**Optimizing Washington’s Health Services for Travel Time**

by Arslan Shahid

**Abstract:**

Washington state wants to optimize the usage of its health resources; such that people can easily access health services. Access, in this context, is defined as the amount of travel time taken to reach the nearest health facility. Since, health facilities can easily be overburdened, people sometimes have to go to other facilities, far from their nearest facility, to get health services.

In order to solve the problem analysis of the population, available resources and travel times is needed.

**Formal Definition of the Problem:**

|  |  |  |
| --- | --- | --- |
| Facility ID | Zip Code | Facility Staff |
| A | 98007 | 21 |
| B | 98290 | 52 |
| C | 98065 | 43 |
| D | 98801 | 9 |
| E | 98104 | 64 |

*Fig 1 - Table of Data*

Using the above data, the problem requires to find the optimal resource allocation for every facility. The condition for optimality is that the ratio of staff to patient should be 1:2808.

The problem requires us to find population of each area and travel time in between facilities using publicly available data. Suggested resources are US Census Bureau for population and Google Maps for average travel time.

Using that data, we are required to find an optimal reallocation of resources such that people’s average travel time is minimized.

**Simplifying Assumptions**:

**Data:**

1. Since, all population data available from US Census Bureau API, was till 2016; it is assumed that no drastic changes in population occur in subsequent years.
2. Reaching a facility in the same city/zipcode will have no or negligible travel time. For example, people in Seattle will take no time to reach the facility in Seattle. The reasoning behind this is that we are given a scenario where it is assumed in the question that people in, say area A will like to go to a facility in area A, even though it is technically possible that they might prefer facility B.
3. In order to consider the mode of transportation when computing average travel times, it is assumed that people will take the same transportation they would take if they were going to work. The reasoning for this is that data for work mode of transportation is available. It is also assumed that people will either drive themselves, take the bus or walk, when they are travelling to a health facility.
4. It is assumed that each facility will cater to at most five nearest zip codes. It would be unreasonable that a facility will only cater to the population in their zip code. It would also be unreasonable that a facility in say Seattle will cater to all the people of Seattle; since, it is a big metropolitan area and would skew our results.

**Optimization:**

1. In the question statement it was written that at least half of people near a facility will go to a health facility to get a check-up. In the solution it was assumed that all the population in an area will avail health services. The reasoning for this is that if we can cater for the extreme case, we automatically cater somewhat for every other case. If a facility can tolerate the maximum amount of people in the area, then it can cater for half of that population.
2. It is assumed that the ratio of health-worker to patient 1:2808 is enough to cater for any contingency and any facility above this ratio has a surplus of resources and any facility below this has a deficit of resources. It is also embedded in this assumption that all facilities are similar except for the number of staff.
3. It is assumed that only people from the five specified areas and nearby zip codes will come for check-ups, all other areas are not considered in the problem.
4. When choosing the nearest facility, only the one-way trip from say area A to B is considered not the return trip. The reasoning is to simplify the code and in most cases the return trip is approximately equal to the starting trip; so, in most cases optimizing for the one-way trip automatically optimizes for the whole trip.
5. It is assumed as a principle that people in all these areas have adequate information about travel time and resources of facilities. For example, consider three facilities A, B, C. B is nearest to A and C is further away. If resources are scarce in facility A then everyone will go to facility B, since it is the nearest. They will only turn to facility C if facility B also doesn’t have adequate resources.

**Data Gathering:**

**1.Miscellaneous Data:**  Getting name of location using zip code.

Using Google Maps API Geocode Utility, we can find the name of the location. This may come in handy later when extracting other data. For full implementation, go to Miscellaneous Data Section in the jupyter notebook attached.

Sample Code: 

Documentation for Google Maps API Geocoding:

[Developers Google – Maps Geocoding](https://developers.google.com/maps/documentation/geocoding/intro)

**2. Average Travel Time Data:**  Getting average travel time in between facilities.

Using Google Maps API’s Distance Matrix function, one can easily get average time in between two locations. One can specify the name of the two locations and it also allows user to specify mode of transport. Possible modes of transport: Driving, Transit(bus) or walking. Since, it is reasonable that not everyone will drive, one needs to find appropriate weights for every mode of transport.

Google Maps also takes care of the terrain when it calculates the average time in between two point and also the traffic.

For full implementation of this part, go to attached jupyter notebook’s Estimating Average Time Section.

Sample Code:

1. **import** googlemaps
2. # SPECIFY API KEY TO GET ACCESS
3. google = googlemaps.Client('api\_key')
5. #GOOGLE MAPS API'S DISTANCE MATRIX CAN BE USED TO FIND Duration of travel.
6. #This gets distance in between Bellevue and Seattle and transportation mode is driving
7. matrix = google.distance\_matrix('Bellevue','Seattle',mode='driving')
8. # CurrentDur stores the Duration in between two areas as a float.
9. currentDur =float(matrix['rows'][0]['elements'][0]['duration']['value'])

Due to a lack of data about people’s preferred mode of transportation for visiting healthcare centers, it was necessary to use a proxy for that data. Since, public data on transportation used for travelling to work was available, it was used instead.

Data Source for mode of transportation used when travelling to work:

[Bureau of Transportation Statistics US Government – Commuting to work](https://www.bts.gov/content/commuting-work)

The source states that 82.8% preferred taking the car to work while, 6.3% preferred bus/public transport and 3.5% preferred walking in the state of Washington.

After normalizing this we get the weights for driving - 89.42%, transit - 6.8% and walking – 3.78%. Using Google Maps to get average time for each mode of transportation we can then weight each of them.

Documentation for Google Maps API Distance Matrix:

[Developers Google – Distance Matrix](https://developers.google.com/maps/documentation/distance-matrix/start)

**3. Population Data:**  Getting Population Data from US Census API.

The easiest way to extract data from US Census Bureau is to use their API. The API can be accessed via a Python library called ‘census’.

The American Community Survey 5-year (ACS5) conducted by US census bureau has accurate estimates for the population in each zip code. In order to extract data using from ACS5 database we need information for variables collected in the survey. Each variable collected in the survey has its own name/code and label. Since, we are interested in total population; we need to see the variable and code for that variable from [here](https://api.census.gov/data/2017/acs/acs5/variables.html).

The name/code for variable labeled total population is B01003\_001E.

Data for year 2016 was only available currently, so it was assumed that no drastic changes in population occurred since then.

Now each facility in the problem has been given a zip code but it would be unreasonable to assume that the facility only caters for people in its respective zip code. Instead, for this problem it was assumed that every facility will cater to at most 5 zip codes in the area. For example, facility A which is in Bellevue City should cater the population of the whole city which has 5 zip codes.

So, we need to find zip code near each facility. In order to solve the problem, a database of all US zip codes was needed, fortunately one such free database existed [here](https://simplemaps.com/data/us-zips).

For full implementation go to the attached jupyter notebook’s Searching US Census section.

Sample Code:

1. **from** census **import** Census
2. #Calls the census api and since population data for 2016 was the most recently available.
3. c = Census("api\_key",year=2016)
4. # B01003\_001E is code for total population and 98007 is the zip code for facility A
5. # population variable contains the population for facility A as an integer type.
6. population = c.acs5.zipcode('B01003\_001E',98007)[0]['B01003\_001E']
7. #Output of population is about 27000

Documentation for US Census API:

[US Census Developer’s Guide](https://www.census.gov/data/developers/guidance/api-user-guide.Overview.html)

[Github – Census Documentation](https://github.com/datamade/census)

**Optimization**

The solution approach taken for solving this problem is the ‘Greedy’ approach.

Formally, the greedy approach or algorithm says to pick a locally optimal step at each iteration and the locally optimal step is determined using what is called the greedy heuristic.

For a more intuitive understanding of this approach consider the example where you a burglar who has broken into a house to steal items with the most value. In this problem you have a knapsack that has limited space. So, the greedy approach to solving this problem is for you take the items with the most value first. You keep putting them in the knapsack until you run out of space.

More information on the greedy algorithm can be found in this lecture:

[MIT OCW – Introduction to Computational Thinking and Data Science – Optimization](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/lecture-videos/lecture-1-introduction-and-optimization-problems/)

And also:

[Greedy Algorithm - Wikipedia](https://en.wikipedia.org/wiki/Greedy_algorithm)

**Inputs**: Average Time in between facilities, current Staff and Population.

The heuristic used for this problem is to first find the facility which is most under resourced. Then find nearest neighbors with surplus resources. Then transfer any surplus from these neighbors to the deficient facility until either the neighbors run out of surplus or the deficient facility has ample staff.

In order to solve this problem using the greedy approach, first we need to find the closest neighboring facilities with surplus resources for every facility.

Sample Code for finding nearest neighbor:

1. #Consider we only have 4 facilities A, B, C, D
2. #AverageTimeA is a list of time from A to B,C,D. with the first element being time from A to B
3. AverageTimeA = [1,3,2]
4. #SurplusFacilities is a list of all facilities with adequate resources.
5. SurplusFacilities= ['C','D']
6. #The full implementation of FindClosest function is in the attached jupyter notebook
7. #ListOfNearestA is the list of facilities with adequate resources but ranked based on
8. #their closeness to A
9. ListOfNearestA = findClosest(AverageTimeA,SurplusFacilities)
11. #Output of ListOfNearestA = ['D','C']

Now we need to find a list of nearest for every facility, so we will have 5 lists in our original problem.

For full implementation check out the notebook’s “Finding Nearest Facility with adequate staff” section. The notebook works with a pandas data frame so the code might look different but, works the same.

Now we need to find the amount of health staff in deficit or surplus for every facility.

For full implementation see “Amount of Surplus or Deficit” part of notebook.

Sample Code:

1. **import** numpy as np
2. #Required - is the ratio 1/2808 in floating point
3. required =0.0003561
4. #Using the numpy ceiling function to get how much staff is needed
5. **def** CalculateK(pop):
6. k = np.ceil(pop\*required)
7. **return** k
8. PopulationA = 140815.0
9. CurrentStaff =21
10. Deficit = CurrentStaff - CalculateK(PopulationA)
12. #Output of Deficit = -30,
13. #indicating we need 30 more to make this facility have the required ratio

Running this for all areas would give us surplus/deficit for every facility.

Applying the greedy step:

As previously stated, we will need to give resources to the most in need or one with highest deficit. Then we give it resources from its nearest neighbor with surplus resources. Since, we are taking into account the nearest neighbors we are reducing time taken to get health services. Since, an assumption is that people will go their nearest facility if resources are scare in their home facility.

Consider for example we have facility A with a deficit of 30, and nearest neighbor are in this order

B,C,D. Each with a surplus of 15,10,30. So by our greedy approach A will receive all of B and C’s resources, but only 5 of D’s resources.

The code for this section is solely in the notebook, as it was too large to be posted as a sample snippet.

**Limitations Of Greedy Approach/Model**

* Usually results in a suboptimal solution. The greedy approach doesn’t necessarily allocate resources in such a way as to give the most or globally optimal solution. Considering our example, the reallocation suggested doesn’t reduce the average travel time the most.
* Limitations due to heuristic or greedy strategy used. Usually in greedy approaches using different metrics for how to be ‘greedy’ result in different answers. Some are more optimal and hence closer to globally optimal solution.

**Why use the Greedy Approach?**

* It is a simple and elegant way to redistribute. It doesn’t require too much computation; due to this, it can be abstracted to much larger datasets.
* Since, finding optimal reallocations can be a combinatorial problem, it takes a lot of time to find the globally optimal solution.
* Greedy algorithms are usually very good approximators for globally optimal solutions. They are said to come very close usually.

**Academic Sources where the Greedy Algorithm is used for reallocation:**

1)<https://www.jstor.org/stable/3008845?Search=yes&resultItemClick=true&searchText=greedy&searchText=algorithm&searchText=for&searchText=resource&searchText=allocation&searchUri=%2Faction%2FdoBasicSearch%3FQuery%3Dgreedy%2Balgorithm%2Bfor%2Bresource%2Ballocation&ab_segments=0%2Fdefault-1%2Frelevance_config_with_defaults_duplicate&refreqid=search%3A3f5ce4af608a2ec4bc45ced16353be7e&seq=1#page_scan_tab_contents>

2) <https://www.sciencedirect.com/science/article/pii/S187661021100991X>

Note: These sources were skimmed through in order to get the general idea about the greedy approach because, these papers are very mathematically dense and are difficult to fully understand. Nonetheless, they are cited to give legitimacy to my approach.