|  |
| --- |
|  |
| Pub Crawl Optimization  Harry Birnbaum, Caitlin Braun, Nikhil Byanna, Felix Dumont, Trevor Thompson |
| |  |  |  | | --- | --- | --- | | Team 1 | 8/15/19 | 15.066 | |

Table of Contents

[Motivation and Context 2](#_Toc16497315)

[Data Filtering 2](#_Toc16497316)

[Optimization Model Formulation 2](#_Toc16497317)

[High-Level Description 2](#_Toc16497318)

[Assumptions 2](#_Toc16497319)

[Decision Variables 2](#_Toc16497320)

[Objective Function 2](#_Toc16497321)

[Set 3](#_Toc16497322)

[Parameters 3](#_Toc16497323)

[User Inputs 3](#_Toc16497324)

[Data Filtering Constraints 3](#_Toc16497325)

[Model Constraints 3](#_Toc16497326)

[Results 7](#_Toc16497327)

[Example Experiment 7](#_Toc16497328)

[Qualitative Discussion 7](#_Toc16497329)

[Sensitivity Analysis 7](#_Toc16497330)

[Implementation Issues 7](#_Toc16497331)

[Appendix 7](#_Toc16497332)

[Python Code 7](#_Toc16497333)

# Motivation and Context

“What care I how time advances? I am drinking ale today.” - Edgar Allen Poe

For over six thousand years, beer has been engrained in our culture. Dating back to nearly 5000 BC, ancient civilizations brewed varieties of beer in large quantities. The beer was used for sustenance, medical purposes and as a form of currency. Recipes and brewing techniques developed over time. In 2018, the U.S. beer industry sold over 200 million barrels of beer. Based on beer shipments and the US Census, U.S. citizens of age 21 years and older consumed 26.5 gallons of beer per person in 20181. To say we love our beer would be considered an understatement by many.

Given this affinity for beer, many people find themselves indulging in the activity of drinking beer on the weekends in the form of a “bar crawl”. However, there is an ever-growing number of potential bars to go to, making it difficult for a group to decide on the best set of bars to include on the bar crawl. Team 1 is here to help!

We propose creating a model for our final project that will optimize a group’s bar crawl route to optimize a function including the aggregate rating of all of the bars that are included on their crawl and the total walking plus wait time at bars needed to complete the crawl. Using a Yelp dataset for the various cities in North America, we will employ a combination of an assignment model and the Traveling Salesman algorithm to select a set of bars that the group will include on their bar crawl, including the suggested route between the bars. Using inputs from the user such as the number of bars they would like to visit, the time and day they want to do the crawl, their desired walking time, and specific parameters regarding the quality and quantity of the Yelp reviews for the bars, we will create additional constraints for the model to ensure the user’s desires are met.

# Data Filtering

# Optimization Model Formulation

## High-Level Description

## Assumptions

## Decision Variables

The optimal bar crawl route is calculated with the following decision variables:

## Objective Function

The objective function will maximize the Yelp rating of the bars visited on the crawl:

## Set

We will define the following set for the formulation of our linear program:

* where n is the number of locations included in our data set

## Parameters

The following parameters are also used in our formulation:

## User Inputs

We will also ask the user for the following inputs to formulate the relevant constraints for our model:

## Data Filtering Constraints

*Minimum Bar Yelp Rating*

The rating for each location on the bar crawl much be greater than or equal to the minimum rating specified by the user:

*Minimum Number of Yelp Reviews*

The number of reviews for each location on the bar crawl must be greater than or equal to the minimum rating specified by the user:

## Model Constraints

*The Number of Locations Visited*

The number of locations visited on the bar crawl must be equal to the number of locations specified by the user:

*Maximum Time Between Locations*

The maximum walk time between any 2 locations *i* and *j* on the bar crawl must be less than or equal to the maximum time specified by the user:

*Maximum Total Walk Time*

The maximum total walk time for the full bar crawl must be less than or equal to the maximum time specified by the user:

*Rating Minimum*

The rating for each location on the bar crawl much be greater than or equal to the minimum rating specified by the user:

*Review Minimum*

The number of reviews for each location on the bar crawl much be greater than or equal to the minimum rating specified by the user:

*No Movements Between the Same Bar*

*From/To Upper Bound*

*From/To Lower Bound*

# rules about z  
  
# can only have one 1 per movement matrix  
for k in range(bar\_num - 1):  
 m.addConstr(quicksum([z[k][i][j] for i in range(len(locations)) for j in range(len(locations))]) == 1)  
  
# make sure we don't visit the same bar twice  
# dimension1  
for i in range(len(locations)):  
 m.addConstr(quicksum([z[k][i][j] for j in range(len(locations)) for k in range(bar\_num - 1)]) <= 1)  
  
# dimension2  
for j in range(len(locations)):  
 m.addConstr(quicksum([z[k][i][j] for i in range(len(locations)) for k in range(bar\_num - 1)]) <= 1)  
  
# have to start from the bar you previously went to  
for k in range(1, bar\_num - 1):  
 for i in range(len(locations)):  
 m.addConstr(quicksum([z[k - 1][j][i]  
 for j in range(len(locations))]) == quicksum(  
 [z[k][i][j] for j in range(len(locations))]))  
  
# open time - only distance is considered for the time being  
for zed in range(bar\_num - 1):  
 m.addConstr(start\_time + zed \* time\_spent\_each\_bar + quicksum([z[w][i][j] \* (dima[i][j] + wait\_times[i])  
 for i in range(len(locations))  
 for j in range(len(locations))  
 for w in range(zed)]) >= quicksum(  
 [open\_times[i] \* quicksum(z[zed][i])  
 for i in range(len(locations))]))  
m.addConstr(start\_time + zed \* time\_spent\_each\_bar + quicksum([z[w][i][j] \* (dima[i][j] + wait\_times[i])  
 for i in range(len(locations))  
 for j in range(len(locations))  
 for w in range(bar\_num - 1)]) >=  
 quicksum([open\_times[j] \* quicksum([z[bar\_num - 2][i][j] for i in range(len(locations))])  
 for j in range(len(locations))]))  
# Close time constraint  
for zed in range(bar\_num - 1):  
 m.addConstr(start\_time + zed \* time\_spent\_each\_bar + quicksum([z[w][i][j] \* (dima[i][j] + wait\_times[i])  
 for i in range(len(locations))  
 for j in range(len(locations))  
 for w in range(zed)]) <= quicksum(  
 [close\_times[i] \* quicksum(z[zed][i])  
 for i in range(len(locations))]))  
m.addConstr(start\_time + zed \* time\_spent\_each\_bar + quicksum([z[w][i][j] \* (dima[i][j] + wait\_times[i])  
 for i in range(len(locations))  
 for j in range(len(locations))  
 for w in range(bar\_num - 1)]) <=  
 quicksum([close\_times[j] \* quicksum([z[bar\_num - 2][i][j] for i in range(len(locations))])  
 for j in range(len(locations))]))  
  
# Must exit last bar before close time  
m.addConstr(start\_time + zed \* time\_spent\_each\_bar + quicksum([z[w][i][j] \* (dima[i][j] + wait\_times[i])  
 for i in range(len(locations))  
 for j in range(len(locations))  
 for w in range(bar\_num - 1)]) <= end\_time)  
  
# Total wait time less than max allowed  
m.addConstr(quicksum([wait\_times[i] \* y[i] for i in range(len(locations))]) <= max\_total\_wait)  
m.setParam('TimeLimit', 30)  
m.setParam('MIPFocus', 1)  
m.optimize()

# Results

## Example Experiment

## Qualitative Discussion

## Sensitivity Analysis

## Implementation Issues

# Appendix

## Python Code