CS164 - LBA

Juan Castro Fernandez, Patrick Mutuku, and Ha Nguyen

Fall 2019 - Professor Shekhar

Introduction

A Minervan's first day in a new rotation city is often most exciting yet confusing. The urge to explore the city and start anew is suppressed by language barriers, an unfamiliar topography, and a lack of knowledge about locations that are crucial to their everyday life.

Empathizing with this problem, the Student Experience team conducts neighborhood tours at the start of every semester. With a limited number of RAs, they must design the tours to cover as many locations as possible in the shortest time. This planning problem is analogous to the travelling salesman problem and can be solved using integer programming methods.

I - Justification of chosen destinations

The following list outlines the locations that were chosen. Our aim is to equip Minervans with the knowledge of places that they would want to visit on a daily/weekly basis.

1. *Melchiorstrasse 32*. This obscure place houses a lazy RA, who conveniently decided that the tour must start at their doorstep.
2. *Superfit Mitte*. As physical well-being is the foundation for nurturing critical wisdom, it is crucial that Minervans know where they can exercise. Superfit is an affordable gym that offers discounts as a reward for the dedicated Minervans who cycle two kilometers each way in the freezing German winter for the sake of their toned buttocks.
3. *Edeka Annenstrasse*. This mid-range supermarket caters to all lifestyles with their diverse offerings, from shea butter-laced toilet paper to vegan pork.
4. *Haci Baba Kreuzberg*. Haci Baba offers affordable Turkish food and a store owner who will not verbally judge Minervans when they use a fork to eat doner kebab.
5. *Tresor Berlin*. Tresor club offers vivacious mixtapes of techno music, which is Berlin's specialty. Minervans are guaranteed to feel more robotic than their TSP algorithm by the time they leave.
6. *Berliner Stadtbibliothek*. This spacious and well-equipped library offers everything Minervans need during their active hours: a quiet place to grind their assignments, dirt-cheap cold food to power their brains, and grumpy old German men who pick a fight with them for talking in the group-study room to improve their conflict resolution skills.

II - Estimated travel distances between destinations

We implemented a Google Maps API in the code to obtain the travel distances in minutes between every pair of destinations. The function `get_coordinates` uses Google's Geocode API to first obtain the coordinates of a list of provided locations. These coordinates are used in the `create_cost_matrix` function, where the travel mode is also specified (driving, bicycling, walking, transit) as well as the departure time.

Using the six destinations of the neighborhood tour, and the travel mode being by bicycle (all Minervans should get bikes in Berlin!), we obtain the cost matrix below:

```
Cost Matrix (times are in minutes). The estimates are based on the following information:
Departure Time: Wednesday, December 18, 2019 06:30:00
Travel Mode: bicycling
```

	Melchiorstrasse 32, Berlin	Superfit Mitte	Edeka Annenstrasse	Hacibaba Kreuzberg	Tresor Berlin	Berliner Stadtbibliothek
Melchiorstrasse 32, Berlin	1000.0	8.0	3.0	3.0	2.0	7.0
Superfit Mitte	6.0	1000.0	5.0	8.0	5.0	7.0
Edeka Annenstrasse	3.0	7.0	1000.0	4.0	3.0	5.0
Hacibaba Kreuzberg	3.0	10.0	4.0	1000.0	5.0	8.0
Tresor Berlin	2.0	7.0	3.0	5.0	1000.0	6.0
Berliner Stadtbibliothek	7.0	7.0	5.0	8.0	6.0	1000.0

Note that the time between two locations may be different depending on which one is the starting point and which one is the ending point. For example, between"Superfit Mitte" and "Melchiorstrasse 32, Berlin" it takes 6 minutes to go from the gym to the res hall and 8 minutes from the res hall to the gym. Google Maps takes into account that from the res to the gym one goes uphill, whereas from the gym to the res one goes downhill.

With our Google Maps API implementation, one can get the cost matrix of any list of any number of addresses, choose any mode of transportation, and any time of departure, and google maps will account for the predicted traffic, weather, construction sites, etc., and will provide us with the most accurate cost matrix.

III - Mathematical formulation

We heavily draw on Chapter 9 of *Applied mathematical programming* (Bradley et. al, 1992) to formulate the TSP. Let c_{ij} be the time it takes to travel from site i to site j (in minutes). Let x_{ij} be a binary variable that takes the value of 1 if we travel from site i to site j and 0 if otherwise. Our problem is then:

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

In our code, `cost_matrix` contains all the c_{ij}. The diagonal of the matrix is assigned a large number (1000) to ensure that $x_{ii} = 0$ (self-loops) in the solution to our problem. Our `choice_matrix` contains all the binary variables x_{ij}. The objective function is rewritten as:

```
cp.Minimize(cp.sum(flat_cost_matrix*choice_matrix))
```

Since we want to pass by each site exactly one, we must impose constraints to ensure each site is entered and exited once:

$$\sum_{i=1}^{n} x_{ij} = 1 \; (j = 1, 2, ..., n; j \neq i) \quad \text{and} \sum_{j=1}^{n} x_{ij} = 1 \; (i = 1, 2, ..., n; i \neq j)$$

$$x_{ij} \in \{0, 1\} \; (i = 1, 2, ..., n; j = 1, 2, ..., n)$$

In our code, we enforce these constraints by ensuring the sum of each row and each column in `choice_matrix` equals 1.

Subtour elimination:

Let $u_i \; (i = 1, 2, ..., n)$ be our auxiliary variables for subtour elimination. Our constraint is thus:

$$u_i - u_j + n x_{ij} \leq n - 1 \; (i = 2, ..., n; j = 2, ..., n; i \neq j)$$

$$x_{ij} \in \{0, 1\} \; (i = 1, 2, ..., n; j = 1, 2, ..., n)$$

$$u_i \geq 0 \; (i = 1, 2, ..., n)$$

Eliminating subtours can be done using a few approaches. Subtours can be eliminated by explicitly adding constraints that prevent subtours of a certain length. In our instance with 6 locations, possible subtours are of length 2,3,4. It is sufficient to only add constraints for subtours of length 2 and 3 (2 is the complement of 4 in our six location example). To add

constraints for subtours of length 2, we generate all possible combinations of pairs from the six locations and form an expression for each i,j then the sum of values in the choice_matrix is at most 1.

$$choiceMatrix[i][j] + choiceMatrix[j][i] \leq 1$$

This method requires explicitly identifying possible subtour sizes and adding them to the list of constraints. If we change to a higher number of say 12 locations, there can now be subtours of length 6 hence this approach is not generalizable. Using MTZ method to formulate the constraints is more compact and simpler to implement in a few lines of code. Intuitively, we create node potentials for all locations excluding the starting location and then add constraints such that no subtours are present which do not include the first node (start location). When these constraints are met, it's easy to see that there is only one continuous tour. An example can be found in Appendix A.

IV - Optimal Solution

The following is the choice matrix that represents the optimal solution returned by `solve_TSP_MTZ` using the CVXPY library and implementing MTZ subtour elimination:

Traveling Solution Matrix: (Estimated time = 26.0 mins)

	Melchiorstrasse 32, Berlin	Superfit Mitte	Edeka Annenstrasse	Hacibaba Kreuzberg	Tresor Berlin	Berliner Stadtbibliothek
Melchiorstrasse 32, Berlin	0.0	0.0	0.0	1.0	0.0	0.0
Superfit Mitte	0.0	0.0	0.0	0.0	1.0	0.0
Edeka Annenstrasse	0.0	0.0	0.0	0.0	0.0	1.0
Hacibaba Kreuzberg	0.0	0.0	1.0	0.0	0.0	0.0
Tresor Berlin	1.0	0.0	0.0	0.0	0.0	0.0
Berliner Stadtbibliothek	0.0	1.0	0.0	0.0	0.0	0.0

Starting from the residence hall (Melchiorstrasse 32), we must follow the path: Melchiorstraße → Hacibaba → Edeka → Stadbibliothek → Superfit → Tresor → Melchiorstraße. The estimated travel time is 26 mins by bike. This estimate was very accurate. The time that took me to follow it on my bike was around 28 mins (the extra time was probably due to the time it took me to take selfies).

If one wants to experiment with different locations (in any city of the world), travel modes, and departure times, we have created the function `main_function` that takes as parameters a list of locations, travel mode (default = bike) , and departure time (default = now). This function will print the Cost Matrix, solve the Optimization Problem, and give you the route you need to follow with the estimated time it will take to travel it.

V - Contribution distribution

Juan Castro Fernandez: Implemented a Google Maps API to estimate the travel distance between each location, wrote sections II and IV, went on tour and took selfies.

Patrick Mutuku: Wrote the code for the Optimization Problem (solve_TSP_MTZ function), explained MTZ subtour elimination.

Ha Nguyen: Wrote the mathematical formulation of the TSP, wrote Introduction and section I and V.

Bibliography

Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1992).Applied mathematical programming. Reading, Mass.: Addison-Wesley. Chapter 9, section 9.1 Retrieved from http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

Diamond, S and Chu, E and Boyd, S. (2014). CVXPY 0.4.11 documentation [online]. Retrieved from http://www.cvxpy.org/en/latest/index.html

Appendix A: Example of MTZ subtour elimination

Suppose in our 6 location example we have subtours of length 2 and 4 such that 1⇒2⇒5⇒ 6⇒ 1 and 3⇒ 4⇒ 3. Using

$u_i - u_j + nx_{ij} \leq n - 1$ $(i = 2, ..., n; j = 2, ..., n; i \neq j)$ on the subtour of length 2:

$u_3 - u_4 + 6 \leq 5$ ($x_{ij} = 1$ *since we go from* 3 *to* 4)

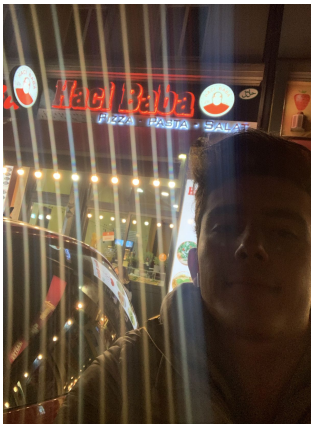
$u_4 - u_3 + 6 \leq 5$

Adding the two inequalities results in $6 \leq 5$ which can never be satisfied. Thus, no subtour that does not include the start node is ever allowed.

As illustrated in our code, this subtour elimination method has $O(n^2)$ complexity, which makes it computationally expensive for large problems. However, with a small number of locations, this method applies well.

Appendix B: Selfies

Melchiorstrasse 32



HaciBaba

Edeka

Stadtbibliothek

Superfit

Tresor

Melchiorstrasse 32

Appendix C: Code Implementation

Link to the Google Colab: https://colab.research.google.com/drive/1Z4YjqKHNUcMToYrlUXRRl20Ryh7tx8x0 (https://colab.research.google.com/drive/1Z4YjqKHNUcMToYrlUXRRl20Ryh7tx8x0)

In [17]:
```
#run the line below if it's the first time
#pip install googlemaps
```

In [2]:
```
import numpy as np
import cvxpy as cp
import googlemaps
import datetime
import time
import pandas as pd
from IPython.display import display
```

In [3]:
```
gmaps = googlemaps.Client(key='AIzaSyAfgIQfv0udiVOjojxVfa8U6AbhN7QIDoY')
```

In [4]:
```
#function that gets the coordinates of given addresses using google maps api
def get_coordinates(addresses):
    coords = []
    for address in addresses:
        coords.append(str(list(gmaps.geocode(address)[0]['geometry']['location'].values()))[1:-1])
    return coords
```

In [5]:
```
locations = ['Melchiorstrasse 32, Berlin',
             'Superfit Mitte',
             'Edeka Annenstrasse',
             'Hacibaba Kreuzberg',
              'Tresor Berlin',
              'Berliner Stadtbibliothek']
             #   'East Side Gallery Berlin',
             #  'Memorial Jewish Cemetery Berlin']
```

In [6]:
```
coords = get_coordinates(locations)
```

In [7]:
```
#function to create cost matrix given coordinates of locations using googlemaps API
#in travel_mode one can specify a mode between "driving" (which includes traffic information from Google Maps),
#"bicycling", "walking", and "transit" (which includes public transport infromation from Google Maps)
#in dep_time one can specify the date and time of departure in the format of a list:
#[year,month,day,hour,minute] and the Google API will provide the estimates for the given date and time
def create_cost_matrix(coordinates,travel_mode = "walking",dep_time = None):
    if not dep_time:
      time_dep = datetime.datetime.now()
    else:
      ye,mo,da,ho,mi = dep_time
      t = datetime.datetime(ye,mo,da,ho,mi)
      time_dep = time.mktime(t.timetuple())

    n=len(coordinates)
    #empty matrix
    costmatrix = np.zeros([n,n])

    #obtaining travel time between each two locations
    for i in range(n):
        for j in range(n):
            if i!=j:
                start_point = coordinates[i]
                end_point = coordinates[j]
                search_result = gmaps.directions(start_point,end_point,
                                         mode=travel_mode,
                                         departure_time=time_dep
                                        )
                if not search_result:
                  print('GOOGLE MAPS DID NOT FIND A ROUTE BETWEEN TWO OF YOUR LOCATIONS')
                duration = search_result[0]['legs'][0]['duration']['text']
                if "hour" in duration:
                  indh = duration.index('h')
                  dur = int(duration[:indh])*60+int(duration[indh+4:-4])
                else:
                  dur = int(duration[:-4])
                costmatrix[i][j] = dur
            else:
                costmatrix[i][j] = 1000
    return costmatrix
```

In [8]:
```
#function to create dataframe of the matrix for nice visualization
def nice_matrix(m,colnames):
    df = pd.DataFrame(m, columns = colnames, index = colnames)
    return df
```

In [9]:
```
#inputs
travel_mode = "bicycling"
now = datetime.datetime.now()
departure = [now.year, now.month, now.day, now.hour, now.minute]
#cost matrix
cm = create_cost_matrix(coords,travel_mode=travel_mode,dep_time=departure)

#printing information and nice matrix
df2 = nice_matrix(cm,locations)
ye,mo,da,ho,mi = departure
time_in_secs = time.mktime(datetime.datetime(ye,mo,da,ho,mi).timetuple())
date_and_time = datetime.datetime.fromtimestamp(time_in_secs).strftime("%A, %B %d, %Y %I:%M:%S")
print('Cost Matrix (times are in minutes). The estimates are based on the following information:\nDeparture Time: {0}\nTravel Mode:
{1}'.format(date_and_time,travel_mode))
df2
```

```
Cost Matrix (times are in minutes). The estimates are based on the following information:
Departure Time: Wednesday, December 18, 2019 08:53:00
Travel Mode: bicycling
```

Out[9]:

	Melchiorstrasse 32, Berlin	Superfit Mitte	Edeka Annenstrasse	Hacibaba Kreuzberg	Tresor Berlin	Berliner Stadtbibliothek
Melchiorstrasse 32, Berlin	1000.0	8.0	3.0	3.0	2.0	7.0
Superfit Mitte	6.0	1000.0	5.0	8.0	5.0	7.0
Edeka Annenstrasse	3.0	7.0	1000.0	4.0	3.0	5.0
Hacibaba Kreuzberg	3.0	10.0	4.0	1000.0	5.0	8.0
Tresor Berlin	2.0	7.0	3.0	5.0	1000.0	6.0
Berliner Stadtbibliothek	7.0	7.0	5.0	8.0	6.0	1000.0

In [10]:
```
#MTZ subtour elim
#including subtour elimination ==> MTZ subtour elim
#using the aux variable idea from time 3:14
# of video https://www.youtube.com/watch?v=nRJSFtscnbA

#cost matrix reprenting distances between locations.
#Travel times are assumed symmetric
def solve_TSP_MTZ(cost_matrix):

    s = time.time()
    #variable for the number of locations
    size = cost_matrix.shape[0]

    #matrix representing what cities are visited and in what order
    #boolean=True is a constraint on the values for the matrix to be 1 or 0
    choice_matrix = cp.Variable((size*size), boolean=True)

    #flatten cost matrix for compatible multiplication
    flat_cost_matrix = np.ndarray.flatten(cost_matrix)

    #elementwise multiplication (broadcasting operation) of the flattened arrays gives total travel cost
    objective = cp.Minimize(
         cp.sum(flat_cost_matrix*choice_matrix)
    )

    #obvious constraints here ==> Row-wise and column-wise sums for our matrix must be 1
    #normal 1d numpy array since cvxpy sum emits a 1d array in current version
    expected_sums = np.ones(size)
    constraints = [
                   (cp.sum(cp.reshape(choice_matrix,(size,size)), axis=0) == expected_sums),
                   (cp.sum(cp.reshape(choice_matrix,(size,size)), axis=1) == expected_sums)
    ]

    #constraints specified by MTZ subtour
    aux_var = cp.Variable(size)

    #adding to constraints
    #ui-uj + Nxij <= N-1, for i!=j, i,j = 2... N
    for i in range(1,size):
        for j in range(1, size):
            if i != j:
                  constraints.append(aux_var[i] - aux_var[j] + size*cp.reshape(choice_matrix,(size,size))[i,j] <= size-1)

    #constraint ui >= 0 for i=1... N
    for i in range(size):
          constraints.append(aux_var[i] >= 0)

    #solve problem
    prob = cp.Problem(objective, constraints)
    result = prob.solve()
    t = time.time()-s
    print('Optimization problem using MTZ subtour elimination solved in {}s'.format(np.round(t,3)))
    return np.absolute(np.round(choice_matrix.value)), np.round(result), t
```

In [11]:
```
m, tt, ts = solve_TSP_MTZ(cm)
```

```
Optimization problem using MTZ subtour elimination solved in 0.066s
```

In [12]:
```
colnames = locations
df = nice_matrix(m.reshape(int(len(m)**(1/2)),int(len(m)**(1/2))),colnames)
print('Traveling Solution Matrix: (Estimated time = {} mins)'.format(tt))
df
```

```
Traveling Solution Matrix: (Estimated time = 26.0 mins)
```

Out[12]:

	Melchiorstrasse 32, Berlin	Superfit Mitte	Edeka Annenstrasse	Hacibaba Kreuzberg	Tresor Berlin	Berliner Stadtbibliothek
Melchiorstrasse 32, Berlin	0.0	0.0	0.0	1.0	0.0	0.0
Superfit Mitte	0.0	0.0	0.0	0.0	1.0	0.0
Edeka Annenstrasse	0.0	0.0	0.0	0.0	0.0	1.0
Hacibaba Kreuzberg	0.0	0.0	1.0	0.0	0.0	0.0
Tresor Berlin	1.0	0.0	0.0	0.0	0.0	0.0
Berliner Stadtbibliothek	0.0	1.0	0.0	0.0	0.0	0.0

In [13]:
```
def print_travel_instructions(choice_matrix, locs, estimated_time):
    size = int(choice_matrix.shape[0]**(1/2))
    reshaped_choice_matrix = np.array(choice_matrix).reshape(size, size)
    cur_spot = 0
    spots_visited = 0
    path = ''
    print("Total estimated time is ",int(estimated_time), "minutes if you proceed as follows: ")
    while True:
        if spots_visited == size:
            break
    for i,v in enumerate(reshaped_choice_matrix[cur_spot]):
        if v == 1:
            next_idx = int(i)
    path+='{0} --> {1} \n'.format(locs[cur_spot],locs[next_idx])
    cur_spot = next_idx
    spots_visited += 1
    print(path[:-2])
```

In [14]:
```
print_travel_instructions(m,locations,tt)
```

```
Total estimated time is  26 minutes if you proceed as follows:
Melchiorstrasse 32, Berlin --> Hacibaba Kreuzberg
Hacibaba Kreuzberg --> Edeka Annenstrasse
Edeka Annenstrasse --> Berliner Stadtbibliothek
Berliner Stadtbibliothek --> Superfit Mitte
Superfit Mitte --> Tresor Berlin
Tresor Berlin --> Melchiorstrasse 32, Berlin
```

In [15]:
```
#putting everything together in one function
def main_function(locations,travel_mode = "bicycling",departure_time = "now"):
  if departure_time == "now":
    now = datetime.datetime.now()
    departure = [now.year, now.month, now.day, now.hour, now.minute]
  else:
    departure = departure_time
  coords = get_coordinates(locations)
  cm = create_cost_matrix(coords,travel_mode=travel_mode,dep_time=departure)

  #printing information and nice matrix
  df2 = nice_matrix(cm,locations)
  ye,mo,da,ho,mi = departure
  time_in_secs = time.mktime(datetime.datetime(ye,mo,da,ho,mi).timetuple())
  date_and_time = datetime.datetime.fromtimestamp(time_in_secs).strftime("%A, %B %d, %Y %I:%M:%S")
  print('Cost Matrix (times are in minutes). The estimates are based on the following information:\nDeparture Time: {0}\nTravel Mod
e: {1}'.format(date_and_time,travel_mode))
  display(df2)

  print('\n')

  m, tt, ts = solve_TSP_MTZ(cm)

  colnames = locations
  df = nice_matrix(m.reshape(int(len(m)**(1/2)),int(len(m)**(1/2))),colnames)
  print('Traveling Solution Matrix: (Estimated time = {} mins)'.format(tt))
  display(df)

  print_travel_instructions(m,locations,tt)
```

Experimentation Area for the User

The example below is from San Francisco

In [16]:
```
#(make sure that the names of the locations are specific enough for Google Maps to find them,
#i.e. write “Edeka Annenstrasse” instead of just “Edeka” for Google Maps to know which Edeka you are looking for)

#The following is an example of some of my favorite places in San Francisco (and the two res halls) traveling by transit
locations = ['1412 Market Street, San Francisco',
             '851 California Street, San Francisco',
             'Pier 39 San Francisco',
             'Twin Peaks San Francisco',
             'Ocean Beach San Francisco',
             "Bob's Donuts San Francisco",
             'SF MOMA',
             'Lands End Labyrinth',
             'Golden Gate Park'
             #here you can add or remove places as you wish :)
             ]


#make sure to wirte the departure_time in the format: [year,month,day,hour,minute]
#Example for December 19, 2019 at 8:15 PM
#set the variable equal to the string "now" for choosing now as the departure time
departure_time = [2019,12,19,20,15]

#choose one between 'driving', 'bicycling', 'walking', and 'transit'
mode = 'transit'

main_function(locations,mode,departure_time)
```

```
Cost Matrix (times are in minutes). The estimates are based on the following information:
Departure Time: Thursday, December 19, 2019 08:15:00
Travel Mode: transit
```

	1412 Market Street, San Francisco	851 California Street, San Francisco	Pier 39 San Francisco	Twin Peaks San Francisco	Ocean Beach San Francisco	Bob's Donuts San Francisco	SF MOMA	Lands End Labyrinth	Golden Gate Park
1412 Market Street, San Francisco	1000.0	19.0	33.0	29.0	39.0	17.0	12.0	65.0	40.0
851 California Street, San Francisco	21.0	1000.0	24.0	48.0	55.0	10.0	16.0	63.0	55.0
Pier 39 San Francisco	37.0	20.0	1000.0	57.0	66.0	23.0	28.0	80.0	54.0
Twin Peaks San Francisco	33.0	53.0	60.0	1000.0	52.0	53.0	43.0	75.0	55.0
Ocean Beach San Francisco	40.0	58.0	69.0	52.0	1000.0	59.0	50.0	33.0	33.0
Bob's Donuts San Francisco	21.0	12.0	21.0	50.0	61.0	1000.0	29.0	54.0	49.0
SF MOMA	12.0	13.0	26.0	38.0	48.0	24.0	1000.0	63.0	47.0
Lands End Labyrinth	71.0	64.0	81.0	83.0	36.0	55.0	65.0	1000.0	46.0
Golden Gate Park	40.0	55.0	59.0	68.0	33.0	52.0	48.0	43.0	1000.0

```
Optimization problem using MTZ subtour elimination solved in 0.513s
Traveling Solution Matrix: (Estimated time = 274.0 mins)
```

	1412 Market Street, San Francisco	851 California Street, San Francisco	Pier 39 San Francisco	Twin Peaks San Francisco	Ocean Beach San Francisco	Bob's Donuts San Francisco	SF MOMA	Lands End Labyrinth	Golden Gate Park
1412 Market Street, San Francisco	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
851 California Street, San Francisco	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Pier 39 San Francisco	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
Twin Peaks San Francisco	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Ocean Beach San Francisco	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Bob's Donuts San Francisco	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
SF MOMA	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Lands End Labyrinth	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Golden Gate Park	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

```
Total estimated time is  274 minutes if you proceed as follows:
1412 Market Street, San Francisco --> SF MOMA
SF MOMA --> 851 California Street, San Francisco
851 California Street, San Francisco --> Bob's Donuts San Francisco
Bob's Donuts San Francisco --> Pier 39 San Francisco
Pier 39 San Francisco --> Golden Gate Park
Golden Gate Park --> Lands End Labyrinth
Lands End Labyrinth --> Ocean Beach San Francisco
Ocean Beach San Francisco --> Twin Peaks San Francisco
Twin Peaks San Francisco --> 1412 Market Street, San Francisco
```