Group Project: LBA

Akmarzhan Abylay, Chantsaldiimaa Lkhagvatogtokh, and Marcela Radilla Deloya

CS164 Fall 2020

Destinations

As London lockdown restrictions recently ended and two of our three members are currently in London, we chose London as our city, and we compiled all the places we wanted to visit in the city. These include magnificent palaces, famous bridges, and other historical sites. We decided to do a walking tour because we want to stay active and get some fresh air during this pandemic and do not want to be restricted by the working hours of public transport.

The sites we want to visit are Trafalgar Square, Buckingham Palace, Birdcage Walk, Westminster Abbey, The Houses of Parliament, Big Ben, Westminster Bridge, London Eye, The Southbank Centre, Waterloo Bridge, the OXO Tower, Tate Modern, Shakespeare's Globe, Millennium Bridge, and St Paul's Cathedral (see Appendix C for photos). We numbered them in this exact order from 1 to 15 for the matrix.

Estimated Travel Distances

We used Google Maps to estimate how long it takes to walk from one site to another (see Appendix B). The estimated distance between each pair of tour destinations is shown in Table 1. We get a symmetrical matrix/table as it takes the same time to walk from destination A to B as it does for vice versa.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	17	15	14	19	13	19	20	14	12	21	28	31	28	28
2		0	6	14	17	17	34	33	29	29	36	48	46	47	45
3			0	9	11	11	31	27	24	26	31	43	41	40	44
4				0	4	4	19	21	18	21	25	37	35	34	40
5					0	3	25	19	17	20	24	36	40	39	45
6						0	23	16	13	18	21	32	31	30	37
7							0	25	17	11	22	33	32	26	30
8								0	5	11	13	24	23	22	23

9									0	8	8	20	18	17	26
10										0	11	23	21	20	22
11											0	7	10	9	17
12												0	5	6	13
13													0	5	11
14														0	7
15															0

Table 1. Estimated distances measured in minutes by walking.

TSP Formulation

The traveling salesman problem can be formulated as an integer program, and we use Miller–Tucker–Zemlin formulation for our problem (Travelling Salesman Problem, n.d.).

- We define a binary variable x_{ij} where $x_{ij} = 1$ if there is a path between site i and site j. Otherwise, $x_{ij} = 0$.
- We define c_{ij} to represent the time it takes to walk from site i to site j or vice versa (since our time matrix is symmetrical).
- With $n = 15$ sites, the objective function that represents the total time it takes to visit all sites becomes:

$$\min \sum_{i=1}^{n} \sum_{j=1,\, j \neq i}^{n} c_{ij} x_{ij}$$

Since we want to visit each site only once and come back to the site where we started the tour, we create two constraints related to leaving and arriving at each site only once:

- For every site $i \neq j$, we arrive at a site only once (from exactly one other site):

$$\sum_{i=1,\, i \neq j}^{N} x_{ij} = 1 \quad for\, j = 1 \ldots N$$

- For every site $i \neq j$, we leave a site only once (to exactly one other site):

$$\sum_{j=1, j\neq i}^{N} x_{ji} = 1 \ \ for\ i = 1...N$$

Furthermore, to ensure that we have a single route connecting all cities and to avoid sub-tours, we create another constraint. For this, we first define an auxiliary variable u_i that represents the site order, where $u_i < u_j$ means the route visits site i before site j .

To ensure that we don't have sub-tours, we must have the following constraints with auxiliary variables:

$$u_i - u_j + nx_{ij} \leq n - 1, \quad 2 \leq i \neq j \leq n$$

$$1 \leq u_i \leq n - 1, \ \ 2 \leq i \leq n$$

In our case, the numbering in Python starts from 0, so we changed the constraint to fit.

Key assumptions:

- The problem is symmetric: the walking time between sites is the same in both directions.
- We know that the distance between a site to itself is 0. However, we set it to be a large positive number as we don't want to repeatedly pick the current site and get trapped there forever without visiting the rest of the sites.

Optimal Solution

As we computed using CVXPY, the minimum time it takes to visit all sites is 144 minutes (2 hours and 24 minutes), and it happens when we visit the sites in the following order (See Appendix A):

1. Trafalgar Square (coming back to this)	2. Westminster Bridge	3. Waterloo Bridge
4. St. Paul's Cathedral	5. Millennium Park	6. Shakespeare's Globe
7. Tate Modern	8. OXO Tower	9. The Southbank Centre
10. London Eye	11. Big Ben	12. The House of Parliament
13. Westminster Abbey	14. Birdcage Walk	15. Buckingham Palace

Division of Tasks

We discussed how to approach the assignment together and had several calls and coworking sessions. Our main focus of tasks:

- **Akma:** code work (i.e., writing, debugging, TSP formulation)
- **Dima:** write-up (i.e., intro, descriptions, TSP formulation, results)
- **Marcela:** write-up (i.e., TSP formulation description)

Appendix A: CVXPY Code

The below code solves the TSP and could also be found here.

```
#importing the libraries
import cvxpy as cvx
import numpy as np
```

```
#actual locations with coming back to the first location
places = ['Trafalgar Square', 'Buckingham Palace', 'Birdcage Walk', 'Westminster Abbey',
          'The Houses of Parliament', 'Big Ben', 'Westminster Bridge', 'London Eye',
          'The Southbank Centre', 'Waterloo Bridge', 'OXO Tower', 'Tate Modern',
          'Shakespeare's Globe', 'Millennium Bridge', 'St Paul's Cathedral', 'Trafalgar Square']

#distance matrix for 15 cities
c = np.matrix([[1000, 17, 15, 14, 19, 13, 19, 20, 14, 12, 21, 28, 31, 28, 28],
               [17, 1000, 6, 14, 17, 17, 34, 33, 29, 29, 36, 48, 46, 47, 45],
               [15, 6, 1000, 9, 11, 11, 31, 27, 24, 26, 31, 43, 41, 40, 44],
               [14, 14, 9, 1000, 4, 4, 19, 21, 18, 21, 25, 37, 35, 34, 40],
               [19, 17, 11, 4, 1000, 3, 25, 19, 17, 20, 24, 36, 40, 39, 45],
               [13, 17, 11, 4, 3, 1000, 23, 16, 13, 18, 21, 32, 31, 30, 37],
               [19, 34, 31, 19, 25, 23, 1000, 25, 17, 11, 22, 33, 32, 26, 30],
               [20, 33, 27, 21, 19, 16, 25, 1000, 5, 11, 13, 24, 23, 22, 23],
               [14, 29, 24, 18, 17, 13, 17, 5, 1000, 8, 8, 20, 18, 17, 26],
               [12, 29, 26, 21, 20, 18, 11, 11, 8, 1000, 11, 23, 21, 20, 22],
               [21, 36, 31, 25, 24, 21, 22, 13, 8, 11, 1000, 7, 10, 9, 17],
               [28, 48, 43, 37, 36, 32, 33, 24, 20, 23, 7, 1000, 5, 6, 13],
               [31, 46, 41, 35, 40, 31, 32, 23, 18, 21, 10, 5, 1000, 5, 11],
               [28, 47, 40, 34, 39, 30, 26, 22, 17, 20, 9, 6, 5, 1000, 7],
               [28, 45, 44, 40, 45, 37, 30, 23, 26, 22, 17, 13, 11, 7, 1000]])

#route
def route(x):
    """Function for finding the optimal path."""

    #starting from the first node
    tour = [0]
    while True:
        for j in range(len(c)):
            if x[tour[-1],j]==1:
                if j in tour:
                  #appending the correct path
                  tour.append(0)
                  return tour
                else:
                    tour.append(j)


def tsp(c):
    """Function for solving the TSP."""

    #variable matrix with binary entries that define our root
    x = cvx.Variable(c.shape, boolean=True)

    #extra matrix for help in constraints
    ones = np.ones((len(c), ))

    #objective function
    obj = cvx.Minimize(sum([c[i,:]*x[:,i] for i in range(len(c))]))

    #constraints, so that there is one 1 in each column and row
    const = [(cvx.sum(x,axis=0) == ones), (cvx.sum(x, axis=1) == ones.T)]

    #subtour elimination
    #extra variable for subtour elimination
    u = cvx.Variable(len(c))
```

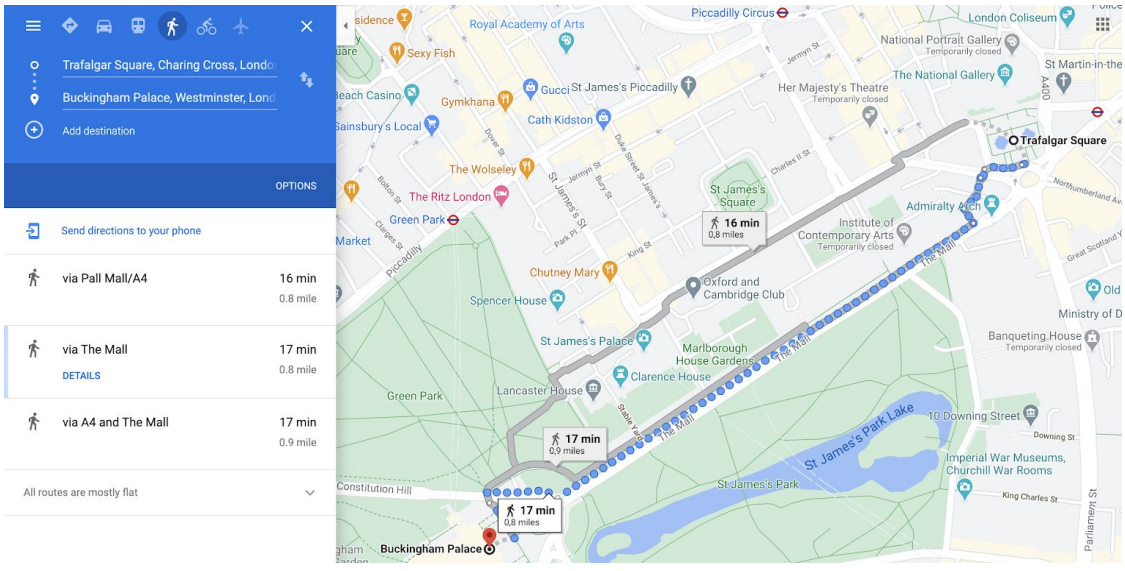
```
    for i in range(1, len(c)):
        for j in range(1, len(c)):
            if i != j:
                const.append(u[i]-u[j]+len(c)*x[i,j]<=len(c)-1)

    #constraints on the extra variable u
    for i in range(len(c)):
        const.append(u[i] >= 0)

    prob = cvx.Problem(obj, const)
    opt = prob.solve()

    return opt, x

def results(path, opt, places):
    """Function for printing the results."""

    string = []
    string.append(input("What time do you plan on starting?\nInput your answer as hour:minute (e.
g., 9:30).\nStarting time: "))
    print("\nWe identified a route for you! Please visit the places in the following order:\n")
    for i, j in enumerate(path):
        time = c[path[i], path[i+1]] if i!=len(c) else c[path[-1], path[0]]

        #splitting for minutes and hours
        colon = string[i].find(":")
        minutes, hours = int(string[i][colon+1:]), int(string[i][:colon])
        end = minutes+time

        #defining the edge cases for transitions
        minutes = (end-60) if end//60==1 else end
        hours = hours+1 if end//60==1 else hours
        hours -= 24 if hours//24==1 else 0

        if minutes<10 and hours<10:
            string.append("0"+str(hours)+":"+"0"+str(minutes))
        elif minutes<10:
            string.append(str(hours)+":"+"0"+str(minutes))
        elif hours<10:
            string.append("0"+str(hours)+":"+str(minutes))
        else:
            string.append(str(hours)+":"+str(minutes))

        #printing destinations
        if i==0:
            print("{}. START: [{}] {}\n".format(i+1, string[i], places[j]))
        elif i==len(c):
            print("{}. COME BACK: [{}-{}] {}\n".format(i+1, string[i-1], string[i], places[j]))
        else:
            print("{}. Walk: [{}-{}] {}\n".format(i+1, string[i-1], string[i], places[j]))

    print('It will take you {} hours and {} minutes. Enjoy!'.format(int(opt)//60, int(opt)%60))

opt, x = tsp(c)
results(route(x.value), opt, places)
```

```
What time do you plan on starting?
Input your answer as hour:minute (e.g., 9:30).
Starting time: 6:45

We identified a route for you! Please visit the places in the following order:

1. START: [6:45] Trafalgar Square

2. Walk: [6:45-07:04] Westminster Bridge

3. Walk: [07:04-07:15] Waterloo Bridge
```

```
4. Walk: [07:15-07:37] St Paul's Cathedral
5. Walk: [07:37-07:44] Millennium Bridge
6. Walk: [07:44-07:49] Shakespeare's Globe
7. Walk: [07:49-07:54] Tate Modern
8. Walk: [07:54-08:01] OXO Tower
9. Walk: [08:01-08:09] The Southbank Centre
10. Walk: [08:09-08:14] London Eye
11. Walk: [08:14-08:30] Big Ben
12. Walk: [08:30-08:33] The Houses of Parliament
13. Walk: [08:33-08:37] Westminster Abbey
14. Walk: [08:37-08:46] Birdcage Walk
15. Walk: [08:46-08:52] Buckingham Palace
16. COME BACK: [08:52-09:09] Trafalgar Square
It will take you 2 hours and 24 minutes. Enjoy!
```

Appendix B: Google Maps

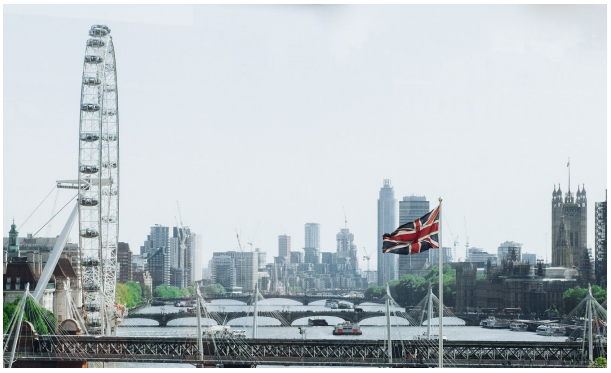
This is to show that we simply used Google Maps to find out the estimated travel distances between each pair of tour destinations. The following shows the distance between the first two destinations in our list – from Trafalgar Square to Buckingham Palace.



Appendix C: Images of Sites

The images without references are taken from Google Photos (Google Photos, n.d.).

1. Trafalgar Square 	2. Buckingham Palace 
3. Birdcage Walk 	4. Westminster Abbey 
5. The Houses of Parliament 	6. Big Ben (Photo by Marcin Nowak) 

7. Westminster Bridge  (Photo by dylan nolte on Unsplash)	8. London Eye  (Photo by Nicholas Doherty on Unsplash)
9. The Southbank Centre 	10. Waterloo Bridge 
11. The OXO Tower (by James Newton) 	12. Tate Modern (by Alastair Maw) 

13. Shakespeare's Globe

14. Millennium Bridge

15. St Paul's Cathedral

References

Google Photos. (n.d.). Retrieved from https://www.google.com/photos/about/.

Parsons, G. (2019). This Is (Possibly) The Most Efficient Sightseeing Tour Of London. Retrieved from https://secretldn.com/sightseeing-london-map-walking-route/

Yongtwang. (2017). Operations Research 09E: Traveling Salesman Problem - Integer Programming. Retrieved from https://www.youtube.com/watch?v=nRJSFtscnbA

“Travelling salesman problem”. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Travelling_salesman_problem