MATH 381
GROUP 8
TOUR OPTIMIZATION

Planning a concert tour, optimizing for the "goodness" of the cities we play in.

Written by
Ayan Saraf
Oliver Speltz
Zach Shaw
University of Washington
Seattle Campus

Contents

Abstract

In this project, we apply the use of linear optimization a concert tour. We collected demographic data on various cities along the West and East Coast of the United States, created a "goodness" ranking based off of it, and created a model to maximize the "goodness" of the cities our band plays. We formulated our model as a Linear Integer Program (LP) and solved it using LPSolve. Based on the constraints we provided, LPSolve produced a list of the cities we should play shows in. One of the constraints we provided was that we started and finished in our hometown of Seattle, WA. We discovered that distance traveled per day was more influential in our model than our budget. Our biggest variation was applying the same model to cities on the East Coast. This variation produced some intuitive results.

1 Introduction

In our project, we created a weighted graph. The vertices of this graph represented the cities we had choosen as a subset of the cities within the United States (and Canada). The edges were weighed with the distance between each city. Since we had the distance from one city to every other city in the graph, our graph is fully connected which means that every single vertex has an edge to every other vertex within the graph. To solve our model and find the optimal path we should take, we formulated a Linear Program.

2 Background

We chose the topic of concert tour optimization as this allows us to transcend our classroom knowledge of the adjacenecy matrix and the travelling salesman problem along with incorporating our personal touch to the model.

Discussions of different variations of this problem ended up with us deciding to do this in the context of an up-and-coming band. The situation we are envisioning going into this problem is that we are college students who play in a 3 piece band, called *Simplex*, and we want to venture out and play some shows to spread our sound. This led from the initial idea of making a path to travel to each city as part of the tour, to finding the subset of cities we should travel to in a given time period to ensure that the shows get maximum public reach. This helped stem the idea to compare shows in different cities based on the goodness of the cities which we based off of the annual median income, total population, median age, crime rate and the poverty rate.

The motivation to choose the topic was that all three of the group members are interested in rock music and each member has visited pubs and shows of upcoming college bands. The cities were chosen based on the fact that the team members are from the West Coast, hence choosing that as the base case. Apart from that, in lecture the topic of traveling salesman and the adjacency matrix also was of common interest to the group members.

To quantify "goodness", we chose a ranking system to order the importance of each

of the afore mentioned factors to calculate how it will affect the goodness of the cities. As per our data points, we used 22 cities on each coast of the country giving us a comparision of tours as a whole and know which one will be better for the band to perform. The cities were chosen as the more popular towns with a younger crowd population-specifically more college towns. Because we assume that our type of concert will attract a larger audience in these cities.

2.1 Literature Review

Based on different papers written on a similar topic we have seen that the tour band optimization has generally been an extension of the travelling salesman problem. In the paper written by Wu and Takahashi "Travel Route Centered Metro Map Layout"[3], there is a direct correlation about city routes having access points for making an easier travel path. The paper also talks about how distance can be maximized to cover a greater number of metro stations. Our paper talks about the "goodness", a concept we discuss later in the paper. The more interesting paper on this topic is written by Kroon on "Routing Trains Through Railway Stations" where cities are used to find the nature of whether and when they can be included in the train route.

2.2 Data Collection

In order to accurately model our problem, we needed to decide on data to collect. For the "goodness", we sought to collect population[1], median age[2], median income, crime rate, and poverty rate. The U.S. Census Bureau provided this data in abundance.

Once we collected data for "goodness", we had to collect the distances between from one city to each of the the other cities in our graph. In order to do this, we recorded the miles from the optimum route according to Google Maps. Once we had this data, we had to form a matrix. The code contained all of the distances, and used for loops to populate the correct distance in the correct rows and columns[1].

Additionally, to capture the cost of living in a city, hotel price data was collected for each city as well. This was done using the website Hotels.com. Hotel prices can be highly variable depending on season, holidays, events etc. To get a good baseline, we decided to look at the hotel price for a room for 4 in each city, on the same non-holiday date in the summer. Then we averaged the three lowest prices on that date. We felt this was reasonable to the spirit of the problem; we are a 3 piece college band looking to go on a tour during the summer in between school years.

2.3 West Coast Cities

Here is a figure containing the cities on the West Coast we choose to test.

[1]Can be found in appendix A.1

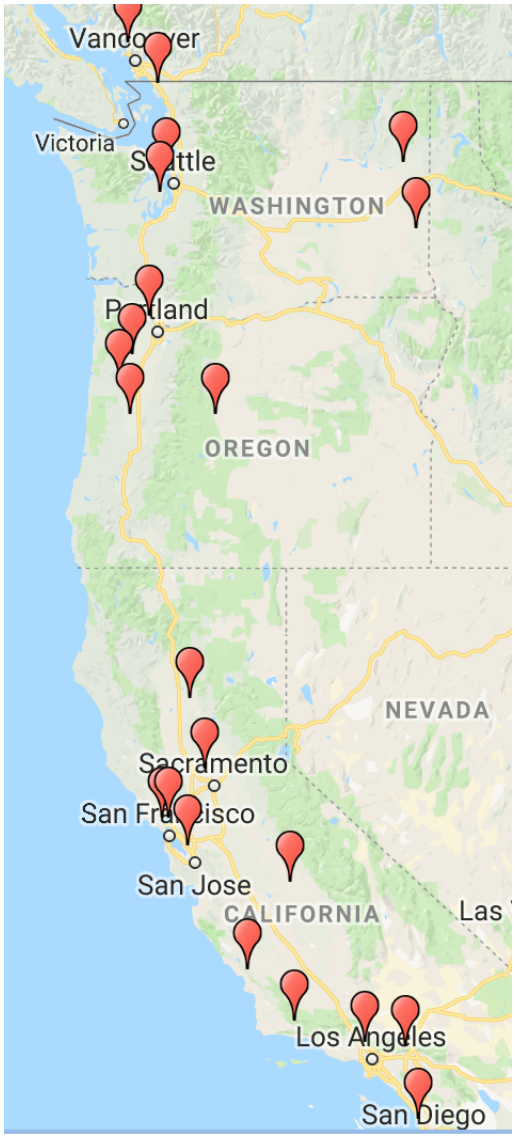

Figure 1: The cities we choose to examine along the West Coast.
In alphabetical order, the cities on the West Coast are Bellingham (WA), Bend (OR), Chico (CA), Corvallis (OR), Eugene (OR), Fresno (CA), Los Angeles (CA), Oakland (CA), Portland (OR), Pullman (WA), Riverside (CA), Sacramento (CA), Salem (OR), San Diego (CA), San Francisco (CA), San Jose (CA), San Luis Obispo (CA), Santa Barbara (CA), Seattle (WA), Spokane (WA), Tacoma (WA) and Vancouver (BC).

2.4 East Coast Cities

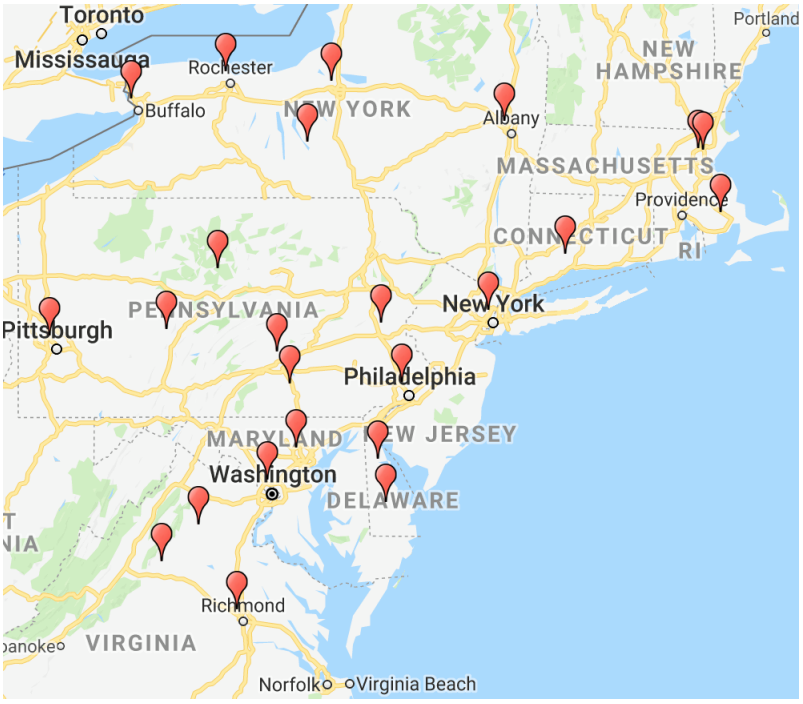

Figure 2: The cities we choose to examine along the East Coast.
In alphabetical order, the cities we choose on the East Coast are: Albany (NY), Allentown (PA), Altoona (PA), Baltimore (MD), Boston (MA), Buffalo (NY), Cambridge (MA), Charlottesville (VA), Concord (NH), Dover (DE), Georgetown (PA), Harrisburg (PA), Ithaca (NY), Manhattan (NY), New Haven (CT), Philadelphia (PA), Pittsburgh (PA), Richmond (VA), Rochester (NY), Syracuse (NY), Washington DC, York (PA).

3 The Model: Objective Function and Constraints

In order to formulate this as a linear program, we must introduce some variables. Let

m be the number of cities we have to chose from. In our case $m = 22$

n is the number of nights we have on our tour.

B is our budget for the tour, in dollars.

D is the maximum distance we are willing to drive in one day.

h is the index of our home city.

g_j the "goodness" of the j city

$d_{a,b}$ is the cost to drive from city a to city b, in dollars.

$x_{i,j} = \begin{cases} 0 \\ 1 \end{cases}$ iff we are in city j on night i of the tour.

$y_{i,j} = \begin{cases} 0 \\ 1 \end{cases}$ iff we play a show in city j on night i.

$t_{a,b} = \begin{cases} 0 \\ 1 \end{cases}$ iff we travel from city a to city b at any time.

This will allow us to keep track of the path we take on our tour, as well as where we play shows to maximize the "value" of each city we play in. Our LP then becomes:

$$\max \sum_{i,j} g_j y_{i,j} \tag{1}$$

$$y_{i,j} \leq x_{i,j} \qquad \text{for } i = 1...n,\ j = 1...m \tag{2}$$

$$\sum_{j} x_{i,j} = 1 \qquad \text{for } i = 1...n \tag{3}$$

$$\sum_{i} y_{i,j} \leq 1 \qquad \text{for } j = 1...m \tag{4}$$

$$x_{i,a} + x_{i+1,b} - 1 \leq t_{a,b} \qquad \text{for } a, b = 1...m \text{ and } i = 1...n-1 \tag{5}$$

$$\sum_{a,b} t_{a,b} d_{a,b} + \sum_{i,j} x_{i,j} c_j \leq B \tag{6}$$

$$d_{a,b} t_{a,b} \leq D \qquad \text{for } a, b = 1...m \tag{7}$$

$$x_{0,h} = 1 \tag{8}$$

$$x_{n-1,h} = 1 \tag{9}$$

Our objective function (1) is straightforward, we want to maximize the sum of the quality of the cities we play shows in. The next set of constraints (2) require that any $y_{i,j}$ is less than the corresponding $x_{i,j}$, that is, you can only play a show in a city on a certain night if you are in that city on that night. The constraints coming from (3) force us to be in exactly one place on each night. For example, if on night i we are in Portland, we can't also be in San Francisco that night. The constraints from (4) make it so we play at most one show in each city, otherwise the objective function would be maximized by just playing all our shows in the most valuable city, which wouldn't be very interesting. The set of constraints (5) is a trick to keep track of travel between cities. If we travel between cities a and b in one step, then $x_{i,a} + x_{i+1,b} - 1 = 1$ so $t_{a,b}$ must also be 1. $t_{a,b}$ could be one even if $x_{i,a} + x_{i+1,b} \neq 2$, but the budget constraint will favor $t_{a,b} = 0$. The budget constraint (6) enforces that our budget is adhered to. The sum of the cost of the distance traveled ($t_{a,b} d_{a,b}$), plus the cost of hotel stays ($x_{i,j} c_j$) must be less than B. The set of constraints from (7) are to require that we do not travel more than D miles in one step. These constraints would rule out solutions that involve driving from Seattle to Los Angeles in one day, for example. The final two constraints (8) and (9) require that we start and end in our home city.

There were many assumptions that went into this model. We assumed that whatever the starting city is is our home, so we would not incur hotel costs in that city. We assumed that the price per mile traveled is the same across all areas, when in reality gas prices can vary widely. We also left the cost of food and drink out of our budget constraint, assuming that this would be the same in all cities. This would make our budget more accurately a *travel* budget. The largest assumption would be that we are guaranteed a venue to play in in any city we go to.

3.1 "Goodness" of Each City

In order to quantify the "goodness" of each of the cities we chose, we collected data we felt represented the quality of life in that city. We collected data on the median age, population, median income, crime rate, and poverty rate. We collected median age data because our goal was to target young adults and college students, we felt this was a good way to identify and prioritize cities with our target audience. Population was also important to us because the larger the population, the more people will come to our show. Median income was another variable we collected data on because a higher median income suggests a more hospitable venue. Crime rate and poverty rate were also collected to quantify the hospitality of the cities we visited[2].

With this data in hand, we had to devise a way to quantify the "goodness" of each city. During discussion, we decided that we should rank our variables based off of our intuition of how they will influence the "goodness" of a city. We choose to rank them as follows: population, income, median age, crime rate, and poverty. Then, we sorted each of the 22 cities by each of the catagories, and took the weighted average of placement of each city to get the "goodness" of each city. We weighed each variable with a linear regression representing the ranking discussed above. We multiplied as follows: 1∗population, 0.8∗income, 0.6∗median age, 0.4∗crime rate, and 0.2∗poverty. Using this schema, the rank of the cities we were examining on the West Coast is:

City	Value	City	Value
Spokane	20.2	Pullman	21.0
Chico	22.6	Tacoma	23.0
San Luis Obispo	23.2	Bellingham	24.2
Corvallis	24.4	Santa Barbara	25.6
Eugene	26.8	Bend	28.0
Salem	29.4	Oakland	30.4
Fresno	32.2	Sacramento	33.0
Portland	35.2	Vancouver	35.8
Los Angeles	37.4	Riverside	37.8
San Francisco	40.8	Seattle	43.4
San Jose	47.6	San Diego	51.0

3.2 Implementation

Implementing this linear program was fairly straightforward. We worked in Python 3.7 and used the open source linear programming solver LPSolve. LPSolve has a Python module, `lpsolve55`, which allows you make calls to the solver from within a Python script. This allowed us to create a single script that would formulate the problem, solve it and interpret the output all in one go. It was especially helpful when playing with parameters to have it all done at once. The script writes out the linear program in the `lp` format. An `lp` file is the standard format for LPSolve, the main points are that the objective function is at the top, then each constraint follows on a new line, the final line is a declaration of all variables and their types. All lines end in semicolons. Examples

[2]Data for West Coast and East Coast cities selected for our model can be found in Appendix C

of `lp` files used to run trials of the LP can be found in Appendix B.1. All of our data was compiled into a `csv` file, so we made use of the `pandas` Python module for importing the data into a data frame to affix the proper coefficients to the right variables. The script for creating and running the LP can be found in Appendix A.2 in its entirety. The main challenge in implementing this linear program is the computational expense of the problem. Using a fully connected graph with 22 vertices results in a lot of possible paths to be explored. We were initially interested in exploring a two week tour ($n = 14$), but LPSolve was not able to come to the optimal solution in a reasonable amount of time. If the parameters were loose, i.e., we ran it with a large budget or not much of a restriction on driving distances, then the optimal solution would be even harder to come to.

For these reasons we chose to set a larger MIP gap. The MIP gap is a tolerance of the branch and bound algorithm that is used by LPSolve. Essentially it checks the difference of the value of the objective function at the best found solution yet and the current solution. If this difference is less than the MIP gap, then this solution and subsolutions are discarded. Increasing the MIP gap will speed up solving time, but it does run the risk that the solution it returns is not the true maximum/minimum. The default MIP gap on LPSolve is 1e-11, considering the "goodness" of each city varies only to one decimal place, we felt setting the MIP gap to be 0.1 would still give us some confidence in the returned solution being at least close to the optimal. This together with searching for only 7 day solutions gave us some interesting results.

4 Solution

Using the LP we had formulated, we found the optimal path for an up and coming band from Seattle with $1000 to spend, 7 days to kill and willing to drive 300 miles in a day is depicted below in Figure 3. LPSolve run information can be found in Appendix B.2. LPSolve produced an objective function value of 196 for this run, indicating that the total "goodness" of the cities played on this trip was 196.

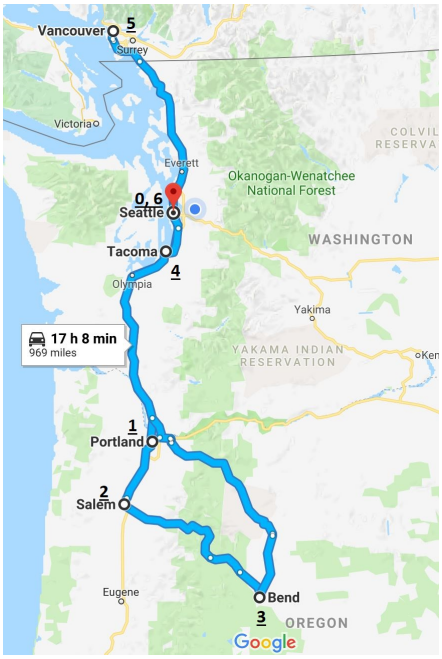

Figure 3: An overlay of the route using Google Maps
Seattle→Portland→Salem→Bend→Tacoma→Vancouver BC→Seattle

5 Analysis

Figure 3 produced an accurate, realistic, and optimal subset of cities to preform in. Personally, all of us believe that if we were given the same constraints LPSolve was, we would come up with the same route. Realistic for our model just simply means that a concert manager would construct a similar path given the same constraints.

After solving and obtaining the route displayed in Figure 3, we began playing with the budget constraint. We discovered that it wasn't binding in this solution. Completing this route only cost \$599.57 of the \$1000.00 budgeted. Our first experiment after this was to vary the budget for the trip, holding all the other parameters constant. We varied the budget from \$500 to \$1400 in increments of \$100 and the results are depicted below in Figure 4. As seen in the figure, despite the budgeted dollars for the trip increasing, the maximum value of the objective function tops out at 196. Along that same vein, the budget constraint was only limiting for $B \leq 600$, after, increasing the budget could not yield a better result. $B = 500$ is the only budget that resulted in a substantially different result. For all other budget levels, the same cities are visited but possibly in different orders.

There must have been some other limiting factor. To investigate further, we explored the maximum distance traveled constraints. In the previous runs varying the budget constraint, the maximum distance to be traveled in one day was 300 miles. In these next runs, depicted in Figure 5, we varied the distance traveled from 50 to 600 miles by 50 miles at a time, while holding the budget to \$1000. This experiment produced some real variation in the results. Additionally, it confirmed that the miles traveled per day was the binding constraint in our model. When we increase the distance we travel in one day,

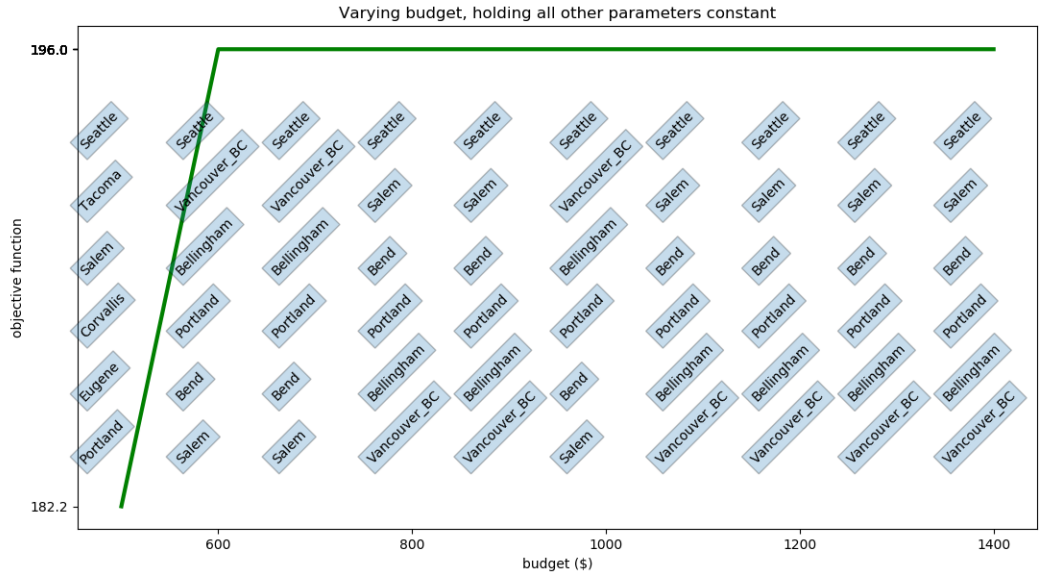

Figure 4: Varying the budget alone did not actually produce much change. Cities displayed vertically are the cities a show was played in at each budget level. The limiting factor is not the budget.

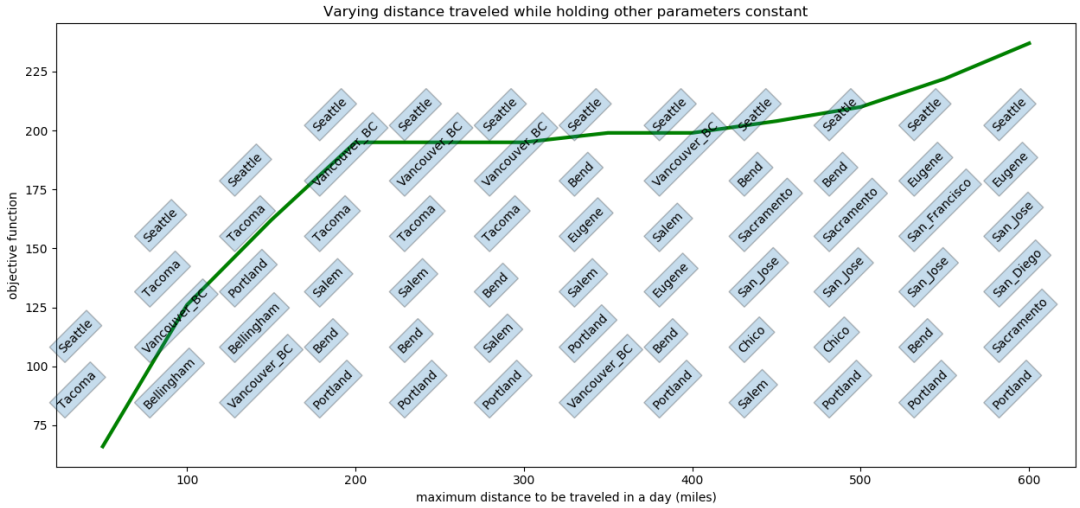

Figure 5: Increasing the distance we can travel in one day allows us cover more ground and get to the valuable cities in the south.

then the set of cities that we play in changes in almost every run. If the distance was too small, $D = 50, 100, 150$, then we could not reach enough cities to play a full set and avoid playing repeats. Then as the distance increased we were able to reach some of the more valuable cities in California. We eventually stretch all the way down to San Diego, the most valuable city, if we we are willing to drive 600 miles in one day. Figure 6 shows

an overlay of this route on Google Maps.

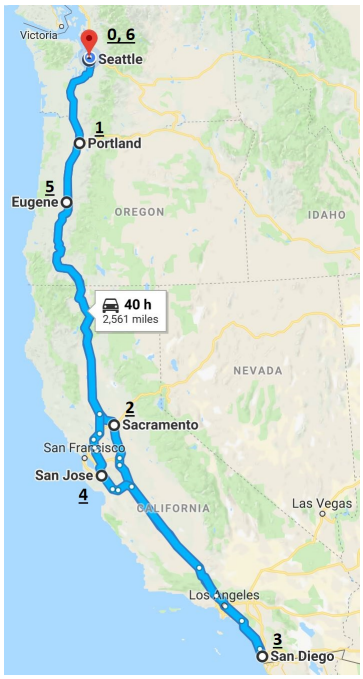

Figure 6: The optimal seven day route, if we are willing to drive 600 miles in a day. Seattle→Portland→Sacramento→San Diego→San Jose→Eugene→Seattle

6 Variations

One variation we were interested in is if we switched one of our constraints with our objective function. Suppose our touring van is really run down, even more than you would expect, and we want to minimize the miles we put on it. Then our objective function would be to minimize miles traveled:

$$\min \sum_{a,b} t_{a,b} d_{a,b} \tag{10}$$

Then we would also want to add constraints that the cities we play shows in have a minimum "goodness" value, so that we don't just play shows in the closest cities to our home. This would create restraints of the form:

$$g_j y_{i,j} \geq G y_{i,j} \qquad \text{for } i = 1...n,\ j = 1...m \tag{11}$$

$$\sum_{i,j} y_{i,j} = n - 1 \tag{12}$$

That way $y_{i,j}$ can only be one if $g_j \geq G$, with G being our minimum value, otherwise, it must be zero to satisfy the constraint. The second constraint is necessary because otherwise, the optimal solution to minimize distance would be to not go anywhere. In the Python script mentioned earlier, an `lp` file of this type can be made by passing the boolean `variation` as `True` to the function `run_it()`. An example of this class of `lp` file can be found in Appendix B.1.

Using the mean value of the "goodness" of the cities as a minimum ($G = 31.5$), starting from Seattle, driving 300 miles a day and having a budget of $1000, this problem is infeasible. There are not enough cities nearby Seattle that are "valuable enough" to hit and get back to Seattle in 7 days. In fact, no city on the west coast is close to enough "valuable" cities for a feasible route starting from any of them. If we relax the constraints some and change G to be the 40th percentile of the "goodness" values ($G = 27.3$), then we can get some solutions. Interestingly, the only cities with feasible solutions with this high of a standard for the cities to play shows in are Fresno, San Francisco and Sacramento. This makes some intuitive sense because those cities are all above the threshold and are centrally located on the map. The cities hit by the various tours include Riverside, San Diego, San Jose, and Oakland. Strangely, Oakland and San Jose did not have feasible routes with them as starting points, despite being included in other routes of this sort.

Upon further review of the solution, we discovered that LPSolve seemed to be ignoring our maximum distance traveled per day constraints in order to produce these solutions. For example, all of these routes included drives from San Jose to Riverside, a distance of 395 miles, when we had $D = 300$ as our set maximum. This produced curiously high objective function values. From Sacramento, the proposed minimum distance was 2279.7 miles, from San Francisco 2333.45 miles and from Fresno 2158 miles. All of these values are greater than $(n-1) \cdot D$, which should be the maximum distance that can be traveled by the formulation of the LP. We believe that LPSolve bypassed this constraint because it is the binding constraint in our model. Bypassing it gave LPSolve the ability to produce

an optimal solution.

Looking at the objective function value is a bit worrisome. Because in this variation, we sought to minimize the distance traveled while also achieving a minimum "goodness". That value doesn't look like a minimum. Upon investigation, we discovered that most of the cities along the West Coast, in particular the Northern West Coast, have significant distance between them. Whereas the cities in California have slightly less distance to travel between "good" cities.

6.1 East Coast

Another variation we tried was using a completely different set of cities to choose from. We imagined that what if instead of being a Seattle based band, we were based in New York. So we collected the same data on East Coast cities and ran them through the LP as well. The cities that we considered and their "goodness" ratings based on the same schema can be found in the table below.

City	Value	City	Value
York	18	Harrisburg	19.2
Altoona	20.8	Ithaca	21.2
Albany	22.2	Rochester	25.2
Georgetown	27.0	Buffalo	27.4
Richmond	28.2	Concord	29.2
Allentown	30.2	Syracuse	31.0
Dover	31.6	New Haven	33.0
Baltimore	33.6	Philadelphia	35.6
Charlottesville	36.4	Pittsburgh	38.4
Washington DC	43.6	Boston	46.4
Manhattan	46.4	Cambridge	48.4

We ran this East Coast set through some of the same tests as the West Coast. First, we varied D, the maximum distance to be traveled in a day while holding all the other variables constant. The results in Figure 7, compared to Figure 5 where we ran the

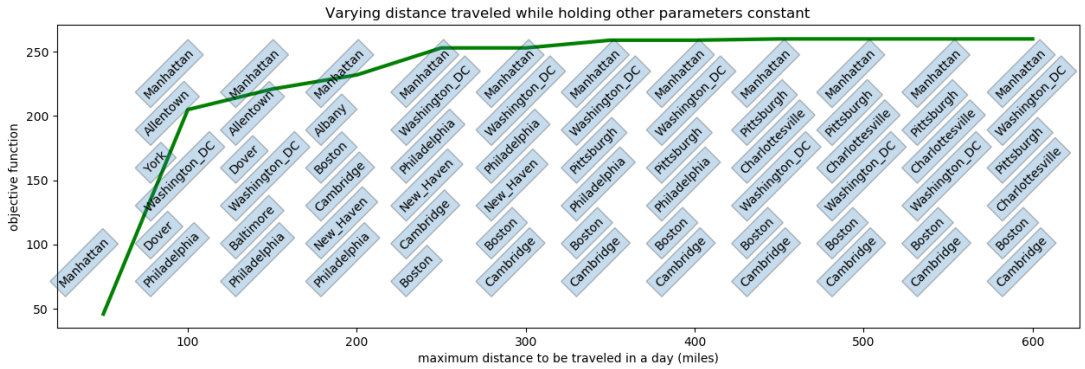

Figure 7: Increasing the mileage per day on our East Coast set. The objective function levels off quicker than the West Coast.

same experiment on the West Coast, show that the "goodness" of the cities reached on the East Coast stops responding to increases in D faster. On the East Coast set, the optimal path for a seven day tour is reached at $D = 350$, hitting the cities of Cambridge, Boston, Philadelphia, Pittsburgh, Washington DC, and Manhattan. On the West Coast, the optimal path was when $D = 600$, this discrepancy must be because the cities we considered are much more geographically close to each other in the East Coast set. Then, as we did on the West Coast, we made the objective function into a constraint for a minimum value to play and ran the LP to minimize distance traveled. Again we used the 40th percentile of the "goodness" values of the cities as our minimal accepted value ($G = 28.6$). The only difference from the West Coast runs was that we ran trials with $D = 150$ miles because the distances between cities are much smaller for the cities we selected on the East Coast. The result was that even more cities had viable routes that hit enough valuable cities. There was one circuit, hitting Allentown, Manhattan, Philadelphia, Dover, Washington D.C. and Baltimore, where every city in the circuit was a viable starting point, in all circuits the distance traveled was 542.7 miles. Additionally, Boston was reported as a viable starting point as well, but similar to the West Coast runs, the minimum distance value (1045 miles) is greater than $(n-1) \cdot D$. So Boston is probably not truly a viable starting route. There must be some loophole in our constraints that were used that allows these discrepancies.

7 Conclusion

This model yielded some interesting results. It showed that our budget doesn't matter much after \$600 for a 7 day tour and that the distance we travel in a day is the binding constraint. From our variation to minimize the distance assuming a minimum "goodness" value, solutions are very difficult to obtain on the West Coast. This characteristic of the model was not shared by the East Coast. Intuitively this makes sense because the East Coast is more densely populated than the West Coast. This allows our touring band to reach good cities within a smaller distance. Whereas on the West Coast, we have to drive further.

It also suggests that bands have more flexibility in choosing optimum cities to play in due to the density. This was exemplified by the fact that we can start in more places on the East Coast and minimize the distance between "good" cities. The model also suggests that starting in Central California is a decent alternative to the East Coast. An interesting extension would be collecting data on the the cities bands were formed in. This would be a way to test the validity of our model.

The model overall aligned with our initial estimations of what the optimal solution would look like. The path targeted the cities that most people would be likely to visit if they were planning this trip by hand. This indicates that the model successfully models the behavior we were targeting. Overall, this formulation was effective at modeling this problem.

Our model is not without its shortcomings as well. One thing we realized late into the

project is that it would've been much better to not have our graph be fully connected, and only have edges between cities that are seperated by D miles or less. This would reduce the number of variables in our LP and probably speed up solve time. Additionally, if we made it impossible to travel to cities that are too far apart, we may have avoided the inconsistency we ran into in our variations where we minimized distance traveled. Another area of improvement would be the quantification of the value of the cities. The schema used was good in that big cities that are visited by larger bands on tour, such as San Francisco, Seattle, San Jose, San Diego, etc., were highly ranked. However, smaller college towns that may have a large population of musically minded people like Eugene, Santa Barbara, Corvallis, etc. were ranked lower than expected. In order to capture the spirit of a college band touring and trying to reach other young listeners, it may have been better to use college attendence data in each city as one of the metrics for calculating value.

It is easy to see ways to increase the scope of our model. Include more cities, gather more data, create a different "goodness" model, inclue alternate means of transportation, or modify the objective to name a few. The future scope of the project would be to inlcude gas milage and different overhead costs to determine the most accurate budget. Since we took an estimate, the model is not perfectly accurate. As an alternate use of the model, one could select a cross country road trip and identify the route to take to tackle all of the major attractions. Or one could do this with bar hopping in a downtown area. Someone could use this to plan tourist excursions in a new city. Or a museum could plan tour routes throughout the musuem to ensure they visit the most important pieces on display. This framework can be applied to many problems to result in an optimal solution.

8 Bibliography

References

[1] United States Census Bureau *2017 Census*, https://www.census.gov/data/tables/2017/demo/popest/total-cities-and-towns.html 2017.

[2] United States Census Beureau *2017 Census*, https://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?src=CF 2017.

[3] Travel Route Centered Metro Map Layout and Annotation *Wu*, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03085.x

[4] Routing trains through railway stations *Kroon*, https://www.sciencedirect.com/science/article/abs/pii/S0377221795003428

[5] Hotels.com *Pricing Projections: August 1, 2019* https://hotels.com Date Accessed: 2/5/19

A Generating Code

A.1 West Coast and East Coast Adjacency Matrix

```
clear all;close all;clc
%Declare West Coast Cities
WC_Cities=["Seattle","Vancouver_BC", "Portland", "Spokane","
   Tacoma",...
    "Eugene", "Corvallis","Sacramento","San_Francisco","
       San_Jose","Chico",...
    "Santa_Barbara","Los_Angeles","San_Diego","Bellingham","
       Pullman","Bend"...
    "Salem","San_Luis_Obispo","Oakland","Fresno","Riverside
       "];
%Distance from one City in the Graph to every other city in
   the graph as a
%vector
Seattle_Dist
   =[140,174,279,33.5,282,256,752,808,836,663,1113,1135,1255,89,
   ...
    285,327,219,1019,801,922,1187];
Vancouver_Dist
   =[315,414,174,424,397,894,950,978,805,1254,1276,1396,53,421,
   ...
    489,361,1161,946,1063,1329];
Portland_Dist
   =[352,144,109,83,579,634,663,490,939,961,1082,262,350,175,
   ...
    46.2,846,628,749,1014];
Spokane_Dist=[293,461,
   435,821,877,905,731,1181,1214,1301,362,75,385,398,...
    1088,870,990,1205];
Tacoma_Dist
   =[252,226,722,778,807,633,1083,1105,1225,122,299,297,190,989,
   ...
    772,892,1158];
Eugene_Dist
   =[47.4,473,529,557,383,833,855,976,371,459,128,66,740,522,
   ...
    642,908];
Corvallis_Dist=[517,573,601,427,877,899,1019,345,433,
   128,39.4,784,566,686,...
    952];
Sacramento_Dist
   =[89,117,88,393,384,505,841,825,437,536,294,82,170,437];
```

```
SF_Dist=[47.5,176,328,381,502,896,885,504,602,233,
   12.2,187,435];
SJ_Dist=[205,281,342,463,925,909,528,626,188,40.9,153,395];
Chico_Dist=[481,470,590,751,729,348,445,380,170,257,523];
SB_Dist=[95,218,1201,1168,797,896,94,318,232,150];
LA_Dist=[120,1224,1113,819,918,189,371,219,55];
SD_Dist=[1344,1201,946,1046,312,491,339,99];
Bellingham_Dist=[368,437,308,1108,890,1010,1276];
Pullman_Dist=[376,388,1124,906,992,1106];
Bend_Dist=[132,704,487,607,872];
Salem_Dist=[802,585,705,971];
SLO_Dist=[224,138,256];
Oakland_Dist=[178,425];
Fresno_Dist=[271];
WC_A_Matrix=zeros(22,22); %Create the 22x22 matrix to store
   the above values
%For loops for each city. These loops go through each data
   set, and
%populate them in the correct row and column.
for z=1:21
    WC_A_Matrix(z+1,1)=Seattle_Dist(z); %populate the column
       corresponding to this City
    WC_A_Matrix(1,z+1)=Seattle_Dist(z); %populate the row
       corresponding to this City.
end
for z=1:20
    WC_A_Matrix(z+2,2)=Vancouver_Dist(z);
    WC_A_Matrix(2,z+2)=Vancouver_Dist(z);
end
for z=1:19
    WC_A_Matrix(z+3,3)=Portland_Dist(z);
    WC_A_Matrix(3,z+3)=Portland_Dist(z);
end
for z=1:18
    WC_A_Matrix(z+4,4)=Spokane_Dist(z);
    WC_A_Matrix(4,z+4)=Spokane_Dist(z);
end
for z=1:17
    WC_A_Matrix(z+5,5)=Tacoma_Dist(z);
    WC_A_Matrix(5,z+5)=Tacoma_Dist(z);
end
for z=1:16
    WC_A_Matrix(z+6,6)=Eugene_Dist(z);
    WC_A_Matrix(6,z+6)=Eugene_Dist(z);
end
for z=1:15
```

```
    WC_A_Matrix(z+7,7)=Corvallis_Dist(z);
    WC_A_Matrix(7,z+7)=Corvallis_Dist(z);
end
for z=1:14
    WC_A_Matrix(z+8,8)=Sacramento_Dist(z);
    WC_A_Matrix(8,z+8)=Sacramento_Dist(z);
end
for z=1:13
    WC_A_Matrix(9,z+9)=SF_Dist(z);
    WC_A_Matrix(z+9,9)=SF_Dist(z);
end
for z=1:12
    WC_A_Matrix(z+10,10)=SJ_Dist(z);
    WC_A_Matrix(10,z+10)=SJ_Dist(z);
end
for z=1:11
    WC_A_Matrix(z+11,11)=Chico_Dist(z);
    WC_A_Matrix(11,z+11)=Chico_Dist(z);
end
for z=1:10
    WC_A_Matrix(z+12,12)=SB_Dist(z);
    WC_A_Matrix(12,z+12)=SB_Dist(z);
end
for z=1:9
    WC_A_Matrix(z+13,13)=LA_Dist(z);
    WC_A_Matrix(13,z+13)=LA_Dist(z);
end
for z=1:8
    WC_A_Matrix(z+14,14)=SD_Dist(z);
    WC_A_Matrix(14,z+14)=SD_Dist(z);
end
for z=1:7
    WC_A_Matrix(z+15,15)=Bellingham_Dist(z);
    WC_A_Matrix(15,z+15)=Bellingham_Dist(z);
end
for z=1:6
    WC_A_Matrix(z+16,16)=Pullman_Dist(z);
    WC_A_Matrix(16,z+16)=Pullman_Dist(z);
end
for z=1:5
    WC_A_Matrix(z+17,17)=Bend_Dist(z);
    WC_A_Matrix(17,z+17)=Bend_Dist(z);
end
for z=1:4
    WC_A_Matrix(z+18,18)=Salem_Dist(z);
    WC_A_Matrix(18,z+18)=Salem_Dist(z);
```

```
111 end
112 for z=1:3
113     WC_A_Matrix(z+19,19)=SLO_Dist(z);
114     WC_A_Matrix(19,z+19)=SLO_Dist(z);
115 end
116 for z=1:2
117     WC_A_Matrix(z+20,20)=Oakland_Dist(z);
118     WC_A_Matrix(20,z+20)=Oakland_Dist(z);
119 end
120 %Populate with Distance from Fresno to Riverside
121 WC_A_Matrix(22,21)=Fresno_Dist;
122 WC_A_Matrix(21,22)=Fresno_Dist;
123 %Put the matrix into a table with labels.
124 WC_A_Table=array2table(WC_A_Matrix,'RowNames',WC_Cities,'
       VariableNames',WC_Cities)
125 %Export to csv for manipulation
126 writetable(WC_A_Table,'WC_Adjacency.csv')
127
128 %% East Coast Adjacency Matrix
129 clear all; close all; clc
130 %Declare East Coast Cities
131 EC_Cities=["Manhattan","Buffalo","Rochester","Syracuse","
       Ithaca","Boston","Philadelphia",...
132     "Pittsburgh","Harrisburg","Altoona","Baltimore","
           Washington_DC","Cambridge","New_Haven",...
133     "Richmond","York","Georgetown","Dover","Charlottesville
           ","Allentown","Condord","Albany"]
134 %Distances from one city to each of the other cities in the
       map
135 Man_Dist
       =[396,334,247,223,214,95,370,190,276,193,229,212,79,339,190,204,
       ...
136     202,342,88,270,158];
137 Buff_Dist
       =[74,152,155,457,404,219,285,209,326,383,454,438,482,310,504,
       ...
138     468,455,355,437,291];
139 Roch_Dist
       =[87,90,392,341,284,264,275,342,385,389,375,484,294,441,405,
       ...
140     504,292,372,226];
141 Syr_Dist
       =[54,311,255,360,256,275,332,374,309,294,474,279,354,319,493,
       ...
142     205,291,146];
```

```
Ith_Dist
   =[331,230,312,208,218,308,350,329,258,450,238,330,294,469,181,
   ...
    311,166];
Bos_Dist
   =[308,571,413,477,407,443,6.2,138,549,400,420,378,556,360,68,169];

Phil_Dist
   =[305,105,236,106,142,306,173,247,101,116,80,255,63,364,237];

Pitt_Dist
   =[203,99,248,246,583,450,336,221,339,331,317,284,641,499];
Harr_Dist
   =[135,78,120,411,253,220,24.7,182,146,245,87,469,296];
Alto_Dist=[180,177,476,350,276,153,271,263,249,216,534,365];
Balt_Dist=[40.7,404,271,154,53,104,107,154,141,462,334];
DC_Dist=[440,306,107,97,106,98,118,190,498,370];
Camb_Dist=[135,546,397,412,375,554,301,68,166];
NH_Dist=[414,264,279,243,421,168,192,150];
Rich_Dist=[210,211,203,71,304,603,476];
York_Dist=[157,130,206,91,456,301];
Geor_Dist=[36.7,224,165,469,345];
Dov_Dist=[216,131,436,311];
Char_Dist=[324,611,533];
Alan_Dist=[365,215];
EC_A_M=zeros(22,22); %Construct 22x22 to store distances
%For loops to populate matrix with the correct distances
for z=1:21
    EC_A_M(z+1,1)=Man_Dist(z); %populate the column
    EC_A_M(1,z+1)=Man_Dist(z); %populate the row
end
for z=1:20
    EC_A_M(z+2,2)=Buff_Dist(z);
    EC_A_M(2,z+2)=Buff_Dist(z);
end
for z=1:19
    EC_A_M(z+3,3)=Roch_Dist(z);
    EC_A_M(3,z+3)=Roch_Dist(z);
end
for z=1:18
    EC_A_M(z+4,4)=Syr_Dist(z);
    EC_A_M(4,z+4)=Syr_Dist(z);
end
for z=1:17
    EC_A_M(z+5,5)=Ith_Dist(z);
    EC_A_M(5,z+5)=Ith_Dist(z);
```

```
end
for z=1:16
    EC_A_M(z+6,6)=Bos_Dist(z);
    EC_A_M(6,z+6)=Bos_Dist(z);
end
for z=1:15
    EC_A_M(z+7,7)=Phil_Dist(z);
    EC_A_M(7,z+7)=Phil_Dist(z);
end
for z=1:14
    EC_A_M(z+8,8)=Pitt_Dist(z);
    EC_A_M(8,z+8)=Pitt_Dist(z);
end
for z=1:13
    EC_A_M(9,z+9)=Harr_Dist(z);
    EC_A_M(z+9,9)=Harr_Dist(z);
end
for z=1:12
    EC_A_M(z+10,10)=Alto_Dist(z);
    EC_A_M(10,z+10)=Alto_Dist(z);
end
for z=1:11
    EC_A_M(z+11,11)=Balt_Dist(z);
    EC_A_M(11,z+11)=Balt_Dist(z);
end
for z=1:10
    EC_A_M(z+12,12)=DC_Dist(z);
    EC_A_M(12,z+12)=DC_Dist(z);
end
for z=1:9
    EC_A_M(z+13,13)=Camb_Dist(z);
    EC_A_M(13,z+13)=Camb_Dist(z);
end
for z=1:8
    EC_A_M(z+14,14)=NH_Dist(z);
    EC_A_M(14,z+14)=NH_Dist(z);
end
for z=1:7
    EC_A_M(z+15,15)=Rich_Dist(z);
    EC_A_M(15,z+15)=Rich_Dist(z);
end
for z=1:6
    EC_A_M(z+16,16)=York_Dist(z);
    EC_A_M(16,z+16)=York_Dist(z);
end
for z=1:5
```

```
    EC_A_M(z+17,17)=Geor_Dist(z);
    EC_A_M(17,z+17)=Geor_Dist(z);
end
for z=1:4
    EC_A_M(z+18,18)=Dov_Dist(z);
    EC_A_M(18,z+18)=Dov_Dist(z);
end
for z=1:3
    EC_A_M(z+19,19)=Char_Dist(z);
    EC_A_M(19,z+19)=Char_Dist(z);
end
for z=1:2
    EC_A_M(z+20,20)=Alan_Dist(z);
    EC_A_M(20,z+20)=Alan_Dist(z);
end
Conc_Dist=[151];
%Populate with distance from Concord to Albany
EC_A_M(21,22)=Conc_Dist;
EC_A_M(22,21)=Conc_Dist;
%Send matrix to an array with labels
EC_A_T=array2table(EC_A_M,'RowNames',EC_Cities,'VariableNames
   ',EC_Cities)
%Export to csv
writetable(EC_A_T,'EC_Adjacency.csv')
```

A.2 Python Code for Running LPSolve

```
import pandas as pd
from lpsolve55 import *
import numpy as np

def goodness(dat):
        ''' returns a vector with the value of each city in the
        order they appear in the dataframe(alphabetical).
        Value is calculated by ordering the cities by each
        parameter, either ascending or descending depeneding
        on whether having a lot of that parameter is "good"
        or "bad" then summing up the rank of each city in
        each category. Therefore the best cities will have
        high numbers'''
        pop = dat['Population']
        inc = dat['Median_Income']
        age = dat['Median_Age']
        crime = dat['Crime_Rate']
        pov = dat['Poverty_Rate']
```

```
19
20          stats = [pop,inc,age,crime,pov]
21          scales = np.linspace(1,0.2,5)
22          asc = [True,True,False,False,False]
23          rank = np.zeros(len(dat))
24          for stat, updown, scale in zip(stats,asc,scales):
25              order = stat.sort_values(ascending=updown).index
26              for i in range(len(order)):
27                  rank[order[i]] += scale*i
28          return rank
29
30  def run_it(dat,area,n,home,D,B,mpg,cpg,output,variation=False,
        quan=0):
31          ''' runs the model with the given parameters,
32          and prints the path to take to output, as well as any
33          constraints that are large enough to be of interest,
34          i.e., the budget constraint and travel constraints.
35          area = 'WC' or 'EC'
36          n = int, number of days in tour
37          home = int < 22, index of home city
38          D = maximum miles to drive in one day
39          B = budget of tour in dollars
40          mpg = miles per gallon of tour van
41          cpg = dollars per gallon of gas
42          output = file handle for writing
43          variation = Boolean, whether to run the
44          minimize ditance variation
45          quan = float [0,1], quantile to use as
46          G, the minimum goodness in the variation
47          '''
48          m = len(dat) # number of cities to choose from
49          output.write('area: '+area+', n: '+str(n)+', home: '+str
                (home)+', D: '+str(D))
50          output.write(', B: '+str(B)+', mpg: '+str(mpg)+', cpg: '
                +str(cpg)+'\n')
51          g = goodness(dat)
52          def d(a,b):
53                  '''returns the distance from city a to city b, a
                        and b are indexes'''
54
55                  # first index is the name of the city itself
56                  return dat.iloc[a][b+1]
57
58          var_names = [] # container to hold the
59                        # names of the vars in the order LP solve
                              will return them
```

```
60          def var(xy,i,j,plus=True,c=''):
61                  '''returns a string '+xy_i_j', also adds this
                       string to the set of variables in the
62                  order they get added to the lp file '''
63                  v = xy + '_' + str(i) + '_' + str(j)
64                  if v not in var_names:
65                          # puts the variables in a list as they
                               are introduced to LP solve
66                          var_names.append(v)
67                  return plus*'+' + str(c) + v
68
69          f = open(area + '_tour.lp','w')
70
71          # first write the objective function
72          if not variation:
73                  # the sum of all the shows we play shows in
74                  # multiplied by their "goodness"
75                  line = 'max: '
76                  for i in range(n):
77                          for j in range(m):
78                                  line += var('y',i,j,c=g[j])
79
80                  f.write(line + ';\n')
81          else:
82                  # in the variation we want to minimize distance
                       traveled
83                  line = 'min: '
84                  for a in range(m):
85                          for b in range(m):
86                                  if a != b:
87                                          line += var('t',a,b,c=d(
                                               a,b))
88                  f.write(line + ';\n')
89
90          # next, the constraints that we can only play a show in
                a city if we are in that city
91          # on that night. For every night and every city
92          for i in range(n):
93                  for j in range(m):
94                          f.write(var('y',i,j) + '<=' + var('x',i,
                               j) + ';\n')
95
96          # next, a constraint we are in exactly one city on every
                night
97          for i in range(n):
98                  line = ''
```

```
99                  for j in range(m):
100                         line += var('x',i,j)
101                 f.write(line + '=1;\n')
102
103         # next, a set of constraints that we play at most one
                show in each city
104         for j in range(m):
105                 line = ''
106                 for i in range(n):
107                         line += var('y',i,j)
108                 f.write(line + '<=1;\n')
109
110         # create dummy binary variables t_a_b = 1 iff we travel
                from a to b at any time
111         for a in range(m):
112                 for b in range(m):
113                         if a != b:
114                                 for i in range(n-1):
115                                         f.write(var('x',i,a) +
                                            var('x',i+1,b) + '
                                            -1<=' + var('t',a,b)
                                            + ';\n')
116
117         # constraint for maximum miles we want to travel in one
                day
118         for a in range(m):
119                 for b in range(m):
120                         if a != b:
121                                 # add in names to these
                                    constraints so lp solve doesn
                                    't
122                                 # interpret it as a bound
123                                 f.write('drive_' + str(a) + '_'
                                    + str(b) + ': ' +
124                                         var('t',a,b,c=d(a,b)) +
                                            '<=' + str(D)+ ';\n')
125
126         # budget constraint, total distance traveled, converted
                to a dollar amount,
127         # summed with cost of hotel for every night of the tour
128         line = ''
129         for a in range(m):
130                 for b in range(m):
131                         if a != b:
132                                 gas = d(a,b) * cpg / mpg
133                                 line += var('t',a,b,c=gas)
```

```
134                 for i in range(n):
135                         cost = (a != home) * dat.iloc[a]['
                                Hotel_Price']
136                         line += var('x',i,a,c=cost)
137         f.write(line + '<=' + str(B) + ';\n')
138
139         # constraint to start and end in our home city
140         # again add names so LP solve doesn't interpret them as
                bounds
141         f.write('home_start: ' + var('x',0,home) + '=1;\n')
142         f.write('home_stop: ' + var('x',n-1,home) + '=1;\n')
143
144         # in the variation, we want a minimum value of cities we
                play a show in
145         if variation:
146                 M = np.quantile(g,quan)
147                 line = ''
148                 for j in range(m):
149                         for i in range(n):
150                                 f.write('g'+str(i)+str(j)+': ' +
                                        var('y',i,j,c=g[j])+'>='+var
                                        ('y',i,j,c=M)+';\n')
151                                 line += var('y',i,j)
152                 # have to require that we play some shows
                        otherwise, we wont go anywhere to minimize
                        distance
153                 f.write(line + '=' + str(n-1) + ';\n')
154
155         # finally, declare all of our variables as binary
156         line = 'bin '
157         for j in range(m):
158                 for i in range(n):
159                         line += (i+j!=0)*',' + var('x',i,j,plus=
                                False)
160                         line += ',' + var('y',i,j,plus=False)
161                 for jj in range(m):
162                         if j != jj:
163                                 line += ',' + var('t',j,jj,plus=
                                        False)
164         f.write(line + ';')
165
166         f.close()
167
168         # load the lp file into the python wrapper
169         lp = lpsolve('read_LP',area + '_tour.lp')
170         # set the MIP gap, this changes the limit between the
```

```
        best solution found
171         # and the current solution, results in not
172         # always finding the true optimum but
173         # speeds up computation time
174         lpsolve('set_mip_gap',lp,True,0.1)
175         lpsolve('solve',lp)
176         obj = lpsolve('get_objective',lp);
177         # these return a list, the first item in the list is the
                list of vars/constraints
178         values = np.array(lpsolve('get_variables',lp)[0])
179         consts = np.array(lpsolve('get_constraints',lp)[0])
180         time = lpsolve('time_elapsed',lp)
181         lpsolve('delete_lp',lp)
182
183         var_names = np.array(var_names)
184         # none of the values are exactly equal to one for some
              reason
185         ones = var_names[values>0.2]
186         ones = np.array([s.split('_') for s in ones])
187         # sort by second element, the night
188         ones = sorted(ones,key=lambda x : int(x[1]))
189
190         for row in ones:
191                 if row[0] == 'x':
192                         output.write('Night '+row[1]+' in '+dat.
                            iloc[int(row[2])]['City']+'\n')
193                 elif row[0] == 'y':
194                         output.write('Show in '+dat.iloc[int(row
                            [2])]['City']+' on night '+row[1]+'\n
                            ')
195
196
197         output.write(str(consts[consts>5])+'\n')
198         output.write('obj = ' + str(obj))
199         output.write(', time = ' + str(time))
200         output.write('\n')
```

B LPSolve Information

B.1 LP File Descriptions

Following is an example first of an `lp` file used in running the main problem, to maximize "goodness". This run was on the West Coast with $n = 7$ days in the tour.

```
max: +24.2y_0_0+28.0y_0_1+...+24.2y_1_0+28.0y_1_1+...+23.0y_6_20+35.8y_6_21;
```

First, the objective function, to maximize the "goodness" of the shows we play.

$$\max \sum_{i,j} g_j y_{i,j}$$

```
+y_0_0<=+x_0_0;
+y_0_1<=+x_0_1;
...
```

Then, the constraints that we can only play a show in a city if we are there. There are 154 ($m \cdot n$) constraints of this type, one for each x.

```
+x_0_0+x_0_1+x_0_2+...+x_0_21=1;
+x_1_0+x_1_1+x_1_2+...+x_1_21=1;
...
```

Then, the constraints that we can only be in one city at a time. There are $n = 7$ of these constraints.

$$\sum_j x_{i,j} = 1 \quad \text{for } i = 1..n$$

```
+y_0_0+y_1_0+y_2_0+y_3_0+y_4_0+y_5_0+y_6_0<=1;
+y_0_1+y_1_1+y_2_1+y_3_1+y_4_1+y_5_1+y_6_1<=1;
...
```

Then, the constraints that we play at most one show in any city. There are $m = 22$ of these constraints.

$$\sum_i y_{i,j} \leq 1 \quad \text{for } j = 1..m$$

```
+x_0_0+x_1_1-1<=+t_0_1;
+x_1_0+x_2_1-1<=+t_0_1;
...
```

Next, the constraints that set up the travel binary variables used to keep track of driving distance. There are $m^2 - m = 462$ of these constraints. We did not include ones for staying in the same city, e.g., $t_{a,a}$.

```
drive_0_1: +437t_0_1<=300;
drive_0_2: +751t_0_2<=300;
...
```

Next the constraints that prevent the solution from requiring us to drive over a certain distance in a day.

$$d_{a,b} t_{a,b} \leq D \quad \text{for } a = 1...m,\ b = 1...m,\ a \neq b$$

There are 462 of these constraints, one for every t. Note that these were named so LPSolve does not interpret them as a bound on t.

```
+101.97t_0_1+175.23t_0_2+...+12.37t_0_21+62.33x_0_0+62.33x_1_0+...
+40.6t_21_20+...+76.0x_6_21<=1000;
```

Then the budget constraint. The sum of all travel expenses and hotel expenses must be less than our budget.

$$\sum_{a,b} t_{a,b} d_{a,b} + \sum_{i,j} x_{i,j} c_j \leq B$$

```
home_start: +x_0_18=1;
home_stop: +x_6_18=1;
```

Then two final constraints that make sure we start and end in the same city, 18 is the index of Seattle.

```
bin: x_0_0,y_0_0,x_1_0,y_1_0,...,x_6_21,y_6_21,t_0_1,t_0_2,...,t_21_20;
```

Finally, declare all $2(m \cdot n) + m^2 - m = 770$ variables as binary variables.

Here, another example of an `lp` file, this one was used to run the variation of our LP where we minimized distance. This example is from an East Coast run with $n = 7$ days in the tour.

```
min: +215t_0_1+365t_0_2+...+301.0t_0_21+215t_1_0+216t_1_2+...+97.0t_21_20;
```

First, the objective function, to minimize distance traveled over the tour:

$$\min \sum_{a,b} t_{a,b} d_{a,b}$$

```
g00: +22.2y_0_0>=+28.6y_0_0;
g10: +22.2y_1_0>=+28.6y_1_0;
...
```

Then, the constraints that if we play a show in a city, it must be up to standard. There are 154 constraints of this type, one for each y variable.

$$g_j y_{i,j} \geq G y_{i,j} \quad \text{for } i = 1...n,\ j = 1...m$$

```
+y_0_0+y_0_1+...+y_1_0+y_1_1+...+y_6_20+y_6_21=6;
```

Then, the constraint that we play $n - 1$ shows in the tour.

$$\sum_{i,j} y_{i,j} = n - 1$$

All of the rest of the constraints and variable declarations are identical to the main problem.

B.2 LPSolve Run Information

Below is some of the run information from all of the trials we ran on LPSolve. In the first table below, we have the runs on the West Coast, varying the budget while holding everything else constant.

$$n = 7 \quad m = 22 \quad \text{area} = \text{West Coast} \quad h = 18 \text{ (Seattle)} \quad D = 300$$

B (\$)	Obj. Fxn	Run Time (s)
500	182.2	4.20
600	196.0	4.53
700	196.0	18.11
800	196.0	37.14
900	196.0	42.94
1000	196.0	73.14
1100	196.0	42.35
1200	196.0	43.83
1300	196.0	58.03
1400	196.0	73.20

In the next two tables we have the trials varying the maximum distance to be traveled in a day, D. On the left is the West Coast trials, on the right, the same trial but on the East Coast cities.

$$n = 7 \quad m = 22 \quad h_w = 18 \text{ (Seattle)} \quad h_e = 13 \text{ (Manhattan)} \quad B = 1000$$

D (miles)	West Coast Obj Fxn	Run Time (s)	East Coast Obj Fxn	Run Time (s)
50	66.4	12.11	46.4	41.66
100	126.4	23.15	205.4	8.02
150	161.6	22.22	221.0	9.16
200	194.8	8.71	232.0	6.02
250	194.8	12.23	253.4	1.32
300	196.0	29.30	253.4	1.35
350	198.6	16.60	258.8	0.38
400	198.6	135.41	258.8	0.47
450	204.0	62.85	259.6	0.14
500	209.8	67.86	259.6	0.14
550	221.8	37.34	259.6	0.15
600	237.0	17.81	259.6	0.13

Finally, here is the data from the runs where we tried to minimize distance traveled, but put a constraint on the "quality" of the cities we visit. Then we found which cities were feasible starting points.

$$n = 7 \quad m = 22 \quad D_w = 300 \quad D_e = 150 \quad B = 1000$$

Starting City	Obj Fxn (miles)	Run Time (s)
Bellingham	infeasible	0.04
Bend	infeasible	4.68
Chico	infeasible	0.04
Corvallis	infeasible	0.06
Eugene	infeasible	0.06
Fresno	2158	1241.95
Los Angeles	infeasible	179.53
Oakland	infeasible	764.72
Portland	infeasible	20.41
Pullman	infeasible	0.13
Riverside	infeasible	106.35
Sacramento	2279.7	312.22
Salem	infeasible	21.76
San Diego	infeasible	9.66
San Francisco	2333.45	487.36
San Jose	infeasible	931.10
San Luis Obispo	infeasible	0.05
Santa Barbara	infeasible	0.04
Seattle	infeasible	8.67
Spokane	infeasible	0.06
Tacoma	infeasible	0.05
Vancouver	infeasible	0.93

Starting City	Obj Fxn (miles)	Run Time (s)
Albany	infeasible	0.06
Allentown	542.7	10.91
Altoona	infeasible	0.06
Baltimore	542.7	12.15
Boston	1045.4	12.07
Buffalo	infeasible	16.68
Cambridge	infeasible	0.05
Charlottesville	infeasible	0.4
Concord	infeasible	9.42
Dover	542.7	18.57
Georgetown	infeasible	0.09
Harrisburg	infeasible	0.13
Ithaca	infeasible	0.10
Manhattan	542.7	27.47
New Haven	infeasible	323.40
Philadelphia	542.7	75.22
Pittsburgh	infeasible	0.40
Richmond	infeasible	0.10
Rochester	infeasible	0.09
Syracuse	infeasible	0.16
Washington D.C.	542.7	17.57
York	infeasible	0.11

C Data

C.1 West Coast Data

	Seattle	Vancouver BC	Portland	Spokane	Tacoma	Eugene	Corvallis	Sacramento	San Francisco	San Jose
Seattle	0	140	174	279	33.5	282	256	752	808	836
Vancouver BC	140	0	315	414	174	424	397	894	950	978
Portland	174	315	0	352	144	109	83	579	634	663
Spokane	279	414	352	0	293	461	435	821	877	905
Tacoma	33.5	174	144	293	0	252	226	722	778	807
Eugene	282	424	109	461	252	0	47.4	473	529	557
Corvallis	256	397	83	435	226	47.4	0	517	573	601
Sacramento	752	894	579	821	722	473	517	0	89	117
San Francisco	808	950	634	877	778	529	573	89	0	47.5
San Jose	836	978	663	905	807	557	601	117	47.5	0
Chico	663	805	490	731	633	383	427	88	176	205
Santa Barbara	1113	1254	939	1181	1083	833	877	393	328	281
Los Angeles	1135	1276	961	1214	1105	855	899	384	381	342
San Diego	1255	1396	1082	1301	1225	976	1019	505	502	463
Bellingham	89	53	262	362	122	371	345	841	896	925
Pullman	285	421	350	75	299	459	433	825	885	909
Bend	327	489	175	385	297	128	128	437	504	528
Salem	219	361	46.2	398	190	66	39.4	536	602	626
San Luis Obispo	1019	1161	846	1088	989	740	784	294	233	188
Oakland	801	946	628	870	772	522	566	82	12.2	40.9
Fresno	922	1063	749	990	892	642	686	170	187	153
Riverside	1187	1329	1014	1205	1158	908	952	437	435	395

Chico	Santa Barbara	Los Angeles	San Diego	Bellingham	Pullman	Bend	Salem	San Luis Obispo	Oakland	Fresno	Riverside
663	1113	1135	1255	89	285	327	219	1019	801	922	1187
805	1254	1276	1396	53	421	489	361	1161	946	1063	1329
490	939	961	1082	262	350	175	46.2	846	628	749	1014
731	1181	1214	1301	362	75	385	398	1088	870	990	1205
633	1083	1105	1225	122	299	297	190	989	772	892	1158
383	833	855	976	371	459	128	66	740	522	642	908
427	877	899	1019	345	433	128	39.4	784	566	686	952
88	393	384	505	841	825	437	536	294	82	170	437
176	328	381	502	896	885	504	602	233	12.2	187	435
205	281	342	463	925	909	528	626	188	40.9	153	395
0	481	470	590	751	729	348	445	380	170	257	523
481	0	95	218	1201	1168	797	896	94	318	232	150
470	95	0	120	1224	1113	819	918	189	371	219	55
590	218	120	0	1344	1201	946	1046	312	491	339	99
751	1201	1224	1344	0	368	437	308	1108	890	1010	1276
729	1168	1113	1201	368	0	376	388	1124	906	992	1106
348	797	819	946	437	376	0	132	704	487	607	872
445	896	918	1046	308	388	132	0	802	585	705	971
380	94	189	312	1108	1124	704	802	0	224	138	256
170	318	371	491	890	906	487	585	224	0	178	425
257	232	219	339	1010	992	607	705	138	178	0	271
523	150	55	99	1276	1106	872	971	256	425	271	0

City	Median Age	Population	Median Income	Crime Rate	Poverty Rate	Hotel Price
Bellingham	31.1	89045	44441	2.91	22.2	62.33333333
Bend	38.2	94520	55625	1.72	12.4	87
Chico	29.9	93293	43148	4.57	25.2	64
Corvallis	27	57961	43922	1.33	27.5	74
Eugene	34.1	168916	44859	3.89	23.1	62.66666667
Fresno	30.5	527438	44905	5.91	28.1	53.66666667
Los Angeles	35.2	3999759	54432	7.7	19.5	63.33333333
Oakland	36.4	425195	68060	13.18	18.9	91.33333333
Portland	36.8	647805	62127	5.19	14.7	72.66666667
Pullman	21.9	33354	27831	0.96	40.8	77.33333333
Riverside	31.3	327728	63548	5.14	15.1	68.66666667
Sacramento	34.3	501901	55187	6.78	19	64.33333333
Salem	35.2	169798	49126	3.91	17.2	72.66666667
San Diego	34.3	1419516	71481	3.72	13.1	76
San Francisco	38.3	884363	103801	7.25	10.1	120
San Jose	36.4	1035317	101940	4.14	10.7	117.6666667
San Luis Obispo	26.5	47541	47777	4.73	32.7	77
Santa Barbara	37.9	92101	66916	4.71	13.9	81
Seattle	35.7	724745	83476	6.33	11.5	0
Spokane	35.8	217108	43274	6.26	19.7	59.33333333
Tacoma	35.9	213418	53553	8.24	17.9	63.33333333
Vancouver BC	41	631490	72662	5.78	13.8	76

This is the adjacency matrix for the West Coast cities. It describes the distances from one city to each of the other cities we are examining in our project, which is measured in miles.[3] And above is the data that we used to determine the "goodness" of our cities of interest for the West Coast[4].

[3]We collected data using the miles of the fastest path described by Google Maps. Coalesced the data using Matlab to export the csv file.

[4]See Bibliography for sources of data

C.2 East Coast Data

This is the East Coast data we collected. The adjacency matrix again measures distance from a city to every other city in the graph. This distance is measured as the fastest driving route determined by Google Maps directions. The population data was collected from the U.S. Census Bureau. And the hotel data was collected from hotels.com.

	Manhattan	Buffalo	Rochester	Syracuse	Ithaca	Boston	Philadelphia
Manhattan	0	396	334	247	223	214	95
Buffalo	396	0	74	152	155	457	404
Rochester	334	74	0	87	90	392	341
Syracuse	247	152	87	0	54	311	255
Ithaca	223	155	90	54	0	331	230
Boston	214	457	392	311	331	0	308
Philadelphia	95	404	341	255	230	308	0
Pittsburgh	370	219	284	360	312	571	305
Harrisburg	190	285	264	256	208	413	105
Altoona	276	209	275	275	218	477	236
Baltimore	193	326	342	332	308	407	106
Washington DC	229	383	385	374	350	443	142
Cambridge	212	454	389	309	329	6.2	306
New Haven	79	438	375	294	258	138	173
Richmond	339	482	484	474	450	549	247
York	190	310	294	279	238	400	101
Georgetown	204	504	441	354	330	420	116
Dover	202	468	405	319	294	378	80
Charlottesville	342	455	504	493	469	556	255
Allentown	88	355	292	205	181	360	63
Condord	270	437	372	291	311	68	364
Albany	158	291	226	146	166	169	237

	Pittsburgh	Harrisburg	Altoona	Baltimore	Washington DC	Cambridge	New Haven
Manhattan	370	190	276	193	229	212	79
Buffalo	219	285	209	326	383	454	438
Rochester	284	264	275	342	385	389	375
Syracuse	360	256	275	332	374	309	294
Ithaca	312	208	218	308	350	329	258
Boston	571	413	477	407	443	6.2	138
Philadelphia	305	105	236	106	142	306	173
Pittsburgh	0	203	99	248	246	583	450
Harrisburg	203	0	135	78	120	411	253
Altoona	99	135	0	180	177	476	350
Baltimore	248	78	180	0	40.7	404	271
Washington DC	246	120	177	40.7	0	440	306
Cambridge	583	411	476	404	440	0	135
New Haven	450	253	350	271	306	135	0
Richmond	336	220	276	154	107	546	414
York	221	24.7	153	53	97	397	264
Georgetown	339	182	271	104	106	412	279
Dover	331	146	263	107	98	375	243
Charlottesville	317	245	249	154	118	554	421
Allentown	284	87	216	141	190	301	168
Condord	641	469	534	462	498	68	192
Albany	499	296	365	334	370	166	150

	Richmond	York	Georgetown	Dover	Charlottesville	Allentown	Condord	Albany
Manhattan	339	190	204	202	342	88	270	158
Buffalo	482	310	504	468	455	355	437	291
Rochester	484	294	441	405	504	292	372	226
Syracuse	474	279	354	319	493	205	291	146
Ithaca	450	238	330	294	469	181	311	166
Boston	549	400	420	378	556	360	68	169
Philadelphia	247	101	116	80	255	63	364	237
Pittsburgh	336	221	339	331	317	284	641	499
Harrisburg	220	24.7	182	146	245	87	469	296
Altoona	276	153	271	263	249	216	534	365
Baltimore	154	53	104	107	154	141	462	334
Washington DC	107	97	106	98	118	190	498	370
Cambridge	546	397	412	375	554	301	68	166
New Haven	414	264	279	243	421	168	192	150
Richmond	0	210	211	203	71	304	603	476
York	210	0	157	130	206	91	456	301
Georgetown	211	157	0	36.7	224	165	469	345
Dover	203	130	36.7	0	216	131	436	311
Charlottesville	71	206	224	216	0	324	611	533
Allentown	304	91	165	131	324	0	365	215
Condord	603	456	469	436	611	365	0	151
Albany	476	301	345	311	533	215	151	0

City	Population	Median Age	Median Income	Crime Rate	Poverty Rate	Hotel Price
Albany	98251	30.9	30784	9.02	32.8	62.33333333
Allentown	121283	31.8	37256	4.59	26.7	55.66666667
Altoona	44098	38	36741	2.74	22.3	55
Baltimore	611648	35	47350	20.42	21.9	61
Boston	685094	32	67846	7.03	21	96.66666667
Buffalo	258612	32.8	32883	10.22	30.5	59
Cambridge	113630	30.4	83122	2.94	14	252
Charlottesville	48019	30.9	50727	3.38	25.9	80
Concord	43019	39.6	56459	0.73	12.2	136
Dover	37538	29.9	46355	7.43	19.7	79
Georgetown	7291	31	46708	6.17	23.7	68.66666667
Harrisburg	49192	31.1	32688	11.49	31.7	61.33333333
Ithaca	31006	21.8	29230	2.74	44.8	59.66666667
Manhattan	8622698	36.2	66739	4.76	17.3	144.6666667
New Haven	131014	30.7	38126	8.43	26.1	67.33333333
Philadelphia	1580863	34.1	41449	9.45	25.7	59.66666667
Pittsburgh	302407	32.9	44707	6.91	19.2	64.33333333
Richmond	227032	33.5	41187	11.88	25.4	49
Rochester	208046	31.7	30784	9.02	32.8	69.66666667
Syracuse	143396	30.6	32704	7.22	33.6	63
Washington DC	693972	33.9	75506	10.05	18.6	60.66666667
York	44132	30.5	30068	10.7	36	57.33333333