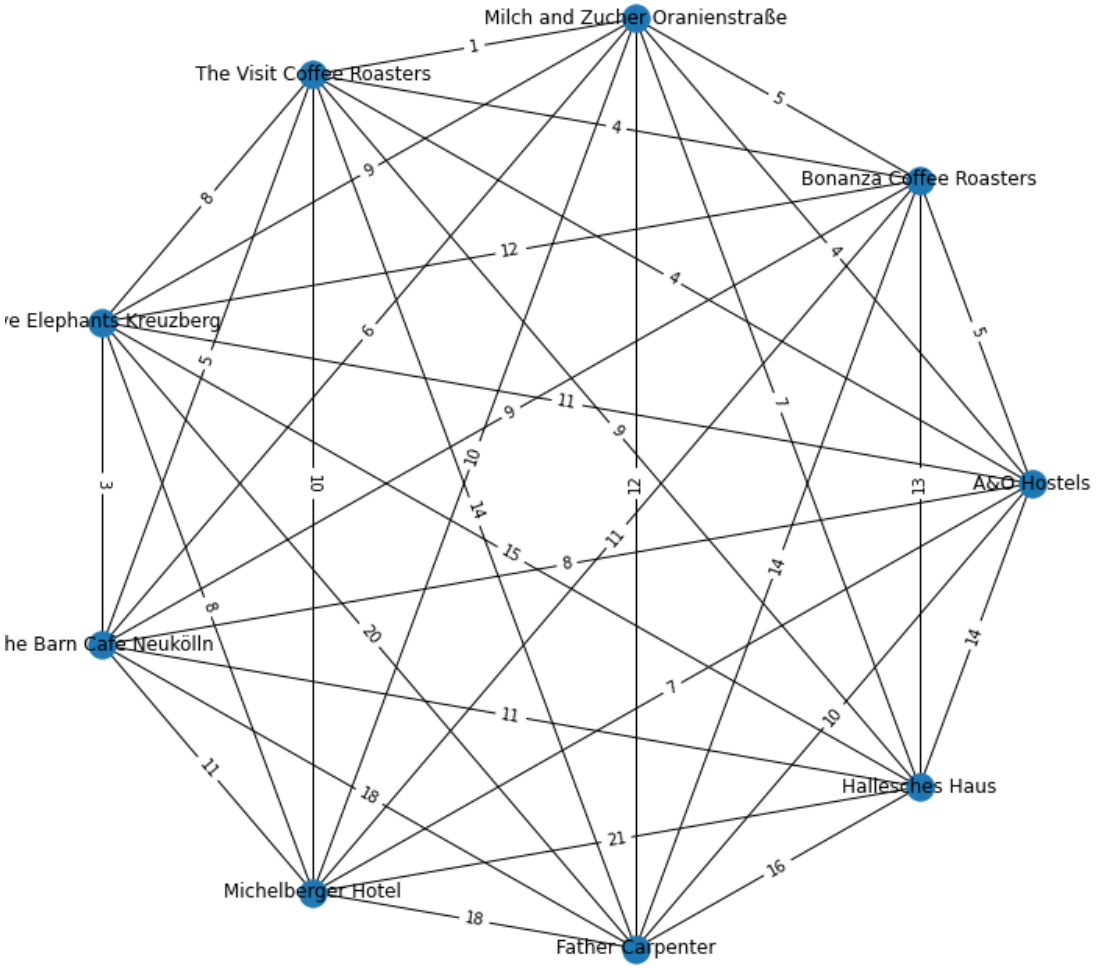
1. THE BARN Café Neukölln
2. Milch and Zucher
3. Bonanza Coffee Roasters
4. Five Elephants Kreuzberg
5. The Visit Cafe
6. Father Carpenter
7. Hallesches Haus
8. Michelberger Hotel

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 $i \neq j$:

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

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

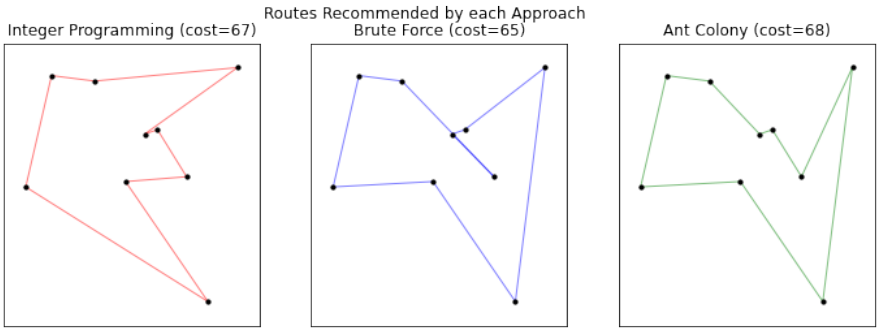
3.2 Objective

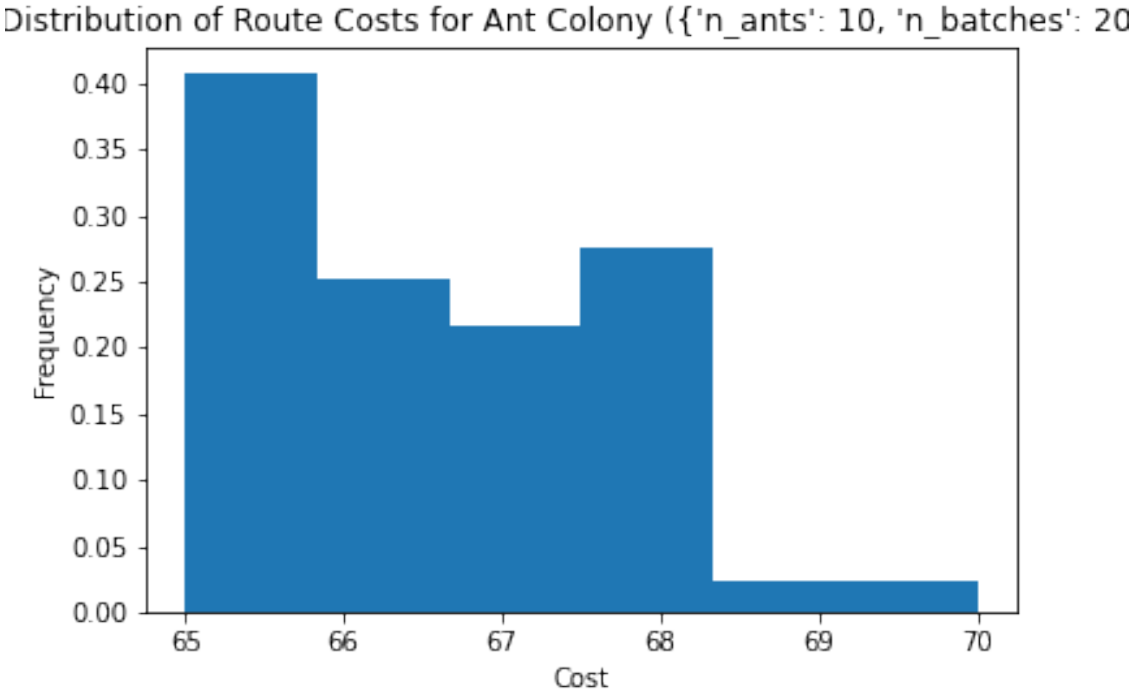
We want to find an expression for the cost of a route desribed by the design variables x_{ij}. Let c_{ij} represent the cost of traveling from node i to node j in the network. Then, the cost of the route will be:

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

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 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.

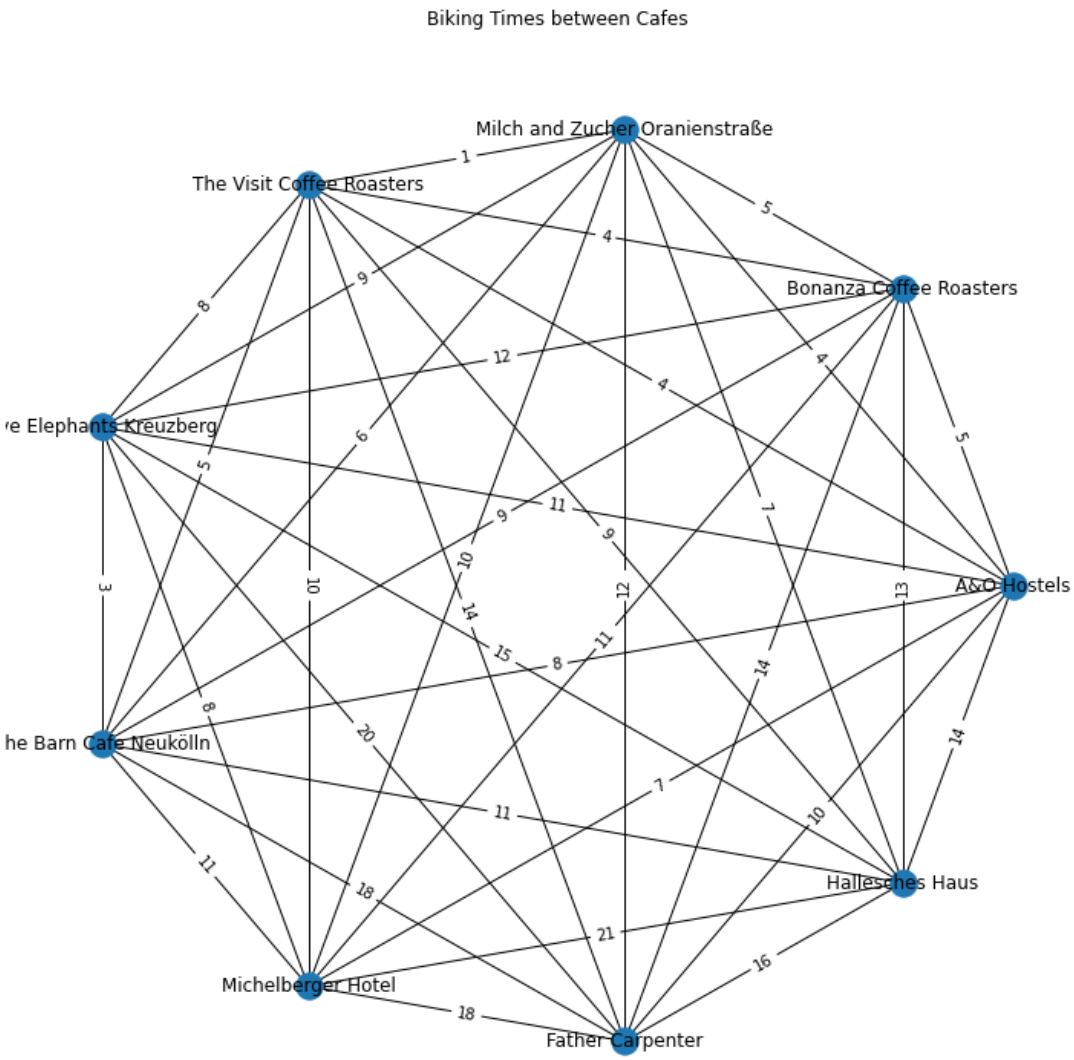
5.1 Appendix: Code

[10]:
```
import numpy as np
import random
import networkx as nx
import cvxpy as cp
import matplotlib.pyplot as plt
import itertools
```

[11]:
```
np.random.seed(0)
random.seed(0)
```

[12]:
```
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]
])
```

[13]:
```
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")
```



[14]:
```
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.
    cp.Problem(objective, constraints).solve()

    # convert the result to a list of visited nodes
    next_pos = {}
    for cafe in range(n):
        next_pos[cafe] = np.nonzero(X.value[cafe,:])[0][0]

    pos = 0
    next = None

    route = [cafes[pos]]
    route_cost = 0

    while next != 0:
        next = next_pos[pos]
        route_cost += cost[pos, next]
        route.append(cafes[next])
        pos = next

    return route, route_cost
```

```
ip_route, ip_cost = integer_programming(cafes, cost)
ip_route, ip_cost
```

[14]:
```
(['A&O Hostels',
  'Bonanza Coffee Roasters',
  'Milch and Zucher Oranienstraße',
  'The Visit Coffee Roasters',
  'Hallesches Haus',
  'The Barn Cafe Neukölln',
  'Five Elephants Kreuzberg',
  'Michelberger Hotel',
  'Father Carpenter',
  'A&O Hostels'],
 67)
```

```
Long-step dual simplex will be used
```

[15]:
```
def brute_force(cafes, g):
    best_cost = np.inf
    best_route = None

    # iterate over possible routes
    for route in itertools.permutations(cafes[1:]):

        # calculate cost of route
        route_cost = 0
        position = cafes[0]
        for next in list(route) + [cafes[0]]:
            route_cost += g.edges[position, next]["weight"]
            position = next

        # check if this is the new winner
        if route_cost < best_cost:
            best_cost = route_cost
            best_route = [cafes[0]] + list(route) + [cafes[0]]

    return best_route, best_cost

bf_route, bf_cost = brute_force(cafes, g)
bf_route, bf_cost
```

[15]:
```
(['A&O Hostels',
  'Michelberger Hotel',
  'Five Elephants Kreuzberg',
  'The Barn Cafe Neukölln',
  'Bonanza Coffee Roasters',
  'The Visit Coffee Roasters',
```

```
 'Milch and Zucher Oranienstraße',
 'Hallesches Haus',
 'Father Carpenter',
 'A&O Hostels'],
65)
```

[16]:
```python
class AntColony:
    def __init__(self, g, start, alpha=1, beta=1, rho=0.99, q=1, n_ants=10,␣
↪n_batches=10):
        """The ant colony optimization algorithm for the TSP

        Args:
            g (networkx.Graph): graph to solve ("weight" attribute carries cost)
            start (Any): starting node
            alpha (float): relative importance of pheromone
            beta (float): relative importance of edge cost
            rho (float): pheromone evaporation factor
            q (float): pheromone deposit factor
            n_ants (int): number of ants per batch
            n_batches (int): number of batches in a run
        """
        self.g = g
        self.start = start

        # optimization parameters
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.q = q

        # length of run
        self.n_ants = n_ants
        self.n_batches = n_batches

        # best found route
        self.best_route = None
        self.best_cost = np.inf

        # initialize pheromones
        for edge in self.g.edges:
            self.g.edges[edge]["pheromone"] = 1.0

    def evaporate(self):
        """Exponential decay of pheromones by a factor of rho"""
        for edge in self.g.edges:
            self.g.edges[edge]["pheromone"] *= self.rho
```

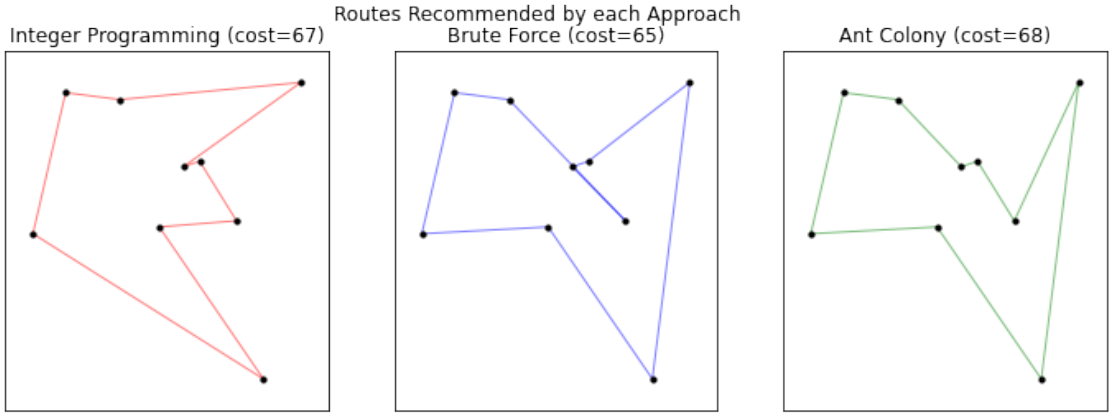
```
    def get_route(self):
        """Route a single ant"""
        position = self.start
        cost = 0
        route = [position]
        options = set(g.neighbors(position)) - set(route)

        while options:
            # choose next destination
            probabilities = []
            for option in options:
                probabilities.append(
                    self.g.edges[position, option]["pheromone"] ** self.alpha *
                    (1 / self.g.edges[position, option]["weight"]) ** self.beta
                    )

            probabilities = np.array(probabilities)
            probabilities /= np.sum(probabilities)

            next = np.random.choice(list(options), p=probabilities)

            # move to next position
            route.append(next)
            cost += self.g.edges[position, next]["weight"]
            position = next

            options = set(g.neighbors(position)) - set(route)

        # move back to start position
        route.append(self.start)
        cost += self.g.edges[position, self.start]["weight"]

        # update best route
        if cost < self.best_cost:
            self.best_route = route
            self.best_cost = cost

        return route, cost

    def deposit(self, route, cost):
        """Deposit pheromones along the route takes by an ant

        Args:
            route (list): route taken by an ant
            cost (float): cost of the route
        """
        for i in range(0, len(route)-1):
```

```
            self.g.edges[route[i], route[i+1]]["pheromone"] += self.q / cost

    def batch(self):
        """Run a single batch of ants"""
        # evaporate pheromones
        self.evaporate()

        # run ants
        routes = []
        costs = []
        for i in range(self.n_ants):
            route, cost = self.get_route()
            routes.append(route)
            costs.append(cost)

        # deposit pheromones
        for route, cost in zip(routes, costs):
            self.deposit(route, cost)

    def run(self):
        for i in range(self.n_batches):
            self.batch()
        return self.best_route, self.best_cost

ac_route, ac_cost = AntColony(g=g, start=cafes[0]).run()
ac_route, ac_cost
```

[16]:
```
(['A&O Hostels',
  'Michelberger Hotel',
  'Five Elephants Kreuzberg',
  'The Barn Cafe Neukölln',
  'The Visit Coffee Roasters',
  'Milch and Zucher Oranienstraße',
  'Bonanza Coffee Roasters',
  'Hallesches Haus',
  'Father Carpenter',
  'A&O Hostels'],
 68)
```

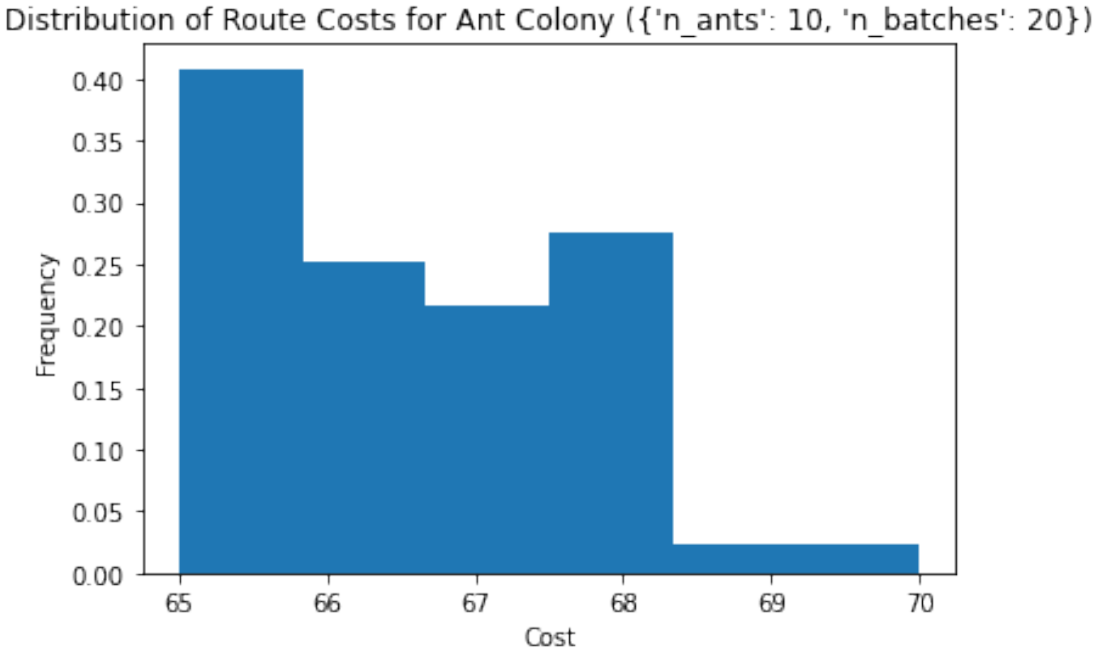
[17]:
```
def route_comparison(routes, costs):
    fig, axes = plt.subplots(1, 3, figsize=(12, 4))

    pos = nx.kamada_kawai_layout(g)

    names = ["Integer Programming", "Brute Force", "Ant Colony"]
    colors = ["red", "blue", "green"]
```

```
    for route, name, route_cost, color, ax in zip(routes, names, costs, colors,␣
↪axes):
        ax.set_xticks([])
        ax.set_yticks([])

        ax.set_title(f"{name} (cost={route_cost})")

        nx.draw_networkx_nodes(g, pos, ax=ax, node_size=10, node_color="black")
        nx.draw_networkx_edges(g, pos, ax=ax, edgelist=list(zip(route[:-1],␣
↪route[1:])), edge_color=color, alpha=0.5)

    fig.suptitle("Routes Recommended by each Approach")

    plt.savefig("imgs/comparison.png")

routes = [ip_route, bf_route, ac_route]
costs = [ip_cost, bf_cost, ac_cost]
route_comparison(routes, costs)
```



[19]:
```
def ac_costs_hist(n_samples, **ac_params):
    costs = []

    for _ in range(n_samples):
        ac = AntColony(g=g, start=cafes[0], **ac_params)
        ac.run()
        costs.append(ac.best_cost)

    plt.hist(costs, density=True, bins=min(np.unique(costs).size, 8))
    plt.title(f"Distribution of Route Costs for Ant Colony ({dict(ac_params)})")
    plt.xlabel("Cost")
    plt.ylabel("Frequency")
    plt.savefig("imgs/ac_hist.png")
```

```
ac_costs_hist(100, n_ants=10, n_batches=20)
```



[ ]: