Franchises Compared: CALK's Analysis of Two Toronto McDonald's Locations

Carlos Solares, Aryan Hrishikesh Nair, Leandro Hamaguchi Vieira Brasil, Karan Singh

April 2024

Introduction

What makes one McDonald's more successful than another? Is it the area it was built in, is it the reviews it has, is it dependent on the traffic surrounding the location? It is likely a huge mix of all these factors, each contributing a certain amount to the McDonald's success, i.e., each factor has a different weight, depending on how much it affects the McDonald's.

The following project is a case study analyzing two Downtown Toronto McDonald's locations. Our goal is to find each McDonald's strengths and weaknesses, and ultimately compare the two using a fixed number. Think of us as consultants hired by McDonald's to examine what they're doing well and what they could improve on. The result will help the company not only make improvements to these existing locations, but also help it in their decision-making processes when deciding to open a new location.

Our Datasets

Our datasets were handcrafted using a wide range of sources. We compiled them on the .csv files within our submission. The following is an overview of the sources we used to craft these datasets.

- We first analyzed a section of Downtown Toronto using Google Maps. This selected section extends from Augusta Avenue in the West to University Avenue in the East, and from Queen Street* in the South to Dundas Street in the North. The 2 McDonald's in this region are the one at Queen and Spadina, and the one next to the Art Gallery of Ontario at Dundas and McCaul. We also used Google Maps to obtain a large part of our tangible data. We manually calculated the distance between every vertex on the map in real life to use for our edge data, and the longitude and latitude values for each of our vertices for our visualization. Finally, we also used reviews of our McDonald's restaurants as a weight for our locations.
- We also used many City of Toronto websites to obtain information about the intersections and streets in this region. In particular, we used a map of road classifications of Downtown Toronto to create our road hierarchy weight in our edge data. We also used a dataset from Open Data Toronto which provided the traffic at all intersections of the city, divided by vehicle traffic, pedestrian traffic, and bike traffic. We manually combed through this data to generate a final vehicle, pedestrian, and bike daily traffic counts for the Queen and Spadina, and McCaul and Dundas intersections.
- We used arcGIS HUB to find the current posted speed limits for all streets within our perimeter to use for our edge data.
- We used areavibes.com to obtain crime rate data for our map, down to the exact roads.
- Finally, we used the websites of both of these McDonald's location to find their Wi-Fi Availability and their Operating Hours.

Computational Overview

0.1 Introductory Overview

Our program uses Weighted Vertices and Weighted Graphs to map out the region of downtown Toronto we are focusing on. The McDonald's themselves, the intersections between streets, competing restaurants in the region,

landmarks in the region, streetcar stops and subway stations, are all vertices. The segments of roads connecting them are edges, each with weights calculated by weighting the factors we mentioned above.

0.2 Weighting System

We decided to score our McDonald's based on two sets of factors: tangible and intangible. Tangible factors take into account the area surrounding our McDonald's locations and are mainly used in our edge data and vertices not including our McDonald's vertices. Intangible takes into account the "intangible" factors which are stored within the McDonald's vertices themselves. The following is a list of the weights we used to calculate the final score of the intangible factors of our graph:

1. Vehicular Traffic: 0.1
2. Pedestrian Traffic: 0.12. Note: This is higher than vehicular traffic because most of the customers entering these locations are coming in as pedestrians.
3. Bike Traffic: 0.05
4. Reviews: 0.15
5. Operating Hours: 0.1
6. Drive-Thru: 0.2. Most fast food locations get a large part of their sales in the drive-thru. The fact that neither of these locations has a drive-thru hurts their business.
7. Wi-Fi Availability: 0.05
8. Physical Limitations: 0.17. This weight is our score for the actual structure of the locations. We visited these locations and decided that the Queen and Spadina McDonald's had significantly better facilities than the AGO McDonald's.

The following is a list of the weights we used to score the tangible factors. Note that, since we did not have direct access to traffic data for streets in this area, we had to improvise and use the following weights instead:

1. Crime Rate: 0.45
2. Road Hierarchy: 0.35
3. Speed Limits: 0.2

The speed limit weight is the lowest because realistically, busy roads such as Queen and Spadina are too congested to abide by the full speed limit, therefore there is usually not a huge discrepancy between roads with lower and higher speed limits. We weighted road hierarchy higher than speed limit because more lanes signifies more flow of traffic. Finally, crime rate is the highest because it has a direct impact on consumer decision. If something is in a sketchier area, consumers are less likely to go there.

0.3 Data Classes

We used the following Data Classes:

1) _WeightedVertex: Similar to Exercise 4, our Project uses Weighted Vertices as a means of storing the weight for each edge. Each vertex has 6 attributes:

1. item: This stores the name of the vertex as a string.

2. vertex_data: A dictionary holding the numerical factors for each vertex. Not all vertex types have numerical factors associated with them, in which case vertex_data is an empty dictionary.

3. neighbours: The neighbours of the vertex. This is a dictionary mapping the Weighted Vertex to a list of integers/floats, the float representing the weight of the edge between them and the integer represnting the real-world distance between the vertices in metres.

4. cluster: The cluster the vertex is a part of.*

5. co-ordinates: It is a tuple of the vertex's coordinates, with the y value (longitude) as the first entry in the tuple, and the x value (latitude) as the second entry in the tuple.

6. vertex_type: This is a string representing the type of the vertex, i.e., whether it is a McDonald's, an IntersectionMain, an IntersectionSmall, a TTC stop, a Landmark, or OtherRestaurant.

*Note about Clusters: Many of the locations we have selected are too close to one another to include a distance factor between them. For such locations, we decided to put them in a 'cluster'. A cluster is an abstract attribute, which can be though of as a bubble containing all the vertices in the cluster. These vertices must be connected to one another for the program to run, but since these vertices are almost in the same spot, the edges between them are unweighted, as in, they are dummy edges, to ensure that our graph is connected. Our graph has 14 clusters and a vertex's cluster attribute indicates which cluster it is in. However, not all vertices are part of a cluster, and these vertices therefore have a cluster value of 0.

A crucial method of this dataclass is the best_weighted_path method, which, given 2 vertices, returns the optimal path between them and the score of this path, taking into account factors like distance, safety, road hierarchy and speed limit (The lower the score, the better the path).

2) WeightedGraph: This is a variation of the Graph dataclass taught in class, which was covered in Exercise 4. This dataclass has one attribute: vertices: This is a dictionary, mapping a string/ an integer to a Weighted Vertex/ a dictionary mapping a string to a Weighted Vertex. For vertices that aren't a part of a cluster, vertices maps the vertex name (a string) to the vertex object. For clusters, vertices maps the cluster number (an integer) to a dictionary which maps the name of each vertex in the cluster to its corresponding vertex object.

This dataclass includes the standard WeightedGraph methods from class and previous assignments, like add_vertex, add_edges, calculate_edge_weight, get_neighbours and adjacent, with some of them modified to suit the needs of our Project. We have also included methods of our own such as:

1. get_vertices: Because of clusters, simply calling the vertices attribute will not return a dictionary of all vertex names and objects. Therefore, we decided to implement this method, which returns the name of every vertex in the graph (regardless of whether or not it is in a cluster) mapped to the corresponding vertex object.

2. get_cluster: This method returns all the vertices in present in the cluster denoted by the input integer. The returned item is a dictionary of the names of all name of the vertices in the cluster mapped to their corrsponding vertex objects.

3. create_cycle: Our idea was to connect all vertices in a cluster using a cycle. This function takes in a list of names of vertices present in the graph and connects them in a cycle.

GraphGenerator is a class created inside program_data.py that generates two weighted graphs: normal_graph and scaled_graph. The normal_graph is a graph that has its edges with weights equivalent to the real-world distance between a vertex (a place) and another vertex. The scaled_graph is one of the main focuses of this case study because it's a graph with its edges' weights equal to the real-world distance times a factor that was calculated.

To generate a weighted graph, GraphGenerator uses 4 main methods: load_graph, load_vertex_data, load_clusters, and load_edge_data. load_graph receives all data file paths, factor weights, and data labels and outputs a loaded weighted graph. load_vertex_data populates the weighted graph with vertices and its data. load_edge_data generates all edges and computes the weights for each. load_clusters generates all clusters in the graph and connects them as a cycle.

0.4 Visualization

For the visualizations, we used the plotly library. Our first function called visualise_map takes in vertex data and edge data and uses plotly to overlay each vertex on the map of Toronto. The edge data is used to represent edges in our graph. Moreover, each vertex has been assigned a colour depending on its cluster value to aid visualization. Our visualize_tree_map function creates a tree map with all of the factors that contribute to the success of each McDonald's. The factor that contributes the most is on the outside of the Tree Map with the next largest inside until we reach the factor that is contributing the least. The treemap_data.csv file contains values on the maximum possible values ie the maximum pedestrian traffic in Toronto, Maximum Bike Traffic, etc. By taking the ratio of the data for each McDonald's over the maximum value we obtain and number. This number multiplied by the factors contributing weight gives us the final value to decide how big the certain factor contributing to the success of that particular McDonald's.

Instructions

Our datasets are included in our uploaded zip file on markus. These are edge_data.csv, vertex_data.csv, and treemap_data.csv. To run our program, simply run main.py in the python console. Then you will be prompted to type a command. If you type map, you will be taken to our visualization of our section of Toronto. If you type treemap2 you will see a tree graph of the final weights of the Queen and Spadina McDonald's. If you type treemap1 you will see a tree graph of the final weights of the McCaul and Dundas McDonald's. If you type compute, you will

see the final scores of each McDonald's, with the Queen Spadina score as the first score, and the AGO one as the second score. You can also type exit to exit.

Project Plan Changes

Initially, our idea was to create what we called a franchise calculator. This would be a program which, when given all McDonald's locations in the world and their respective data, would create a decision tree. The data stored within each vertex would contain the stores' respective profit margins, expenses, and other key data that would ultimately serve in helping the user of this program make a decision on where they should open their own franchise location. Essentially, we wanted to create a program that would answer the question of "how can I make an informed decision on where in the world to open a new franchise location based on a given data set of all existing franchise locations of that restaurant?"

Analysis

Results

Based on our analysis, we found a score for each of the 2 McDonald's: the one at the intersection of Queen and Spadina had a rounded score of 85.76, while the one by the AGO had a rounded score of 41.62. This score was calculated by weighing the above-mentioned factors; since a lower score for a path between two vertices represents a better path between them, this score was reciprocated and added to the scores obtained by weighing the innate factors of each of the McDonald's. A larger overall score thus signifies a more successful McDonald's and our analysis therefore validates our hypothesis of the McDonald's over at Queen and Spadina being more prosperous overall than the one by the AGO/OCAD.

Limitations

- We were not able to find data for the traffic on each individual road in this section of Toronto. This data would have potentially made our analysis more accurate and created more detailed especially in regards to calculating the scores of the weight.
- We were also unable to find data for the number of customers that these McDonald's locations received daily, as that is private information. Therefore, we used the average of daily McDonald's customers in Canada as a rough estimate.
- Our project may have been more accurate had we used and created a larger map. This would have allowed us to take into account more landmarks, TTC stations, and competing restaurants, but due to the time restriction on this project, we decided to restrict our map perimeter to what we have now.

Next Steps

First off, we would start by creating a larger map. As previously mentioned, this would make our analysis much more accurate and detailed. Within the map that we already have, we could add many more restaurants in the 'competing restaurants' category. We chose to only keep fast food restaurants or restaurants which would potentially have the same customer base as McDonald's, however expanding this category would also likely make our analysis more accurate.

References

"Arcgis Hub." ArcGIS Hub, hub.arcgis.com/datasets/68a87df9a8344d0691bce3f05315dffa/explore?location=43.646301

Areavibes. "Toronto, on Crime Rates." Toronto, ON Crime Rates: Stats and Map, www.areavibes.com/toronto-on/crime/. Accessed 4 Apr. 2024.

City of Toronto Road Classification of Streets List, www.toronto.ca/wp-content/uploads/2018/11/9287-TS$_{R}oad-Classification_{C}ity-Streets-2018.pdf.Accessed4Apr.2024.$

Munro, Steve. "TTC Ridership Update October 2023." Steve Munro, 13 Nov. 2023, stevemunro.ca/2023/11/13/ttc-ridership-update-october-2023/.

"Open Data Dataset." City of Toronto Open Data Portal, open.toronto.ca/dataset/traffic-volumes-at-intersections-for-all-modes/. Accessed 4 Apr. 2024.

Staff. "The 10 Busiest Intersections in Toronto Are ..." Toronto Star, 10 Apr. 2015, www.thestar.com/news/gta/the-10-busiest-intersections-in-toronto-are/article$_b$c1c6ff9 − 7c5d − 5972 − 941b − 576f5cead98a.html.

too, Kenneth Kimutai. "The Busiest Subway Stations in Toronto." WorldAtlas, WorldAtlas, 25 Apr. 2017, www.worldatlas.com/articles/the-busiest-subway-stations-in-toronto.html.

Toronto, www.toronto.ca/wp-content/uploads/2019/09/97a3-TS$_R$CS − CC − 2019$_{TEY.pdf.Accessed4Apr.2024.}$

unimaginative2. "TTC: Complete Subway Station Ridership Figures." UrbanToronto, UrbanToronto, 1 Mar. 2006, urbantoronto.ca/forum/threads/ttc-complete-subway-station-ridership-figures.3047/.

Visit Our Toronto, (Food Court) 109 Mccaul Street, Location — McDonald's, www.mcdonalds.com/ca/en-ca/location/toronto/village-by-the-grange-relo/food-court-109-mccaul-street/29131.html. Accessed 4 Apr. 2024.

Visit Our Toronto, 160 Spadina Avenue , Location — McDonald's, www.mcdonalds.com/ca/en-ca/location/toronto/queen–spadina/160-spadina-avenue/21873.html. Accessed 4 Apr. 2024.