UC DAVIS

Bi-objective Time-Dependent Dynamic Shortest Path Problem for Modal Choice Application

Kalai Ramea

ECI 253, Final Project

E-Mail: kramea@ucdavis.edu

Table of Contents

1. Introduction

Shortest path problems have a numerous applications in the transportation field. When it comes to mode choice problems, there are number of factors that are taken into account while choosing the optimal mode between two points. Typically, the factors include total travel time, cost of travel, inconvenience, resource consumption and so on, and the importance of each factor may vary from person to person. This project will limit the number of objectives to two, and will compare the optimal mode choices for different income groups for the Bay Area Rapid Transit (BART) network in the San Francisco Bay Area. It will also limit the study for 'peak hours', since the off-peak mode competition can give an unfair advantage to driving as the train schedule intervals are wide and no congestion delays happen to driving.

2. Problem Definition

This project will focus on the competition between the three modes of choice (Driving alone, Carpooling, taking transit) for the different income groups ranging from minimum wage to top 5%, for the BART train network in the Bay Area. The objective function is two-fold in this case, it is designed to first minimize the total travel time, and second, it is designed to minimize the total travel cost, and the optimal mode based on these two is then selected. The different terminologies that will be used in the problem are defined below:

- Bi-objective: There are two objectives to this problem, as stated above. They are (a) Minimize the total travel distance; (b) Minimize the total travel cost.
- Total Travel Time: Time taken by the mode to travel from origin to destination. For cars, the time includes the congestion delays during the travel, and for the train, it includes the waiting time and transfer time (if applicable) along with the time taken by the train to reach the destination.
- Total Travel Cost: This is the total cost that will take to travel from origin to destination. The cost components differ for each mode, which is explained in detail below:
    - *Cars:* There are mainly three components of cost in this mode: Fuel cost, Time cost, Parking and Toll costs.
        - Fuel cost depends on the distance of travel, mileage of the car and the current cost of gasoline. It is calculated by the formula given below:

$$Fuel\ Cost, \alpha_C = \frac{G.A_C}{\rho}$$

where α_C = Fuel Cost ($)
G = Cost of Gasoline ($/gallon)
A_C= Total distance traveled from through car (miles)
ρ = Average mileage of the car used (miles/gallon)

- Time cost is determined based on the value of time perceived by the user (in $/hr), it differs for each income group (shown in Table 1). The value of time is typically 50% of the hourly wage of each income group. The value of time is then multiplied by the 'Total Travel Time' to obtain the time cost.
- Parking and Toll costs: The city of San Francisco and city of Oakland are very expensive when it comes to parking. Also, the number of parking spaces is very limited, especially during the weekdays. Moreover, a toll of $5 is charged for cars (driving alone) and $2.50 is charged for carpooling vehicles. This is one of the major reasons among people to choose the preferred mode to travel. So, this is also included in the total travel cost component.
- Operations and Maintenance Costs: The cost of car travel should include the insurance, operations and maintenance costs for every trip, though they are not direct costs of the driver, it is a cost added over time. An average of $0.70 per mile is added for every trip as part of the travel cost for cars.

- *<u>BART:</u>* The total travel cost is the sum of ticket fare and the time cost (calculated from the total travel time of the trip and value of time of each income group).

Table 1. Value of Time expressed in $/hr for each income group in California

Income Group Classification	Income Quintile	Mean Annual Income ($)	Value of time for trips ($/hr)
1	**Lowest fifth (Min. Wage)**	11, 034	2.87
2	**Second fifth**	28, 636	7.45
3	**Third fifth (Median)**	49, 309	12.84
4	**Fourth fifth**	79, 040	20.58
5	**Highest fifth**	169, 633	44.17
6	**Top 5 %**	287, 686	74.92

(Source: USDOT. "Departmental Guidance: Valuation of Travel Time in Economic Analysis". Office of the Secretary of Transportation. 2003.)

It has to be noted that the total travel cost for carpooling is the sum of fuel cost, time cost and parking/toll costs, except that the fuel cost and parking & toll costs are shared by three people (i.e. divided by three), and the time cost is the not shared, although, the total driving time in the carpool lane might be shorter, and an 'inconvenience' time is added to the total travel time, explained in detail in the section 6.

3. BART Network

BART network covers the important routes in SF bay area, running almost parallel to all major freeways. Typically the train schedule for a line is every fifteen minutes, in stations where more than one train line runs, the schedule is more frequent, especially in the San Francisco city train stations, one could board a train within five minutes of arrival.

Main points of interest are identified in the BART network map (as shown in Figure 1) that includes all the terminal and transfer stations in order to keep the problem under a workable dimension.

Figure 1. BART Network map with identified project stations

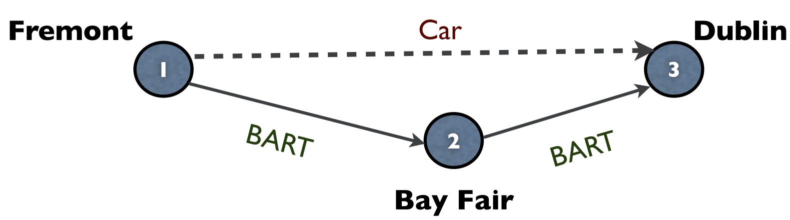
All the links between these stations are bidirectional, and the train timetables for each line are coded for every station included in the project. This is a very important step in order to calculate the waiting time and transfer time between the origin and destination nodes.

4. Modal Choice: Schematic Representation

This section explains the modal choice competition problem between driving and taking BART for a simple origin-destination problem through a schematic representation. Here the origin is Fremont, and the destination is Dublin.

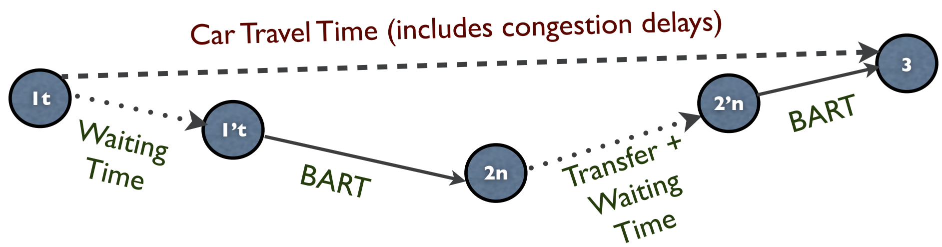
- **Step 1**: In the first step, the paths are identified between the origin and destination that would minimize the total travel time for each mode. As shown in Figure 2, if one is driving, he could reach Dublin from Fremont directly. However, if one is taking train, he should know the 'shortest path' to reach Dublin, in this case, there is no direct train, so he has to transfer at Bay Fair station to reach the destination.

Figure 2. Minimizing total travel distance for each mode



- **Step 2:** Once the shortest path is identified in Step 1, the total travel cost is calculated for each mode and is compared to obtain the optimal choice. Now, in order to compare the travel cost, one has to know the total travel time taken by each mode from origin to destination. For driving, this time includes all the congestion delays caused from Origin to Destination. The path is modified for the train network, by an additional node for each station from Origin until it reaches its destination (as shown in Figure 3). The lengths of the new nodes added vary with the arrival time of the user at the Origin station, the waiting time is determined from the timetable of the station coded at each station.

Figure 3. Introduction of additional nodes in the selected BART path



Where, t = arrival time at the station
n = t + (BART travel time between 1 and 2)

The waiting time is assigned to the length of the link 1_t-$1'_t$. Based on the timetable of the train at node 1, waiting time link length is the difference between the departure time of the train and the arrival time of the user. Once the user reaches node 2 (i.e. Bay Fair station), the waiting time link length 2_n-$2'_n$ is calculated by the timetable at node 2.

For example, if the user arrived at the Fremont station at 7:05 AM, and the trains run every 15 minutes from 7:00 AM, the length of the link 1_t-$1'_t$ is 10 minutes. This adds up the total travel time, and depending on when the train departs from station 2, the delay of transfer link 2_n-$2'_n$ varies when the user arrives at the station.

5. Problem Formulation

As explained in the section 2, this project is a bi-objective problem. The objective function of the problem is given by,

$$\text{Minimize} \quad A_m + \Sigma\, C_m$$

Where, A_m = total distance traveled for mode 'm'
C_m = total travel cost for mode 'm' (individual components of the cost are explained below)
m = mode of choice [BART, Car, Carpool]

Step 1:

Calculate 'A_m' for each mode. In this case, driving alone and carpooling drive the same distance, so 'Car' is used a common term. The distance traveled by car for any point from origin to destination is given in the problem. The shortest paths (that might include transfers) are determined using Bellman's shortest path problem for the BART system.

Let 's' be the destination node and $p_{i,j}$ be the distance between the BART nodes i and j.

The function that determines shortest path length,

$$u_j = Min\left\{u_k + p_{k,j}\right\}, j \neq s$$

Boundary Condition: $u_s = 0$

A_B = shortest path that is obtained from this optimization problem.

The length of A_B is the minimum travel distance between the origin and destination points.

Step 2:

The next step is to calculate the total travel cost for each mode and select the mode that minimizes both distance and cost. Some of the important definition terms are mentioned below:

B = BART; C = Car; P = Carpool

A_B = shortest path through BART (obtained in Step 1)

A_C = shortest path through Car (obtained in Step 1)

T_{ijB} = Travel time through BART (excluding the waiting time) from i to j

d_{ij} = Waiting time at station i to board train to station j

E_i = Arrival time at station i

D_{ij} = Departure time of the next train at station i to station j

T_{ijC} = Travel time through Car from i to j (includes congestion delays)

T_{ijP} = Travel time through Car from i to j (includes congestion delays)

α_B = Ticket fare for the path A_B

α_C = Fuel Cost for the path A_C (mileage assumed to be 35 mpg)

β_C = Other costs for driving alone for the path A_C (such as parking, toll)

β_P= Other costs for carpooling for the path A_C (such as parking)
$V(\gamma)$ = Value of time of income group γ (refer Table 1)
TOT_m = Total Travel time for $\forall$ m = [B, C, P]

In order to arrive at the optimal modal choice matrix between two points, for different income groups for different starting points of travel, the following algorithm is used.

Initialization:

t = 0 to 59 (minutes in an hour)
$R_N(t) = 0$ (initialize result matrix)

for t = 0 to 59, do
    $TOT_m = 0 \; \forall$ m = [B, C, P]

    $E_i = t$
    for all link ij $\in A_B$, do
        $d_{ij} = D_{ij} - Ei$
        $TOT_B = D_{ij} + T_{ijB} + TOT_B$
        $E_i = TOT_B$

    for all link ij $\in A_C$, do
        $TOT_C = T_{ijC}$
        $TOT_P = T_{ijP}$

    for all N $\in \gamma$, do

$$\pi = \underset{Q}{Min} \begin{cases} Q_B = \alpha_B + \left(TOT_B * V(N)\right) \\ Q_C = \alpha_C + \beta_C + \left(TOT_C * V(N)\right) \\ Q_P = \alpha_C + \beta_P + \left(TOT_P * V(N)\right) \end{cases}$$

        if $\pi = Q_B$, then $R_N(t)$ = "BART"
        if $\pi = Q_C$, then $R_N(t)$ = "CAR"
        if $\pi = Q_P$, then $R_N(t)$ = "CARPOOL"

6. Data Collection and Assumptions

The data needed to run this model are mainly collected from two resources: Google maps, and the BART website. For the car travel between two points, that includes traffic congestion delays, Google maps' real-time travel data was used to determine the total travel time. For the transit data, BART website was extensively used to collect all the station timetables (which were coded for each station in the program), and the train travel time between two points. For the carpool data, non-peak hour travel time was used from Google maps. The operations and maintenance costs of driving are calculated from the www.commutesolutions.org website.

The following are the important assumptions made in the model:

- The walking times at origin and destination are ignored for all the modes.
- Multiple modes for each trip are not considered in this problem.
- The origin and destination points for driving coincide with the corresponding BART station locations.
- The average parking rate at the City of San Francisco is assumed to be $15/day, and for the City of Oakland, it is assumed to be $12/day.
- Toll fee of $5.00 is included for the cars coming from East Bay to San Francisco, and for carpooling vehicles, the toll fee is $2.50.
- The price of gasoline is assumed to $4.00/gallon, which is the average current price in the SF bay area.
- An inconvenience time cost is assigned for carpooling vehicles, it is assumed be 15% of the car travel time, and it increases by the same amount for each income group (in the increasing classification fashion).
- BART paths are designed to choose minimum number of transfers. In cases where there is more than one transfer, preference is given to minimizing the number of transfers, followed by the travel time.
- In order to calculate the operation and maintenance costs of driving, an average of 30 miles per trip is assumed.
- Indirect costs of driving to calculate 'Carbon Tax' (in the scenario III in section 7), includes costs attributing to accidents, construction, air pollution damage, road noise, CO2 reduction, water pollution, transportation diversity and equity, land use impact, congestion, and roadway land value. This is calculated from the www.commutesolutions.org website.

7. Scenarios and Results

Optimal mode is chosen for various starting times of a peak hour and different income groups. The output of the model is the optimal mode choice matrix between the two parameters. The following scenarios are run through the model to analyze how the optimal mode choice results would vary:

- Long distance vs. Short distance trips: The distance of the trip plays a very important role in the modal choice selection. This scenario will analyze the differences between these two trips (excluding SF as destination).
- Fuel Economy of cars: This scenario will analyze whether possessing a car with a higher mileage for gasoline (such as Toyota Prius) will change the way the optimal modal choice matrix is generated.
- Equity discussions: This scenario will discuss two types of equity problems. One, what happens when public transit is subsidized to travel, and another, what happens when a carbon tax is introduced for driving.

For all the scenarios, the result matrix is constructed for the first fifteen minutes in an hour of the peak time period (as the train schedules are cyclical every 15 minutes and the results are the same).

SCENARIO I: LONG TRIPS Vs. SHORT TRIPS

The short trips in this project are typically characterized as trips of distance less than 15 miles or less, and the long trips are characterized as trips greater than 15 mile distance.

The trips can be divided into four groups, short distance trips, long distance trips, short distance trips to SF, and long distance trips to SF. The trips to SF are put in a separate group as the city has high parking costs and toll costs across the bridge, and it influences the user to choose modes in a different way than the other trip choices.

The following paths are considered that could delve into the four groups discussed above:

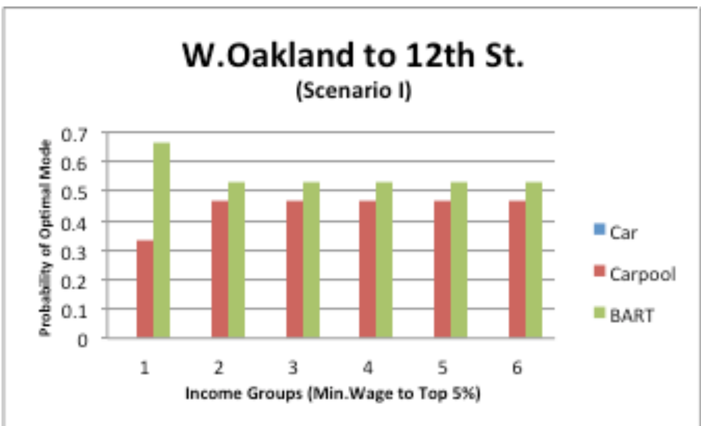
⇒ Group 1: Short distance trip→West Oakland to 12th Street (Oakland)
⇒ Group 2: Long distance trip → Richmond to Dublin
⇒ Group 3: Short distance trip (SF destination) → 12th Street(Oakland) to SF

⇒ Group 4: Long distance trip (SF destination) → Fremont to SF

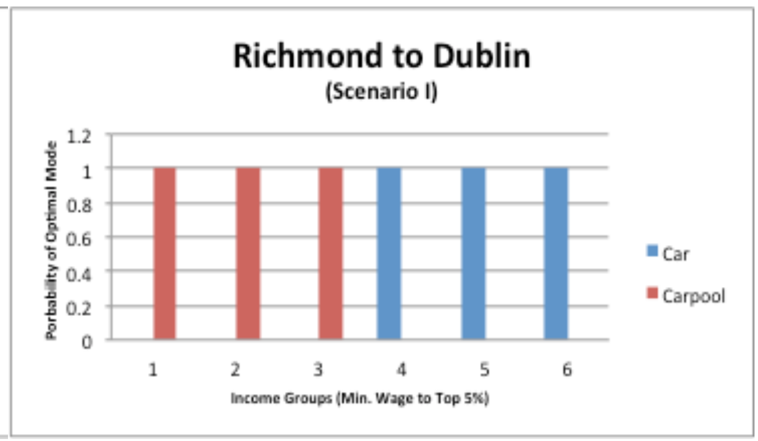
The income groups are divided into 6 classifications (shown in Table 1) ranging from Minimum wage income to top 5% income population. Figure 4 shows the probability of each mode being the optimal mode for the user in a peak hour period depending on what income group the user is in.

Figure 4. Probability of a mode being the optimal choice for short and long distance trips

Group 1: Short distance trip

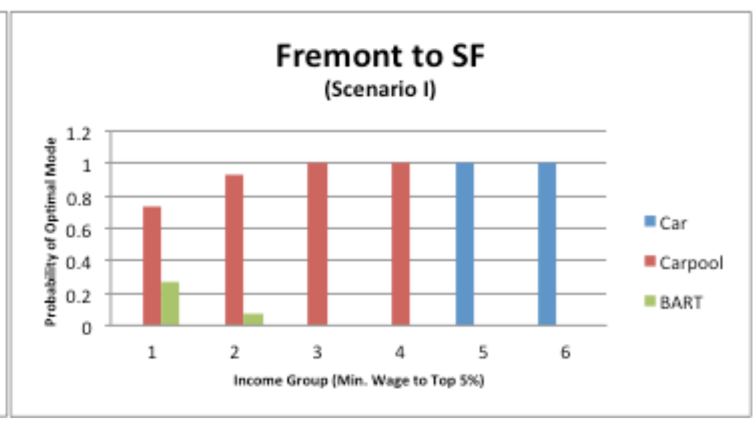


Group 2: Long distance trip



Group 3: Short distance trip to SF

Group 4: Long distance trip to SF



From the model results, most of the short trips are dominated by BART and Carpool mode choices across the income groups, and the long trips are dominated by carpool modes for lower income groups and alone-driving for higher income groups. Moreover, among the trips directed to San Francisco, BART dominates almost all of the short trips, and the long trips are a combination of BART/Carpool for lower income groups, and Carpool/Car for higher income groups.

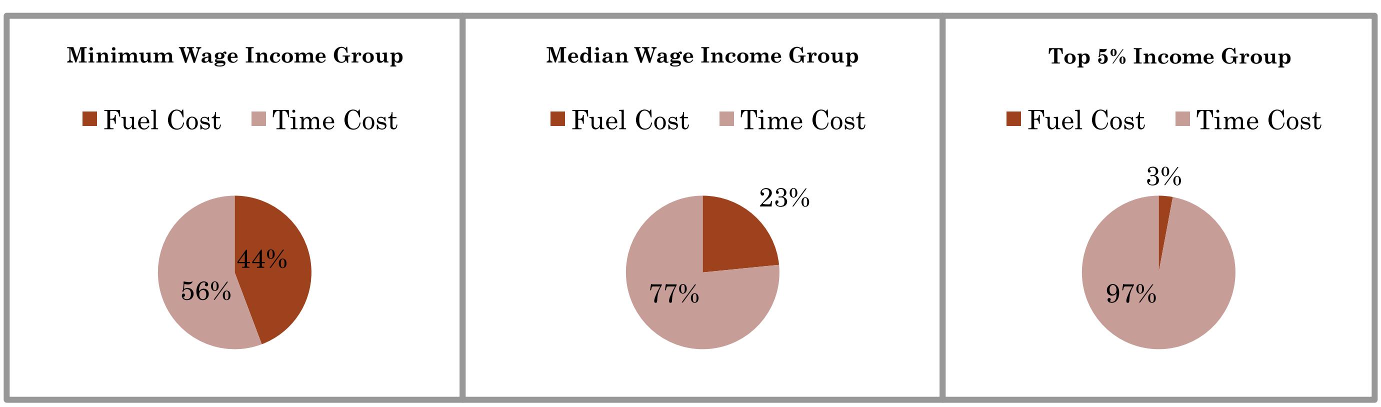
SCENARIO II: FUEL ECONOMY OF CARS

The model currently assumes 30 mpg as the average fuel economy of car driven in bay area (it is a typical fuel economy of a four-seated sedan such as Honda Accord). Many hybrid cars in the market have a much higher mileage than this. For example, EPA estimates that the fuel economy of Toyota Prius can reach up to 50 mpg. This scenario ran the model for the four different path groups (discussed in scenario 1) to check if it has any significant effect on the mode choice selection of the user. It was observed that the variation in optimal mode choice due to driving a high fuel economy car is minimal in this system. This is mainly due to the distribution of time cost and fuel cost in the total travel cost.

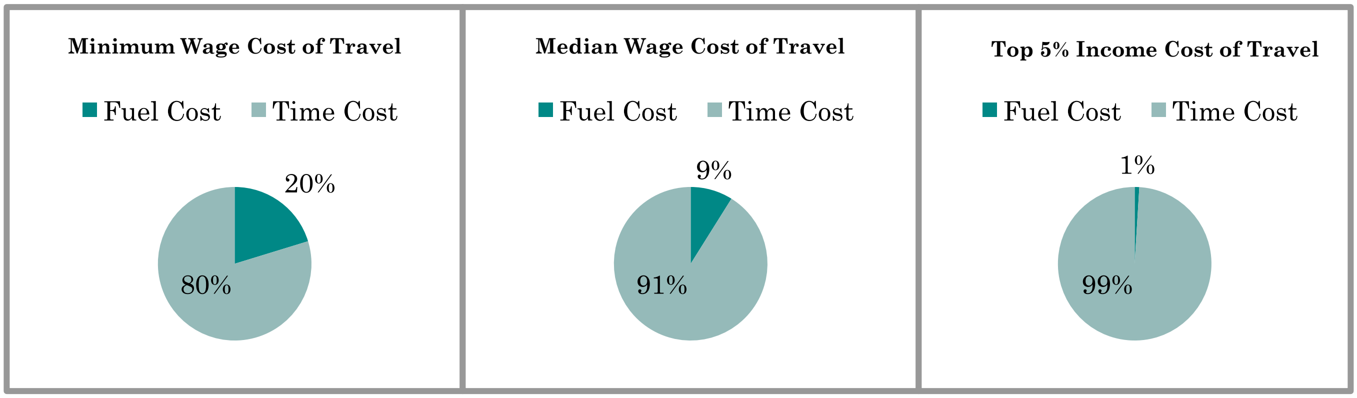
Even among the minimum wage group, who are perceived to have the lowest value of time, have a major part of their travel cost as 'time cost'. Figure 5 shows the differences in share of time cost and fuel cost (driving alone) for long and short distance trips for three chosen income groups: Minimum wage, Median wage and Top 5%. It is observed that the share of time cost has a very important role especially when the person's value of time increases. Therefore, it can be concluded that fuel economy does not play a very important role when it comes to modal choice while competing with BART system.

Figure 5. Total Travel Cost distribution for short and long distance trips

Short distance trip (West Oakland to 12th Street)



<u>*Long distance trip (Richmond to Dublin)*</u>



SCENARIO III: CARBON TAX AND 'GREEN' INCENTIVES

This scenario analyzes the output of modal choices under two policy initiatives: carbon tax, and providing incentives to ride public transportation.

Under carbon tax, additional costs are added to the cost of driving per mile, that includes the carbon tax of about $ 0.7 per mile. The paths in the four groups discussed in scenario I are run in the model. It is observed that including the carbon tax in the cost does not have an effect in the short distance trips, but has an effect on long distance trips. More 'carpool' and BART options are chosen as optimal choices under this policy. Figure 6 shows the probability of the mode being an optimal choice in a peak hour under the carbon tax scenario.

Under the 'green incentive' policy, the public transit ticket fares are subsidized completely. So, the user can practically receive refunds for using BART. Including this policy in the model does not have major changes in the optimal modal choices in the long distance trips, though there are a few BART trips chosen in the short distance trips. On the whole, there isn't much difference in the choice of modes. Hence, it can be concluded that under this policy, the tax policy works much better than incentive policy.

Figure 6. Probability of a mode being the optimal choice long distance trips Comparison of 'Carbon Tax' scenario with Scenario I

8. Summary and Conclusions

This model takes in two main objectives to minimize: travel distance and travel cost, for the SF bay area, to find the optimal modes between BART, carpooling and driving alone. Shortest paths are found in the BART network based on distance as the first step. Then those paths are used to calculate the total travel cost, that includes fuel cost, parking and toll costs, operating costs for cars, ticket fare for transit and time cost for both modes based on the value of time perceived by the different income groups.

The model is programmed in Python language (attached in Appendix). Various scenarios are run in the model to analyze the outcome of the optimal model choice decisions.

It is observed that, BART is chosen as the preferred choice is most short distance trips, followed by carpool among all the income groups. For long distance trips, the choices are split between driving alone and carpooling, the former preferred by higher income groups. It is also noted that the fuel economy of cars driven by the users have little to no effect on the modal choice outcomes as the ‘time cost’ forms the major part of the total travel cost. Also, when compared between carbon tax and subsidizing transit, it is observed that the tax policy has an effect in long distance trips that pushes choice of BART and carpool mode choices in the matrix. Subsidizing transit again has little to no effect on the mode choice decisions of the user, since time cost dominates the total cost in the long distance trips, and the short distance trips already have BART as the optimal mode choice for most paths.

Overall, it is a very interesting project to study the tradeoff between the travel cost and travel time from the perspective of the user. This project currently focuses on single mode choice for the whole trip. In real life, people take more than one mode to travel, park and ride, for example. Expansion of this project would include other modes such as, Muni, Alameda Transit in the mix, and especially it would be a great topic of interest to look at multi-modal transit competitions.

9. References

"Bay Area Rapid Transit." www.bart.gov (accessed June 12, 2012).

"Google Maps." Google. http://maps.google.com (accessed June 8, 2012).

"The True Cost of Driving." Commute Solutions. http://commutesolutions.org/external/calc.html (accessed June 10, 2012).

Bérube, Jean-François , Jean-Yves Potvin and Jean Vaucher. "Time-dependent shortest paths through a fixed sequence of nodes: application to a travel planning problem." Computers & Operations Research 33 (2006): 1838–1856.

Aifadopoulou, Georgia, Athanasios Ziliaskopoulos and Evangelia Chrisohoou. "Multiobjective Optimum Path Algorithm for Passenger Pretrip Planning in Multimodal Transportation Networks." Journal of the Transportation Research Board 2032 (2007): 26–34.

Bielli, Maurizio, Azedine Boulmakoul and Hicham Mouncif. "Object modeling and path computation for multimodal travel systems." European Journal of Operational Research 175 (2006): 1705–1730.

Pallottino, Stefano, and Maria Scutella. *Shortest Path Algorithms in Transportation models: classical and innovative aspects*. Pisa, Italy: University of Pisa, Department of Computer Science, 1997.

Ahuja, Ravindra , James Orlin, Stefano Pallottino, and Maria Scutella. *Dynamic Shortest Paths Minimizing Travel Times and Costs*. Cambridge, MA: Sloan School of Management, Massachusetts Institute of Technology, 2002.

"US EPA Fuel Economy Leaders." Environmental Protection Agency. http://www.epa.gov/fueleconomy/overall-high.htm (accessed June 12, 2012).

USDOT. *"Departmental Guidance: Valuation of Travel Time in Economic Analysis"*. Office of the Secretary of Transportation. 2003.

Appendix: Python Code

```python
# Import required modules
import networkx as nx
import numpy

O = raw_input("Enter Origin Station Code")
D = raw_input("Enter Destination Station Code")
r = numpy.zeros(shape=(15,6))

#Define the networks
X = nx.Graph()
C = nx.Graph()
BTICKT = nx.Graph()

#BART network--distance and time
X.add_weighted_edges_from([('FR', 'BF', {'dist':16, 'time':18}), ('FR', 'WO', {'dist':27, 'time':38}), ('FR', 'EM', {'dist':35, 'time':46}), ('FR', 'DC', {'dist':43, 'time':63}), ('FR', '12', {'dist':26, 'time':36}), ('FR', 'RC', {'dist':38, 'time':62}), ('DU', 'BF', {'dist':12, 'time':17}), ('DU', 'WO', {'dist':26, 'time':38}), ('DU', 'EM', {'dist':34, 'time':45}), ('DU', 'DC', {'dist':42, 'time':63}), ('RC', '12', {'dist':12, 'time':24}), ('RC', 'WO', {'dist':12, 'time':28}), ('RC', 'EM', {'dist':20, 'time':35}), ('RC', 'DC', {'dist':28, 'time':53}), ('PB', '12', {'dist':32, 'time':41}), ('PB', 'WO', {'dist':33, 'time':46}), ('PB', 'EM', {'dist':41, 'time':53}), ('PB', 'DC', {'dist':49, 'time':60}), ('DC', 'EM', {'dist':8, 'time':17}), ('DC', 'WO', {'dist':16, 'time':24}), ('DC', 'BF', {'dist':27, 'time':44}), ('DC', '12', {'dist':17, 'time':27}), ('BF', 'WO', {'dist':11, 'time':20}), ('BF', 'EM', {'dist':19, 'time':27}), ('BF', '12', {'dist':10, 'time':18}), ('BF', 'RC', {'dist':22, 'time':43}), ('WO', 'EM', {'dist':8, 'time':7}), ('WO', '12', {'dist':1, 'time':3}),('EM', '12', {'dist':9, 'time':10})])

#Car network--distance and time
C.add_weighted_edges_from([('FR', 'BF', {'dist':16, 'time':24}), ('FR', 'WO', {'dist':27, 'time':34}), ('FR', 'EM', {'dist':35, 'time':43}), ('FR', 'DC', {'dist':43, 'time':47}), ('FR', '12', {'dist':26, 'time':32}), ('FR', 'RC', {'dist':38, 'time':43}), ('FR', 'DU', {'dist':20, 'time':25}), ('FR', 'PB', {'dist':52, 'time':58}), ('DC', 'EM', {'dist':11, 'time':15}), ('DC', 'PB', {'dist':48, 'time':54}), ('DC', 'RC', {'dist':25, 'time':32}), ('DC', '12', {'dist':19, 'time':26}), ('DC', 'WO', {'dist':17, 'time':23}), ('DC', 'BF', {'dist':31, 'time':39}), ('DC', 'DU', {'dist':42, 'time':47}), ('DU', 'BF', {'dist':13, 'time':17}), ('DU', 'EM', {'dist':33, 'time':42}), ('DU', 'WO', {'dist':26, 'time':33}), ('DU', '12', {'dist':25, 'time':30}), ('DU', 'RC', {'dist':35, 'time':40}), ('DU', 'PB', {'dist':34, 'time':38}), ('EM', 'WO', {'dist':8, 'time':18}), ('EM', '12', {'dist':11, 'time':21}), ('EM', 'RC', {'dist':16, 'time':26}), ('EM', 'PB', {'dist':39, 'time':48}), ('EM', 'BF', {'dist':23, 'time':33}), ('PB', 'BF', {'dist':40, 'time':46}), ('PB', 'WO', {'dist':39, 'time':40}), ('PB', '12', {'dist':39, 'time':36}), ('PB', 'RC', {'dist':32, 'time':38}), ('RC', 'WO', {'dist':10, 'time':16}), ('RC', '12', {'dist':12, 'time':18}), ('RC', 'BF', {'dist':24, 'time':31}), ('WO', '12', {'dist':2, 'time':6}), ('WO', 'BF', {'dist':15, 'time':22}), ('BF', '12', {'dist':15, 'time':19})])

#BART ticket cost for each O-D path
BTICKT.add_weighted_edges_from([('FR', 'BF', 1.75), ('FR', 'WO', 4.10), ('FR', 'EM', 5.60), ('FR', 'DC', 6.00), ('FR', '12', 4.00), ('FR', 'RC', 4.85), ('FR', 'DU', 4.35), ('FR', 'PB', 6.40), ('DC', 'EM', 2.95), ('DC', 'PB', 6.35), ('DC', 'RC', 4.65), ('DC', '12', 3.80), ('DC', 'WO', 3.75), ('DC', 'BF', 4.75), ('DC', 'DU', 6.00), ('DU', 'BF', 1.75), ('DU', 'EM', 5.55), ('DU', 'WO', 4.10), ('DU', '12', 4.00), ('DU', 'RC', 4.85), ('DU', 'PB', 6.35), ('EM', 'WO', 2.90), ('EM', '12', 3.10), ('EM', 'RC', 4.25), ('EM', 'PB', 5.95), ('EM', 'BF', 4.30), ('PB', 'BF', 5.10), ('PB', 'WO', 4.45), ('PB', '12', 4.30), ('PB', 'RC', 4.90), ('RC', 'WO', 2.75), ('RC', '12', 2.60), ('RC', 'BF', 3.60), ('WO', '12', 1.75), ('WO', 'BF', 2.75), ('BF', '12', 2.60)])

#Function to calculate distances between the two points
def original_path(origin, destination):
    w = nx.shortest_path(X, origin, destination, weight='dist')
    distance = []
    for wnode in range(len(w)-1):
        distance.append(X.edge[w[wnode]][w[wnode+1]]['weight']['time'])
    return distance

```

```python

#This function adds additional nodes to the selected BART path
def find_path(origin,destination):
    l = nx.shortest_path(X, origin, destination, weight='dist')
    cl = nx.shortest_path(C, origin, destination, weight='dist')
    #Create extra nodes in between
    for m in range(len(l)-1):
        a = " "
        n = m+1
        a = str(l[m]) + str(l[n])
        if len(l) < 5:
            X.add_edge(l[m], a, {'time':0})
            X.add_edge(a, l[n], {'time':0})
            if X.has_edge(l[m], l[n]):
                X.remove_edge(l[m], l[n])
    return nx.shortest_path(X, origin, destination, weight='time')

pdistance = original_path(O, D)
ppath = find_path(O, D)

if len(ppath) == 3:
    if ppath[0] == 'DC': # Daly City ***********************************************************
        if ppath[2] == 'EM' or ppath[2] == 'WO':
            X.node['DC']['stime'] = 1
            dclist = []
            for t in range(1,60,15):
                newt = t
                dclist.append(newt)
                newt = newt + 5
                dclist.append(newt)
                newt = newt + 3
                dclist.append(newt)
                newt = newt+ 4
                dclist.append(newt)
        if ppath[2] == 'DU':
            X.node['DC']['stime'] = 6
            dclist = []
            for t in range(6,60,15):
                dclist.append(t)
        if ppath[2] == 'FR':
            X.node['DC']['stime'] = 13
            dclist=[]
            for t in range(13,60,15):
                dclist.append(t)
        if ppath[2] == 'PB':
            X.node['DC']['stime'] = 9
            dclist=[]
            for t in range(9,60,15):
                dclist.append(t)
        if ppath[2] == 'RC':
            X.node['DC']['stime'] = 1
            dclist = []
            for t in range(1,60,15):
                dclist.append(t)
        if ppath[2] == '12':
            X.node['DC']['stime']=1
            dclist = []
            for t in range(1,60,15):
                newt = t
                dclist.append(newt)
                newt = newt + 8
                dclist.append(newt)
        if ppath[2] == 'BF':
            X.node['DC']['stime'] = 6
```

```
            dclist = []
            for t in range(6,60,15):
                newt = t
                dclist.append(newt)
                newt = newt + 7
                dclist.append(newt)

    if ppath[0] == 'EM': # Embarcadero ********************************************************
        if ppath[2] == 'DC':
            X.node['EM']['stime'] = 2
            emlist = []
            for t in range(2,60,15):
                newt = t
                emlist.append(newt)
                newt = newt + 3
                emlist.append(newt)
                newt = newt + 2
                emlist.append(newt)
                newt = newt + 6
                emlist.append(newt)
        if ppath[2] == 'WO':
            X.node['EM']['stime'] = 0
            emlist = []
            for t in range(0,60,15):
                newt = t
                emlist.append(newt)
                newt = newt + 3
                emlist.append(newt)
                newt = newt + 5
                emlist.append(newt)
                newt = newt + 3
                emlist.append(newt)
        if ppath[2] == 'DU':
            X.node['EM']['stime'] = 9
            emlist = []
            for t in range(9,60,15):
                emlist.append(t)
        if ppath[2] == 'FR':
            X.node['EM']['stime'] = 0
            emlist = []
            for t in range(0,60,15):
                emlist.append(t)
        if ppath[2] == 'PB':
            X.node['EM']['stime'] = 11
            emlist = []
            for t in range(11,60,15):
                emlist.append(t)
        if ppath[2] == 'RC':
            X.node['EM']['stime'] = 3
            emlist = []
            for t in range(3,60,15):
                emlist.append(t)
        if ppath[2] == 'BF':
            X.node['EM']['stime'] = 0
            emlist = []
            for t in range(0,60,15):
                newt = t
                emlist.append(newt)
                newt = newt + 8
                emlist.append(newt)
        if ppath[2] == '12':
            X.node['EM']['stime'] = 3
            emlist = []
            for t in range(3,60,15):
```

```
                newt = t
                emlist.append(newt)
                newt = newt + 8
                emlist.append(newt)

    if ppath[0] == 'WO': # West Oakland *************************************************
        if ppath[2] == 'DC' or ppath[2] == 'EM':
            X.node['WO']['stime'] = 6
            wolist = []
            for t in range(6,60,15):
                newt = t
                wolist.append(newt)
                newt = newt + 4
                wolist.append(newt)
                newt = newt + 3
                wolist.append(newt)
                newt = newt + 1
                wolist.append(newt)
        if ppath[2] == 'BF':
            X.node['WO']['stime'] = 0
            wolist = []
            for t in range(0,60,15):
                newt = t
                wolist.append(newt)
                newt = newt + 7
                wolist.append(newt)
        if ppath[2] == 'FR':
            X.node['WO']['stime'] = 7
            wolist = []
            for t in range(7,60,15):
                wolist.append(t)

        if ppath[2] == 'DU':
            X.node['DU']['stime'] = 0
            wolist = []
            for t in range(0,60,15):
                wolist.append(t)
        if ppath[2] == '12':
            X.node['WO']['stime'] = 3
            wolist = []
            for t in range(3,60,15):
                newt = t
                wolist.append(newt)
                newt = newt + 7
                wolist.append(newt)
        if ppath[2] == 'RC':
            X.node['WO']['stime'] = 10
            wolist = []
            for t in range(10,60,15):
                wolist.append(t)
        if ppath[2] == 'PB':
            X.node['WO']['stime'] = 3
            wolist = []
            for t in range(3,60,15):
                wolist.append(t)

    if ppath[0] == '12': #12th Street Oakland
*************************************************
        if ppath[2] == 'BF' or ppath[2] == 'FR':
            X.node['12']['stime'] = 0
            twlist = []
            for t in range(0,60,15):
                twlist.append(t)
        if ppath[2] == 'DC' or ppath[2] == 'EM' or ppath[2] == 'WO':
            X.node['12']['stime'] = 4
```

```
            twlist = []
            for t in range(6,60,15):
                if t == 6:
                    twlist.append(t)
                    newt = t
                newt = newt + 2
                twlist.append(newt)
                newt = newt + 13
                twlist.append(newt)
        if ppath[2] == 'PB':
            X.node['12']['stime'] = 6
            twlist = []
            for t in range(6,60,15):
                twlist.append(t)
        if ppath[2] == 'RC':
            X.node['12']['stime'] = 4
            twlist = []
            for t in range(6,60,15):
                if t == 6:
                    twlist.append(t)
                    newt = t
                newt = newt + 7
                twlist.append(newt)
                newt = newt + 8
                twlist.append(newt)
    if ppath[0] == 'PB': #Pittsburg
**********************************************************
        pblist = []
        if ppath[2] == 'DC' or ppath[2] == '12' or ppath[2] == 'EM' or ppath[2] == 'WO':
            X.node['PB']['start'] = 2
            X.node['PB']['interval'] = 15
            #pblist = []
            for t in range(2,60,15):
                pblist.append(t)
    if ppath[0] == 'DU': #Dublin
*************************************************************
        dulist = []
        if ppath[2] == 'DC' or ppath[2] == 'BF' or ppath[2] == 'WO' or ppath[2] == 'EM':
            X.node['DU']['stime'] = 13
            X.node['DU']['interval'] = 15
            for t in range(13,60,15):
                dulist.append(t)
    if ppath[0] == 'FR': #Fremont Station
****************************************************
        if ppath[2] == 'DC' or ppath[2] == 'EM' or ppath[2] == 'WO':
            X.node['FR']['stime'] = 6
            X.node['FR']['interval'] = 15
            frlist = []
            for t in range(6, 60, 15):
                frlist.append(t)
        if ppath[2] == 'RC' or ppath[2] == '12':
            X.node['FR']['stime'] = 0
            X.node['FR']['interval'] = 15
            frlist = []
            for t in range(0,60,15):
                frlist.append(t)
        if ppath[2] == 'BF':
            X.node['FR']['stime'] = 0
            frlist = []
            for t in range(0,60,15):
                if t == 0:
                    frlist.append(t)
                    newt = t
                newt = newt + 6
                frlist.append(newt)
```

```
                    newt = newt + 9
                    frlist.append(newt)
        if ppath[0] == 'RC': #Richmond Station ************************************************
            if ppath[2] == 'FR' or ppath[2] == 'BF':
                X.node['RC']['stime'] = 5
                X.node['RC']['interval'] = 15
                rclist = []
                for t in range(5,60,15):
                    rclist.append(t)
            if ppath[2] == 'DC' or ppath[2] == 'EM' or ppath[2] == 'WO':
                X.node['RC']['stime'] = 12
                X.node['RC']['interval'] = 15
                rclist = []
                for t in range(12,60,15):
                    rclist.append(t)
            if ppath[2] == '12':
                X.node['RC']['stime'] = 5
                rclist = []
                for t in range(0,60,15):
                    if t == 5:
                        rclist.append(t)
                        newt = t
                    newt = newt + 7
                    rclist.append(newt)
                    newt = newt + 3
                    rclist.append(newt)
        if ppath[0] == 'BF': #Bay Fair ********************************************************
            if ppath[2] == 'DU':
                X.node['BF']['stime'] = 5
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)
            if ppath[2] == 'FR':
                X.node['BF']['stime'] = 3
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)
            if ppath[2] == 'RC' or ppath[2] == '12':
                X.node['BF']['stime'] = 3
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)
            if ppath[2] == 'DC' or ppath[2] == 'EM' or ppath[2] == 'WO':
                X.node['BF']['stime'] = 0
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)

    if len(ppath) > 4:
        if ppath[0] == 'RC': #Richmond*********************************************************
            if ppath[2] == '12':
                X.node['RC']['stime'] = 5
                rclist = []
                for t in range(0,60,15):
                    if t == 5:
                        rclist.append(t)
                        newt = t
                    newt = newt + 7
                    rclist.append(newt)
                    newt = newt + 3
                    rclist.append(newt)
```

```
            if ppath[4] == 'PB':
                X.node['12']['stime'] = 6
                X.node['12']['interval'] = 15
                twlist = []
                for t in range(6,60,15):
                    twlist.append(t)
        if ppath[2] == 'BF':
            X.node['RC']['stime'] = 5
            X.node['RC']['interval'] = 15
            rclist = []
            for t in range(5,60,15):
                rclist.append(t)
            if ppath[4] == 'DU':
                X.node['BF']['stime'] = 5
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)

    if ppath[0] == 'PB': #Pittsburg
**********************************************************
        pblist = []
        if ppath[2] == '12':
            X.node['PB']['stime'] = 2
            #pblist = []
            for t in range(2,60,15):
                pblist.append(t)
            if ppath[4] == 'RC':
                X.node['12']['stime'] = 4
                twlist = []
                for t in range(6,60,15):
                    newt = t
                    twlist.append(newt)
                    newt = newt + 7
                    twlist.append(newt)
            if ppath[4] == 'FR' or ppath[4] == 'BF':
                X.node['12']['stime'] = 0
                twlist = []
                for t in range(0,60,15):
                    twlist.append(t)
                if len(ppath) > 5:
                    if ppath[6] == 'DU':
                        X.node['BF']['stime'] == 5
                        bflist = []
                        for t in range(5,60,15):
                            bflist.append(t)


    if ppath[0] == 'BF': #Bay Fair
************************************************************
        if ppath[2] == '12':
            X.node['BF']['stime'] = 3
            bflist = []
            for t in range(3,60,15):
                bflist.append(t)
            if ppath[4] == 'PB':
                X.node['12']['stime'] = 6
                twlist = []
                for t in range(6,60,15):
                    twlist.append(t)

    if ppath[0] == 'DU': # Dublin
*************************************************************
        dulist = []
        if ppath[2] == 'BF':
```

```
            X.node['DU']['stime'] = 13
            for t in range(13,60,15):
                dulist.append(t)
            if ppath[4] == 'RC':
                X.node['BF']['stime'] = 3
                bflist = []
                for t in range(3,60,15):
                    bflist.append(t)
            if ppath[4] == 'FR':
                X.node['BF']['stime'] = 3
                bflist = []
                for t in range(3,60,15):
                    bflist.append(t)
            if ppath[4] == '12':
                X.node['BF']['stime'] = 3
                bflist = []
                for t in range(3,60,15):
                    bflist.append(t)
                if len(ppath) > 5:
                    if ppath[6] == 'PB':
                        X.node['12']['stime'] = 6
                        twlist = []
                        for t in range(6,60,15):
                            twlist.append(t)

    if ppath[0] == 'FR':
#Fremont**************************************************************
        if ppath[2] == 'BF':
            X.node['FR']['stime'] = 0
            frlist = []
            for t in range(0,60,15):
                if t == 0:
                    frlist.append(t)
                    newt = t
                newt = newt + 6
                frlist.append(newt)
                newt = newt + 9
                frlist.append(newt)
            if ppath[4] == 'DU':
                X.node['BF']['stime'] = 5
                X.node['BF']['interval'] = 15
                bflist = []
                for t in range(5,60,15):
                    bflist.append(t)
        if ppath[2] == 'WO':
            X.node['FR']['stime'] = 6
            X.node['FR']['interval'] = 15
            frlist = []
            for t in range(6, 60, 15):
                frlist.append(t)
            if ppath[4] == 'PB':
                X.node['WO']['stime'] = 3
                X.node['WO']['interval'] = 15
                wolist = []
                for t in range(3, 60, 15):
                    wolist.append(t)
        if ppath[2] == '12':
            X.node['FR']['stime'] = 0
            X.node['FR']['interval'] = 15
            frlist = []
            for t in range(0,60,15):
                frlist.append(t)
            if ppath[4] == 'PB':
                X.node['12']['stime'] = 6
                X.node['12']['interval'] = 15
```

```
                    twlist = []
                    for t in range(6, 60, 15):
                        twlist.append(t)

    xlist = []
    ylist = []
    zlist = []
    if ppath[0] == 'FR':
        xlist = frlist
    if ppath[0] == 'RC':
        xlist = rclist
    if ppath[0] == 'DC':
        xlist = dclist
    if ppath[0] == 'EM':
        xlist = emlist
    if ppath[0] == 'WO':
        xlist = wolist
    if ppath[0] == '12':
        xlist = twlist
    if ppath[0] == 'BF':
        xlist = bflist
    if ppath[0] == 'DU':
        xlist = dulist
    if ppath[0] == 'PB':
        xlist = pblist

    if len(ppath) > 3:
        if ppath[2] == 'BF':
            ylist = bflist
        if ppath[2] == '12':
            ylist = twlist
        if ppath[2] == 'WO':
            ylist = wolist

    if len(ppath) > 5:
        if ppath[4] == '12':
            zlist = twlist
        if ppath[4] == 'BF':
            zlist = bflist
        if ppath[4] == 'WO':
            zlist = wolist

    for start in range(0,15,1):
        for time in xlist:
            if start > time:
                pass
            if start <= time:
                delay = time - start
                break

        if len(ppath) > 3:
            midtime = pdistance[0]
            totaldist = delay + midtime
            for time in ylist:
                if midtime > time:
                    pass
                if midtime <= time:
                    delay2 = time - midtime
                    break
        else:
            totaldist = pdistance[0] + delay

        if len(ppath) > 3:
            totaldist = delay2 + totaldist + pdistance[1]

```

```
    cartime = C.edge[O][D]['weight']['time']
    cardist = C.edge[O][D]['weight']['dist']

    vot = numpy.array([2.87, 7.45, 12.84, 20.58, 44.17, 74.92]) #value of time for different income groups
    parking = 15
    toll = 5
    oakpark = 12

    ctime_cost = vot * cartime
    btime_cost = vot * totaldist
    carp_time = numpy.zeros(6)
    cp_cost = (cartime*0.15)
    for i in range(len(ctime_cost)):
        carp_time[i] = ctime_cost[i] + cp_cost
        cp_cost = cp_cost + (cartime*0.15)

    carfuel = ((4.0/30)*cardist) + (cardist*(0.7))
    if D == 'EM':
        carfuel = parking + carfuel
        if O != 'DC':
            carfuel = carfuel + toll

    carp_fuel = (((4.0/30)*cardist) + (cardist*(0.7)))/3
    if D == 'EM':
        carp_fuel = parking/3 + carp_fuel
        if O != 'DC':
            carp_fuel = carp_fuel + 2.50

    if D == '12':
        carfuel = oakpark + carfuel
        carp_fuel = oakpark/3 + carp_fuel


    bart_cost = btime_cost + BTICKT.edge[O][D]['weight']
    car_cost = ctime_cost + carfuel
    carp_cost = carp_time + carp_fuel

    for i in range(len(vot)):
        if car_cost[i] < bart_cost[i] and car_cost[i] < carp_cost[i]:
            r[start,i] = 1
        elif bart_cost[i] < car_cost[i] and bart_cost[i] < carp_cost[i]:
            r[start,i] = 2
        elif carp_cost[i] < car_cost[i] and carp_cost[i] < bart_cost[i]:
            r[start,i] = 3

opmode = []
for col in range(0,15):
    opmode.append([])
    for row in range(0,6):
        if r[col, row] == 1:
            opmode[col].append('CAR')
        if r[col,row] == 2:
            opmode[col].append('BART')
        if r[col,row] == 3:
            opmode[col].append('Carpool')

for col in range(0,15):
    print opmode[col]
```