Suggesting Public Transit Networks Given Points of Interest

IB Mathematics HL Internal Assessment

Contents

1 Introduction

As a child, I was obsessed with Thomas the Tank Engine. When I was six, my dad took me on an outing into town on the commuter train because he knew I would be so excited to go on *a real life train!* What started out as a passion for locomotives gradually evolved into an interest in the design and implementation of public transit networks. *What factors are considered when planning metro lines? How can transfer times between different modes of transport be minimized?*

In this Mathematics IA, I develop a methodology for optimizing the placement of transit stations given the points of interest (POIs) in a given region. Specifically, I seek to propose a layout for a brand new metro system in the city of Montreal, Canada. Montreal already has a 68-station metro system. However, I chose to suggest a new system for the city because I am familiar with its neighbourhoods and could use my knowledge of the region to validate my results. In theory, this same methodology could be used in a city that does not have an existing transit system, or even to plan a network for autonomous vehicles, drones, etc. While my methodology is far from perfect, it does offer interesting results and could be used as a springboard in the process of urban planning.

Due to the numerous, complex calculations required to achieve this, a program I wrote in R, a language in which I have a lot of experience, was used to process the dataset. This program is based on several publicly licensed, open-source R packages that are identified throughout this IA where relevant. The source code for my program is available in Appendix B.

2 Data Preparation

My methodology aims to ease congestion in dense urban areas by offering public transit alternatives. These dense areas can be identified by looking at where all the POIs in the city are located. This being said, a dataset containing the name, latitude, longitude, and type of every POI in Greater Montreal was obtained using the Google Places API ("Places API Web Service", 2018). A POI could be a business, a park, a restaurant, a library, a fire station, a school, etc.

Figure 1 contains a map of every POI in the city.

As can be clearly seen in Figure 1, there are significantly more POIs in the downtown core of the city and along major thoroughfares than in the rural areas off the island.

2.1 Weighting the POIs

Once all the POIs are collected, we will heuristically assign each one a relative weighting w depending on what type of POI it is (as given by the Google Places API): POIs are weighted more heavily depending on how important it is for them to be in proximity to a transit station. The domain of the weights used is $w \in [\,0, 1\,]$, with a POI of 0 having no impact on the output of the algorithm, and a weighting of 1 having a strong impact on the output of the algorithm. For example, universities and hospitals received a heavy weighting of 1, small businesses such as post offices and real estate agencies received a light weighting of 0.05, and POIs such as fire stations, gas stations, and existing transit stations (which I did not want to influence the suggested configuration) received a weighting of 0.

Table 1 summarizes the weights assigned to each POI type for this paper.

3 POI Clustering

Because this methodology aims to alleviate congestion in dense urban cores, an efficient suggested transit system will seek to maximize the number of weighted POIs that can be made accessible to passengers. As such, the collection of POIs needs to be searched for dense pockets that would be ideal candidates for station locations.

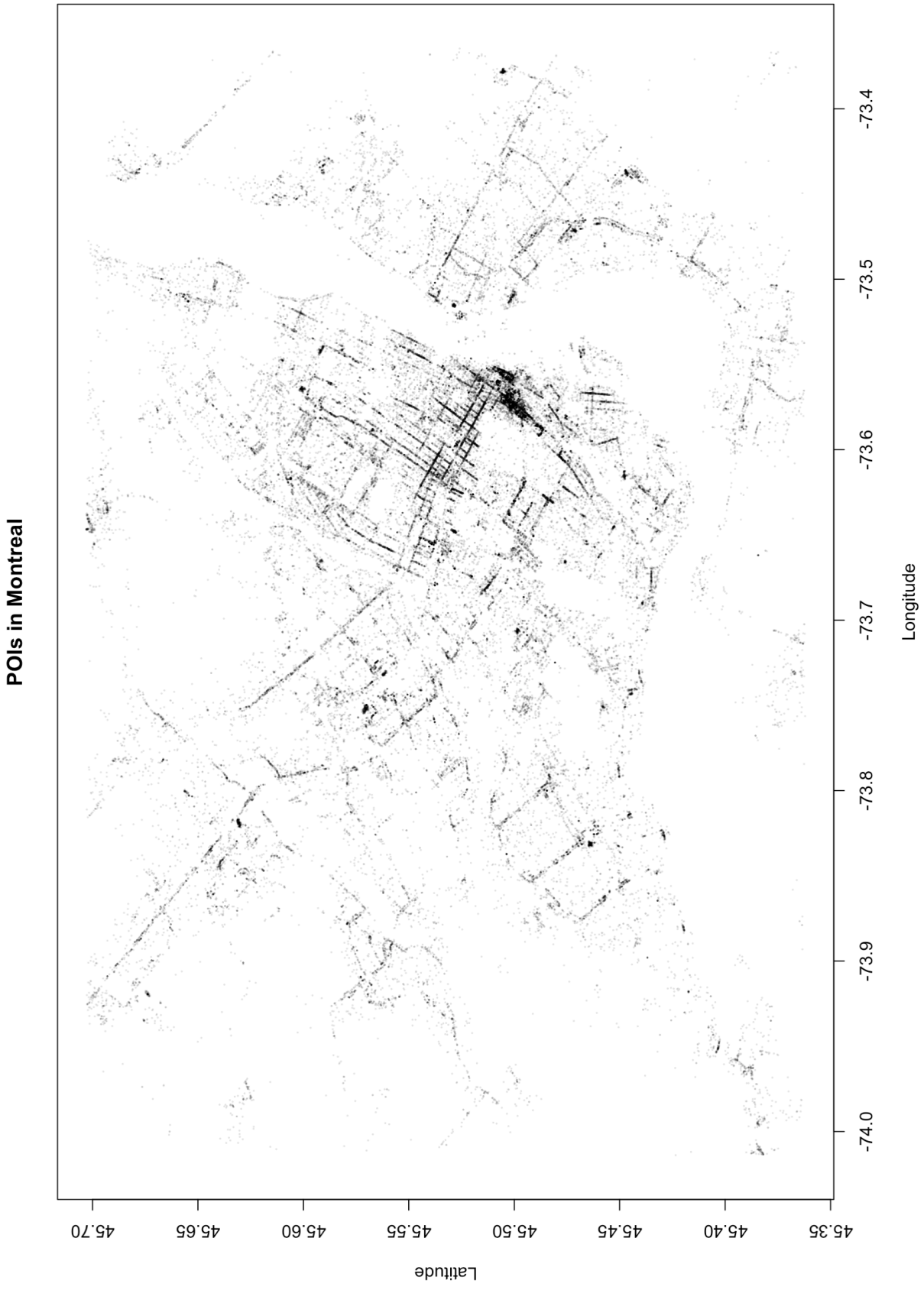


Figure 1: All the POIs in the Greater Montreal region, with each black point representing one POI. To emphasize density, all points have an opacity of 0.1 regardless of the weight of the POI it represents. Generated using the `plot` R function.

Table 1: Weights of Google Places POI Types

POI Type	Weight	POI Type	Weight	POI Type	Weight
accounting	0	establishment	0	parking	0.4
airport	1	fire_station	0	pet_store	0.2
amusement_park	0.8	florist	0.2	pharmacy	0.2
aquarium	0.8	funeral_home	0.1	physiotherapist	0.4
art_gallery	0.4	furniture_store	0.1	plumber	0
atm	0.4	gas_station	0	point_of_interest	0
bakery	0.25	gym	0.6	police	0
bank	0.2	hair_care	0.4	political	0
bar	0.6	hardware_store	0.2	post_office	0.05
beauty_salon	0.25	hindu_temple	0.1	real_estate_agency	0.05
bicycle_store	0.6	home_goods_store	0.25	restaurant	0.4
book_store	0.6	hospital	1	roofing_contractor	0
bowling_alley	0.4	insurance_agency	0.1	route	0
bus_station	0	intersection	0	rv_park	0
cafe	0.4	jewelry_store	0.2	school	0.6
campground	0.2	laundry	0	shoe_store	0.2
car_dealer	0.1	lawyer	0.25	shopping_mall	0.6
car_rental	0.2	library	0.6	spa	0.2
car_repair	0.4	liquor_store	0.2	stadium	0.8
car_wash	0	local_government_office	0.4	storage	0.15
casino	0.8	locality	0	store	0.25
cemetery	0.1	locksmith	0	street_address	0
church	0.1	lodging	0.6	street_number	0
city_hall	0.4	meal_delivery	0	subway_station	0
clothing_store	0.6	meal_takeaway	0.1	synagogue	0.1
convenience_store	0.1	mosque	0.1	taxi_stand	0.4
courthouse	0.2	movie_rental	0.1	train_station	0
dentist	0.1	movie_theater	0.4	transit_station	0
department_store	0.6	moving_company	0	travel_agency	0.15
doctor	0.6	museum	0.8	university	1
electrician	0.1	night_club	0.8	veterinary_care	0.25
electronics_store	0.2	painter	0.05	zoo	0.4
embassy	0.6	park	0.6		

This table contains all of the possible POI types provided by the Google Places API. I heuristically assigned a weight to each type depending on how important I felt it was to have each type of POI connected to public transit. It goes without saying that these weightings are subjective and could be modified on a case-by case-basis.

This can be accomplished using a clustering algorithm. In graphical data analysis, clustering algorithms are used to identify data points whose given properties are similar and group them into clusters. Because we will be grouping POIs that are geographically close to each other, the properties in which we will look for similarity are latitude and longitude. Therefore, the type of clustering algorithm we need to use is a *density-based* one. A popular density-based clustering algorithm that we will use is DBSCAN (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise), developed by Ester, Kriegel, Sander, and Xu (1996). This algorithm requires the user to specify two parameters: a maximum radius ε in which to search for neighbouring points, and a threshold m_p that is the minimum number of points required to form a cluster.

3.1 The DBSCAN Algorithm

The DBSCAN algorithm works as follows (Ester et al., 1996):

- Let $S_\varepsilon(\mathbf{p})$ be the neighbouring set of points from the set of all points D whose distance from $\mathbf{p}$ is less than or equal to ε.

$$S_\varepsilon(\mathbf{p}) = \{\mathbf{q} \in D \,|\, \text{dist}(\mathbf{p}, \mathbf{q}) \leq \varepsilon\}$$

- Label point $\mathbf{p}$ as *directly density-reachable* from a point $\mathbf{q}$ if $\mathbf{p}$ is a neighbouring point of $\mathbf{q}$ and there are at least m_p points in the neighbouring set of $\mathbf{q}$.

$$\mathbf{p} \in S_\varepsilon(\mathbf{q}) \quad \text{and} \quad |\, S_\varepsilon(\mathbf{q}) \,| \geq m_p$$

- Label point $\mathbf{p}$ as *density-reachable* from a point $\mathbf{q}$ if there is a chain of points $\mathbf{p_1}$, $\mathbf{p_2}$, …, $\mathbf{p_n}$ given $\mathbf{p_1} = \mathbf{q}$ and $\mathbf{p_n} = \mathbf{p}$ such that $\mathbf{p_{i+1}}$ is directly density-reachable from $\mathbf{p_i}$.
- Label point $\mathbf{p}$ as *density connected* to a point $\mathbf{q}$ if there is a point $\mathbf{o}$ such that $\mathbf{p}$ and $\mathbf{q}$ are both density-reachable from $\mathbf{o}$.
- A cluster C is a non-empty subset of D such that, $\forall\, \mathbf{p}, \mathbf{q}$, if $\mathbf{p} \in C$ and $\mathbf{q}$ is density-reachable from $\mathbf{p}$, then $q \in C$, and that $\forall\, \mathbf{p}, \mathbf{q} \in C$, $\mathbf{p}$ is directly-connected to $\mathbf{q}$.
- Label any point $\mathbf{z}$ not in a cluster C as *noise*.

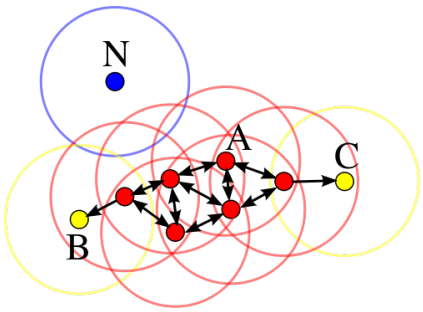


Figure 2: A DBSCAN cluster given $m_p = 4$. The red points are all density-reachable from each other, and are directly density-reachable from each other if their centres lie within the radius ε from the core of another red point. Points A and B are density-connected to each other. Point N is noise. Taken from Wikipedia user Chire (2011; CC BY-SA 3.0).

Figure 2 illustrates how the DBSCAN algorithm works.

3.2 Choosing ε and m_p

The values for ε and m_p have a significant influence on the suggested transit network. By varying these parameters, different types of transit systems can be suggested with different distances between the stations: larger values for the parameters could be selected for determining ideal locations for intercity rail stations over large geographical regions, smaller values for commuter rail stations into a city core, smaller values still for an underground metro system, and even smaller values for tram lines or bus routes.

Ultimately, we will set the values for ε and m_p heuristically:

$$\varepsilon = 0.0003, \quad m_p = 65$$

Note that the value for ε is set in degrees of latitude/longitude. As we will soon see, these values yield a system with distances between stations

akin to a metro system.

Given these parameters, 165 high-density clusters are identified in the Greater Montreal region.

3.3 Cluster Centroid Calculation

The DBSCAN algorithm will simply assign a point $\mathbf{p}$ to a cluster C. It is up to us, however, to do something with these assignments. We will use these assignments to calculate the centroid of the cluster using a weighted average of the POIs in the cluster. If the POI $\mathbf{p_i}$ is an ordered pair of latitude φ_i and longitude λ_i ($\mathbf{p_i} = (\varphi_i, \lambda_i)$), γ_i is the weight of POI $\mathbf{p_i}$, and there are N_p POIs in the cluster, then the centroid $\boldsymbol{\kappa_C}$ of cluster C is given by

$$\boldsymbol{\kappa_C} = \frac{\sum_{i=1}^{N_p} \gamma_i \mathbf{p_i}}{\sum_{i=1}^{N_i} \gamma_i}$$

3.4 Cluster Weight Calculation

Next, each cluster needs to be assigned a weight. Despite all being clusters with high densities of POIs, some clusters would benefit more from becoming a station than others. For example, a cluster dense with POIs such as laundromats, convenience stores, real estate agencies, and hairdressers will almost certainly see less passenger traffic than a cluster containing a movie theatre, a university, a library, and a shopping centre.

This being said, public transit users are not necessarily homogeneous; they can be hard to predict. While the rule of thumb used in the urban planning industry is that passengers are willing to walk 400 m to bus stops, and 800 m to railway stations (El-Geneidy, Grimsrud, Wasfi, Tétreault, & Surprenant-Legault, 2013, p. 2), some passengers may be more likely to walk a certain distance after exiting a station than others.

As such, I do not think the weight of a cluster would simply be the sum of all the POIs in that cluster. Instead, the weight of a cluster needs to also take into account the distance of the POI from center of the cluster (where the station would hypothetically be) and the likelihood that a passenger would walk that distance in order to get to that POI from the station, assuming they do not transfer to another mode of transit such as a bus or a bicycle.

El-Geneidy et al. studied the walking habits of public transit users in Montreal. They found that, as suggested by the rule of thumb, passengers are generally willing to walk further to reach faster modes of transit like trains and metros than they are to reach bus stations. While the number of passengers willing to walk a given distance to most bus stations offered by the regional providers saw an approximately exponential decline as the distance increased, the trend for train and metro passengers resembled a Gaussian distribution.

Gaussian distributions are often used in the social sciences to model human behaviour (Simonton, 2008). Also known as the normal distribution, Gaussian distributions have the form

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad \mu \in \mathbb{R},\ \sigma^2 > 0$$

where μ is the mean of the dataset and σ is the standard deviation of the dataset. El-Geneidy et al. suggested that the standard deviation of the dataset for metro users in Montreal was 297.37 m, and so σ was hence defined. The mean μ was set to 0 m so that POIs that required little walking were weighted heavily, whereas POIs far from the cluster's centroid were penalized.

Finally, the weight Γ_C of cluster C could be calculated:

$$\Gamma_C = \sum_{i=1}^{N_p} \gamma_i \mathcal{N}(\mathrm{dist}(\mathbf{p_i}, \boldsymbol{\kappa_C}) \mid \mu, \sigma^2)$$

where N_p is the number of POIs in the cluster, γ_i is the weight of POI $\mathbf{p_i}$ in the cluster, $\boldsymbol{\kappa_C}$ is the centroid of cluster C, $\mathrm{dist}(\mathbf{p_i}, \boldsymbol{\kappa_C})$ is the distance between $\mathbf{p_i}$ and $\boldsymbol{\kappa_C}$, and μ and σ are as previously defined.

By multiplying the weight of each POI in a cluster by the normal distribution suggested by El-Geneidy et al., the weight of each POI would be considered relative to how much traffic it would actually attract from the station given the distance passengers would have to walk to get there. This being said, it is crucial to represent this distance in metres, as the standard deviation suggested by El-Geneidy et al. is in metres. However, our coordinate system is in degrees of latitude and longitude, and not in metres. As such, we need to use the haversine formula to approximate distances in metres between our POIs.

3.4.1 The Haversine Formula

The haversine formula is used for determining the great-circle distance between two points on a sphere. Because Earth is practically spherical, the haversine formula can be used to approximate distances between coordinate pairs on the Earth's surface. The haversine *function* $\text{hav}(\theta)$ is defined as follows (Chopde & Nichat, 2013):

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

The haversine formula relates the distance d between two points on the sphere, the radius R of the sphere in question, the latitudes of the two points φ_1, φ_2, and the longitudes of the two points λ_1, λ_2 following the equation (Chopde & Nichat, 2013)

$$\begin{aligned}\text{hav}\left(\frac{d}{R}\right) &= \text{hav}(\varphi_2 - \varphi_1) + \\ &\quad + \cos(\varphi_1)\cos(\varphi_2)\text{hav}(\lambda_2 - \lambda_1)\end{aligned}$$

which can be rearranged such that d is isolated:

$$\begin{aligned}d &= 2R\,\text{asin}\left(\sqrt{\begin{aligned}&\text{hav}(\varphi_2 - \varphi_1) + \\ &+ \cos(\varphi_1)\cos(\varphi_2)\text{hav}(\lambda_2 - \lambda_1)\end{aligned}}\right) \\ &= \text{dist}\left((\varphi_1, \lambda_1), (\varphi_2, \lambda_2)\right)\end{aligned}$$

3.5 Map of Cluster Centroids

Figure 3 displays the centroids of the 165 identified clusters, coloured by their respective weights. Each point represents a potential station placement in our network.

4 Identifying Potential Lines

Next, we need to identify where placing a line would make sense. Logically, a transit line should follow a thoroughfare where there are many POIs as opposed to, say, zigzag nonsensically across the city. This also makes it easier to plan the routes that make up the network.

4.1 Using Gaussian Mixture Models

To achieve this, we can perform another clustering of the high-density clusters we already found. Doing so will allow us to identify pockets of cluster centroids in the region. This will be done using Gaussian mixture models (GMMs) as laid out by Sanderson and Curtin (2017). In a nutshell, GMMs iteratively adapt N_G D-dimensional Gaussian distributions to the D-dimensional dataset in order to determine *probabilistically* which cluster a point most likely belongs to. This means that a point technically could belong to all N_G Gaussians simultaneously, but there is one Gaussian it most probably belongs to (and usually by far). The probability that a point $\mathbf{x}$ (represented as a D-dimensional vector) is in GMM cluster[1] G is given by

$$P(\mathbf{x} \mid \Theta) = \sum_{G=1}^{N_G} \rho_G \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_G}, \boldsymbol{\Sigma_G})$$

where $\Theta = \{\rho_G, \boldsymbol{\mu_G}, \boldsymbol{\Sigma_G}\}_{G=1}^{N_G}$ with the constraints $\sum_{G=1}^{N_G} \rho_G = 1, \rho_G \geq 0$, where ρ_G is the weight of cluster G (not to be confused with Γ_C, representing the weight of DBSCAN cluster C). The Gaussian distribution $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu_G}, \boldsymbol{\Sigma_G})$ in D-dimensions given the mean $\boldsymbol{\mu}$ (a D-dimensional vector) and the covariance matrix $\boldsymbol{\Sigma}$ (a $D \times$

[1] To avoid confusion between high-density clusters identified by the DBSCAN algorithm and the clusters *of those clusters* found by the GMM algorithm, the latter will henceforth be identified as GMM clusters.

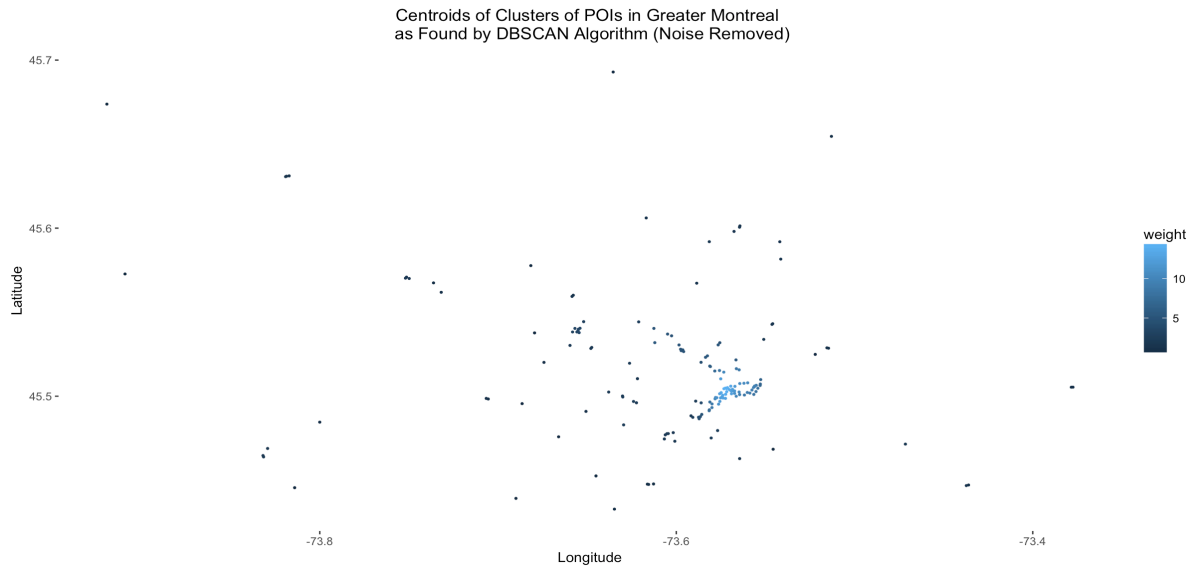


Figure 3: The centroids of the clusters found by the DBSCAN algorithm. The shade of blue of each point represents the weight of each cluster. Note that clusters with heavier weights are focused around the downtown core. Generated using the `ggplot2` R package.

D matrix whose particular nature is explained in detail in the next section) is (Sanderson & Curtin, 2017)

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}}{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}}$$

In order to best fit N_G Gaussian distributions to a dataset, Θ needs to be initialized and then iteratively adapted, achieving a better and better fit with each iteration. The `mclust` R package (Scrucca, Fop, Murphy, & Raftery, 2016) handles this in a similar way to that outlined by Sanderson and Curtin. Ultimately, the math behind this process, while interesting, is fairly insignificant in the planning of transit networks. In short, we substitute the two-dimensional position vector $\boldsymbol{\kappa}$ for $\mathbf{x}$ in the preceding equations so that we can cluster our centroids. Essentially, once we identify clusters of centroids, we can look at the shape of that cluster to identify where it makes sense to run a line.

4.1.1 Covariance Matrices

A covariance matrix is essentially a matrix representing the variance of the distribution of points in a given dataset in multiple dimensions. In one-dimensional data, variance, represented by the symbol σ^2, is often simply defined as the average squared distance from the mean μ of a dataset $X = \{x_1, x_2, \ldots, x_n\}$:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

For multidimensional data, the element $\text{cov}(j,k)$ of the covariance matrix $\boldsymbol{\Sigma}$ represents the variability of a dataset between the j^{th} and k^{th} dimensions, and the element σ_j^2 represents the variance of the dataset in the j^{th} dimension. Note that $\text{cov}(j,k) = \text{cov}(k,j)$. For example, for a four-dimensional dataset of dimensions w, x, y, and z, the covariance matrix $\boldsymbol{\Sigma}$ would be

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_w^2 & \text{cov}(w,x) & \text{cov}(w,y) & \text{cov}(w,z) \\ \text{cov}(x,w) & \sigma_x^2 & \text{cov}(x,y) & \text{cov}(x,z) \\ \text{cov}(y,w) & \text{cov}(y,x) & \sigma_y^2 & \text{cov}(y,z) \\ \text{cov}(z,w) & \text{cov}(z,x) & \text{cov}(z,y) & \sigma_z^2 \end{bmatrix}$$

In our case, using GMM clusters, we only have two dimensions to consider: the latitude and longitude of the centroid of the high-density cluster found by the DBSCAN algorithm, centred at $\boldsymbol{\kappa}$. The similarity $\delta(\boldsymbol{\kappa}, \boldsymbol{\kappa_i})$ of a point $\boldsymbol{\kappa}$ *to be clustered* to a point $\boldsymbol{\kappa_i}$ *already in a GMM cluster* centred at $\boldsymbol{\upsilon}$ can be found by squaring the scalar product of $\boldsymbol{\kappa}$ and $\boldsymbol{\kappa_i}$ relative to $\boldsymbol{\upsilon}$ (Rojas, 2009):

$$\delta(\boldsymbol{\kappa}, \boldsymbol{\kappa_i}) = \Big((\boldsymbol{\kappa} - \boldsymbol{\upsilon})^\top (\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})\Big)^2$$

By repeating this computation for each POI $\boldsymbol{\kappa_i}$ in the GMM cluster in question ($i = 1, \ldots, N_i$) and averaging the results, we can understand where $\boldsymbol{\Sigma}$ comes from:

$$\begin{aligned}
\Delta(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) &= \frac{1}{N_i} \sum_{i=1}^{N_i} \Big((\boldsymbol{\kappa} - \boldsymbol{\upsilon})^\top (\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})\Big)^2 \\
&= \frac{1}{N_i} \sum_{i=1}^{N_i} (\boldsymbol{\kappa} - \boldsymbol{\upsilon})^\top (\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})(\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})^\top (\boldsymbol{\kappa} - \boldsymbol{\upsilon}) \\
&= (\boldsymbol{\kappa} - \boldsymbol{\upsilon})^\top \left(\frac{1}{N_i} \sum_{i=1}^{N_i} (\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})(\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})^\top \right) (\boldsymbol{\kappa} - \boldsymbol{\upsilon}) \\
&= (\boldsymbol{\kappa} - \boldsymbol{\upsilon})^\top \boldsymbol{\Sigma} (\boldsymbol{\kappa} - \boldsymbol{\upsilon})
\end{aligned}$$

Therefore, $\boldsymbol{\Sigma} = \frac{1}{N_i} \sum_{i=1}^{N_i} (\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})(\boldsymbol{\kappa_i} - \boldsymbol{\upsilon})^\top$.

4.2 Identifying Potential Transit Lines

Finally, the Gaussian distributions have been determined. Typically, the results of GMM clustering are illustrated using variance ellipses. A variance ellipse is essentially a visual representation of a covariance matrix.

4.2.1 Eigenvectors and Eigenvalues

In order to understand variance ellipses, one first must understand *eigenvalues* and *eigenvectors*. Essentially, if there exists some square matrix $\mathbf{A}$, some scalar t, and some non-zero vector $\mathbf{v}$, then t is deemed an eigenvalue and $\mathbf{v}$ is deemed an eigenvector if the equation

$$\mathbf{A}\mathbf{v} = t\mathbf{v}$$

is satisfied (Weisstein, 2018). This equation can be rewritten as

$$(\mathbf{A} - t\mathbf{I})\mathbf{v} = 0$$

where $\mathbf{I}$ is the identity matrix. This will only have a solution if

$$|\mathbf{A} - t\mathbf{I}| = 0$$

Note that the eigenvectors of a matrix are linearly independent and are often normalized.

Expanding the matrix multiplication embedded in $\boldsymbol{\Sigma}$ reveals that

$$\boldsymbol{\Sigma} = \frac{1}{N_i} \sum_{i=1}^{N_i} \begin{bmatrix} (\kappa_{\varphi_i} - \upsilon_\varphi)^2 & (\kappa_{\varphi_i} - \upsilon_\varphi)(\kappa_{\lambda_i} - \upsilon_\lambda) \\ (\kappa_{\varphi_i} - \upsilon_\varphi)(\kappa_{\lambda_i} - \upsilon_\lambda) & (\kappa_{\lambda_i} - \upsilon_\lambda)^2 \end{bmatrix}$$

Note that the elements in the main diagonal of $\boldsymbol{\Sigma}$ are equal to the latitude and longitude variances σ_φ^2 and σ_λ^2, respectively. The off-diagonal gives the covariance between the latitude and longitude (φ and λ). Also note that the two elements in the off-diagonal are identical.

Finally, to solve for the eigenvectors $\mathbf{v}$ and eigenvales t of $\boldsymbol{\Sigma}$, we need to solve the equation

$$\boldsymbol{\Sigma}\mathbf{v} = t\mathbf{v}$$

where t is as defined in the preceding equation. D-dimensional covariance matrices yield D linearly independent eigenvectors and their respective eigenvalues. The `eigen()` R function is able to quickly solve for them for us. Knowing these, the variance ellipses can be plotted for each GMM cluster, as shown in Figure 4.

4.2.2 Identifying Potential Termini

Note that, in Figure 4, the longest of the scaled eigenvectors in the variance ellipse is aligned with the main axis of the distribution of the points in the cluster its ellipse represents. We will use this property to identify the termini of the lines: the termini will be the two closest points in each GMM cluster to the points of intersection between the line representing the longest of the eigenvectors in the variance ellipse

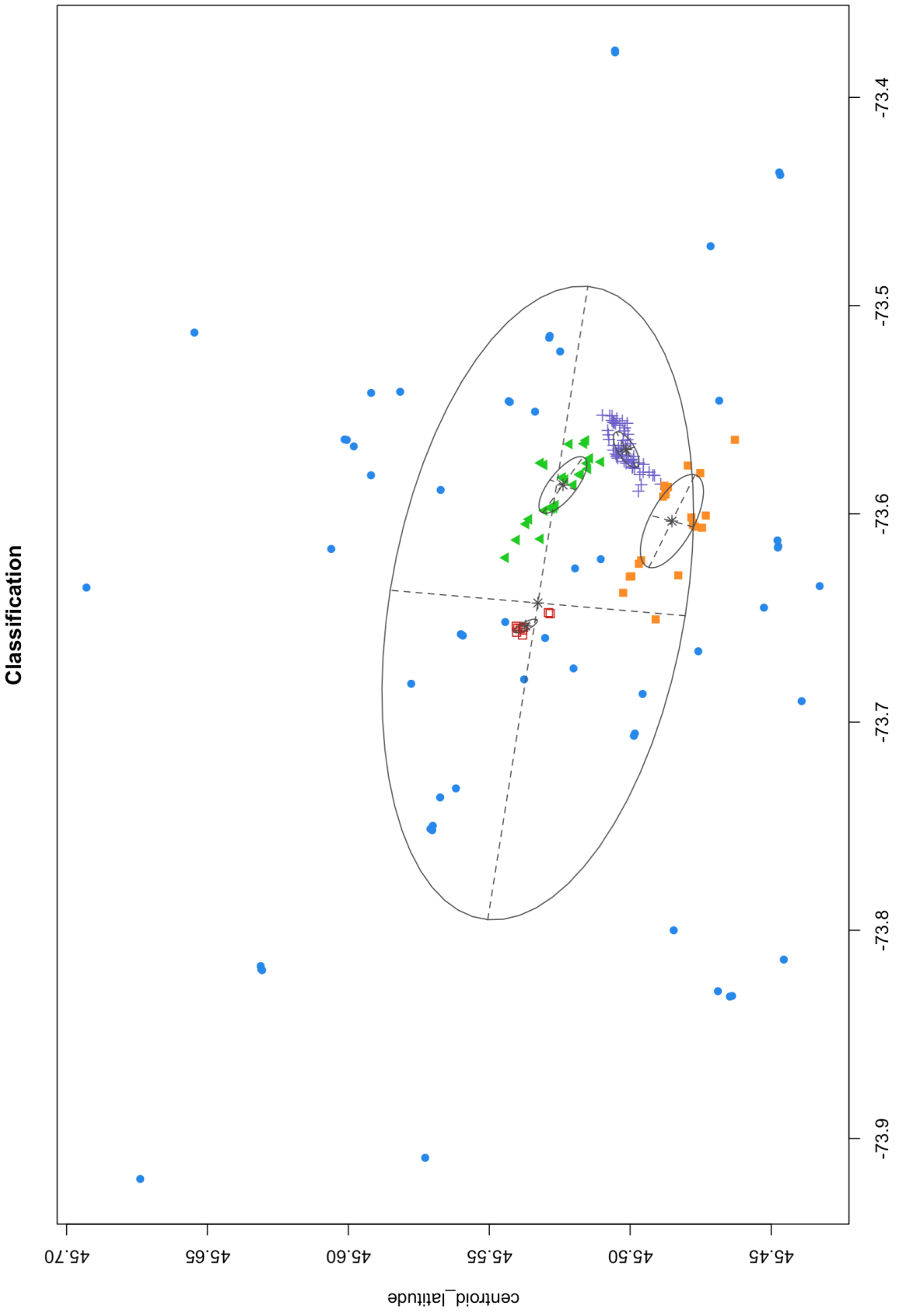


Figure 4: The 5 variance ellipses resulting from the GMM clustering. Each ellipse illustrates a GMM cluster. The smaller the ellipse, the denser the cluster. The dashed lines illustrate the two scaled eigenvectors that define each ellipse. Generated using the `mclust` R package.

and the geographical region being analyzed (in this case, Greater Montreal).

The eigenvectors provided by `eigen()` R function are normalized, so we need to rescale them for our purposes. Let $\mathbf{v_{G_1}}$ and $\mathbf{v_{G_2}}$ be the scaled eigenvectors of the GMM cluster G (Wold, Esbensen, & Geladi, 1987):

$$\mathbf{v_{G_1}} = \hat{\mathbf{v}}_{\mathbf{G_1}}\sqrt{t_{G_1}}, \quad \mathbf{v_{G_2}} = \hat{\mathbf{v}}_{\mathbf{G_2}}\sqrt{t_{G_2}}$$

where $\hat{\mathbf{v}}_{\mathbf{G_1}}$ and $\hat{\mathbf{v}}_{\mathbf{G_2}}$ are the normalized eigenvectors of the covariance matrix and t_{G_1} and t_{G_2} are the respective eigenvalues. Now that the eigenvectors are rescaled, we can determine the slope a of the line representing the main axis along which points are found in the GMM cluster:

$$B = \underset{k=1,2}{\operatorname{argmax}} ||\mathbf{v_{G_k}}||, \quad a = \frac{\upsilon_{G_B\varphi}}{\upsilon_{G_B\lambda}}$$

Note that the operator $\underset{x}{\operatorname{argmax}}\, f(x)$ represents the value of x for which $f(x)$ is maximum.

Given that the point slope form of a line is $y - y_1 = a(x - x_1)$, the two points in the GMM cluster G closest to those where the line of slope a leaves the region being examined would be the termini of the line because they are at opposite ends of the cluster. If the northwest corner of the region in question is given by the vector $(\Phi_{\mathrm{N}}, \Lambda_{\mathrm{W}})$ and the southeast corner is given by the vector $(\Phi_{\mathrm{S}}, \Lambda_{\mathrm{E}})$, the latitude φ_e represents the latitude of the endpoint where the line of slope a leaves the region being examined:

$$\varphi_{e_N} = a(\Lambda_{\mathrm{W}} - \upsilon_\lambda) + \upsilon_\varphi$$

$$\varphi_{e_S} = a(\Lambda_{\mathrm{E}} - \upsilon_\lambda) + \upsilon_\varphi$$

where $\boldsymbol{\upsilon}$ is the centroid of the GMM cluster being analyzed.

The indices of the two termini τ_{NW} and τ_{SE} are therefore given by

$$\tau_{\mathrm{NW}} = \underset{i=1,\ldots,N_i}{\operatorname{argmin}} \operatorname{dist}\left((\varphi_{e_N}, \Lambda_{\mathrm{W}}), \boldsymbol{\kappa_i}\right)$$

$$\tau_{\mathrm{SE}} = \underset{i=1,\ldots,N_i}{\operatorname{argmin}} \operatorname{dist}\left((\varphi_{e_S}, \Lambda_{\mathrm{E}}), \boldsymbol{\kappa_i}\right)$$

Note that the operator $\underset{x}{\operatorname{argmin}}\, f(x)$ represents the value of x for which $f(x)$ is minimum.

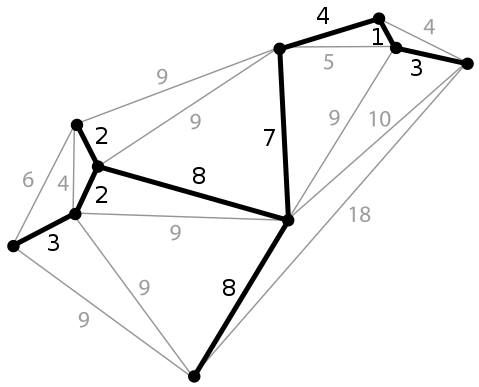


Figure 5: The minimum spanning stree of an example network of nodes and weighted edges. Taken from Wikipedia user Dcoetzee (2005; public domain).

5 Linking Stations in a Line

Next, points in each GMM cluster can be linked to form the most efficient route. Note that not all points in the cluster will become part of the line. To do this, we will turn to graphs. A graph is a network composed of nodes connected by edges. These edges may be weighted, as is the case with our lines: the weight of each edge is a function of the sum of the weights of the clusters (the nodes) that edge connects. To begin, each cluster is connected to all the other clusters in the network this way. However, we want to find the most efficient layout for our transit system. So, we will use minimum spanning trees that seek to connect all the nodes in a set to each other such that the sum of the cost of the edges connected is minimized.

5.1 Preparing the Cost Matrix

The `igraph` R package has a built in function that finds the minimum spanning tree. The function requires a square cost matrix $\boldsymbol{\Omega}$ as a param-

eter of the form

$$\mathbf{\Omega} = \begin{bmatrix} 0 & \omega_{12} & \cdots & \omega_{1i} \\ \omega_{21} & 0 & \cdots & \omega_{2i} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{i1} & \omega_{i2} & \cdots & 0 \end{bmatrix}$$

where ω_{jk} is the cost of the edge connecting nodes j and k. Note that elements along the main diagonal of the cost matrix are 0 as there is no cost to connecting a station to itself.

5.1.1 Designing the Cost Function

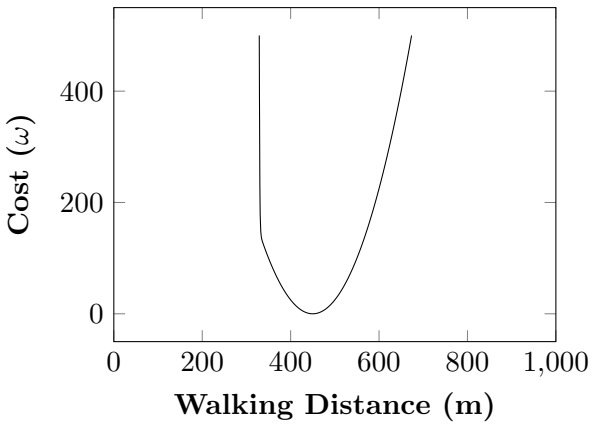
The nature of the function $\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$ is significant in the output of the program. Our design will severely penalize the connection of points that are significantly closer than one and a half times farther apart the standard deviation σ of the distance passengers were willing to walk after exiting the station, while only gradually penalizing stations as they grow farther apart from one and a half times σ. Multiplying σ by $\frac{3}{2}$ is necessary because a passenger might walk a distance about σ metres from the station, so to avoid overlapping walkability-radii, the stations needed to be separated a bit more. My cost function is:

$$\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k}) = \frac{1}{100}\left(d - \frac{3}{2}\sigma\right)^2 + \frac{1 + \frac{1}{e^{d-\frac{3}{2}\sigma}}}{10^{10\left(\Gamma_{C_j}+\Gamma_{C_k}\right)}},$$

$$d = \text{dist}(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$$

The quadratic term of $\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$ serves to gradually penalize the connecting of stations that are progressively further apart than $\frac{3\sigma}{2}$. The exponential term serves to severely penalize the connection of stations that are closer together than $\frac{3\sigma}{2}$. The sum $\Gamma_{C_j} + \Gamma_{C_k}$ in the coefficient of the exponential term serves to fine tune where the exponential penalization begins. This is because it might be worth connecting two potential stations with very high densities even if they are slightly closer together than $\frac{3\sigma}{2}$ so as to not leave one station bear the brunt of high passenger traffic.

Figure 6: Plot of the cost function $\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$ given $\sigma = 297.37$ and $\Gamma_{C_j} + \Gamma_{C_k} = 5$, akin to the combined weights of two relatively dense neighbouring clusters.



Two plots of $\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$ with two different values for $\Gamma_{C_j} + \Gamma_{C_k}$ are available in Figures 6 and 7, respectively, to illustrate the effect of the $\Gamma_{C_j} + \Gamma_{C_k}$ parameter.

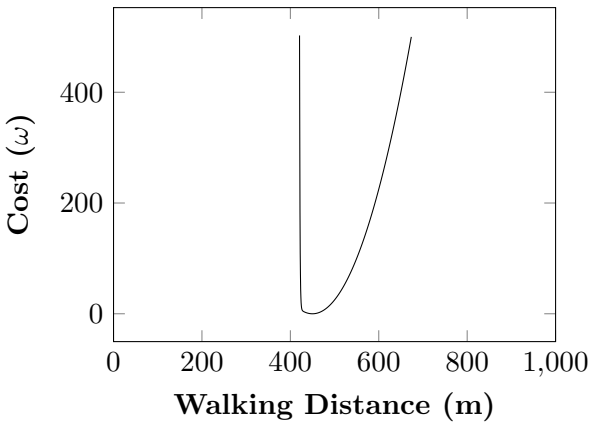
5.2 Connecting the Dots with Prim's Algorithm

Finally, we are ready to connect the dots, so to speak—to form metro lines and suggest a public transit network given only the POIs in a region.

The `igraph` R package uses Prim's algorithm (Prim, 1957) to find the minimum spanning tree efficiently. The algorithm is simple: an arbitrary node seeds the tree, and the algorithm searches for the edge connected to the node with the lowest cost. The node at the other end of that edge is also added to the tree. Then, the process is repeated for the node at the other end of the edge, connecting it to a node not yet part of the tree. The process is repeated until all the nodes are part of the tree. No node may be connected to more than two other nodes. See Figure 8 for an example of Prim's algorithm connecting a set of nodes.

Once the minimum spanning tree is determined, the transit line follows the least costly path

Figure 7: Plot of the cost function $\omega(\boldsymbol{\kappa_j}, \boldsymbol{\kappa_k})$ given $\sigma = 297.37$ and $\Gamma_{C_j} + \Gamma_{C_k} = 1$, akin to the combined weights of two relatively sparse neighbouring clusters.



within the minimum spanning tree that connects the two termini we identified in Section 4.2.2.

5.2.1 Filtering Irrelevant Stations

There are only a few loose ends to tie up. First of all, a verification is added to the program to ensure that each line has a minimum of m_s stations. This filters out the odd GMM cluster of a few insignificant stations that would never justify having their own line in the real world, often far away from the rest of the network.

Furthermore, another verification is added that filters out lines formed from extremely large GMM clusters, such as the blue cluster in Figure 4. This is done by comparing the variance of the magnitude of the scaled eigenvector of that GMM cluster with the mean magnitude for all clusters. If the variance surpasses a certain threshold $\sigma^2_{\max}$, the line is ignored. In other applications, this filtering process could be used to distinguish between lines for different modes of transit. For example, while I am trying to identify lines for a metro system, lines filtered out for being too spread out could make for good commuter rail lines which cover more distance, with more separation between stations.

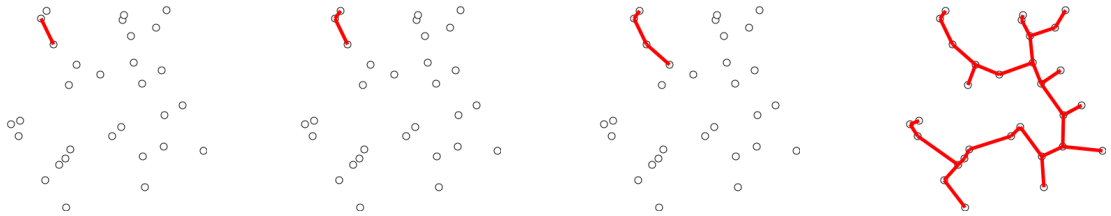
6 Conclusions

Ultimately, the program as described suggests the network depicted in Figure 9. The network actually resembles that of the core of Montreal's existing metro system, which is overlaid in Figure 9 and also included separately in Figure 10 of Appendix A.

6.1 Suggestions for Improvement & Further Work

It goes without saying the output of the program can still be significantly improved upon. First of all, using the haversine as the distance function to determine the walkability-radii assumes that pedestrians are not, in fact, limited to walking on sidewalks but are able to walk through buildings. This is certainly not the case, and using a Manhattan distance function given the road network of the city could offer a more accurate alternative. Furthermore, many of the stations in the network are still very close together despite the high cost penalty for this being the case. This suggests that there might be a more effective way to incorporate the suggested walkability radius of 300 m from El-Geneidy et al. into the program than how it is being done currently. Similarly, in some cases, the minimum spanning tree would cross over itself in order to avoid high penalties for linking stations that are too close together. It would not make sense in the real world for a metro line to do that. Also, a suggestion for where to link two separate metro lines in the proposed system is not currently offered.

The most significant flaw in this methodology, however, is that it does not consider the fact that public transit users need to somehow reach the network from their homes. The suggested network in the output of the program solely runs through dense commercial neighbourhoods; branches of lines extending into residential neighbourhoods as with the Green and Orange lines in Montreal's existing metro system are notably missing from the output of the program.



(a) After 1 iteration (b) After 2 iterations (c) After 3 iterations (d) After 29 iterations

Figure 8: Prim's algorithm connecting a sample of nodes. The cost of each connection is the Euclidean distance between nodes. Taken from Wikipedia user Shiyu Ji (2016; CC BY-SA 4.0).

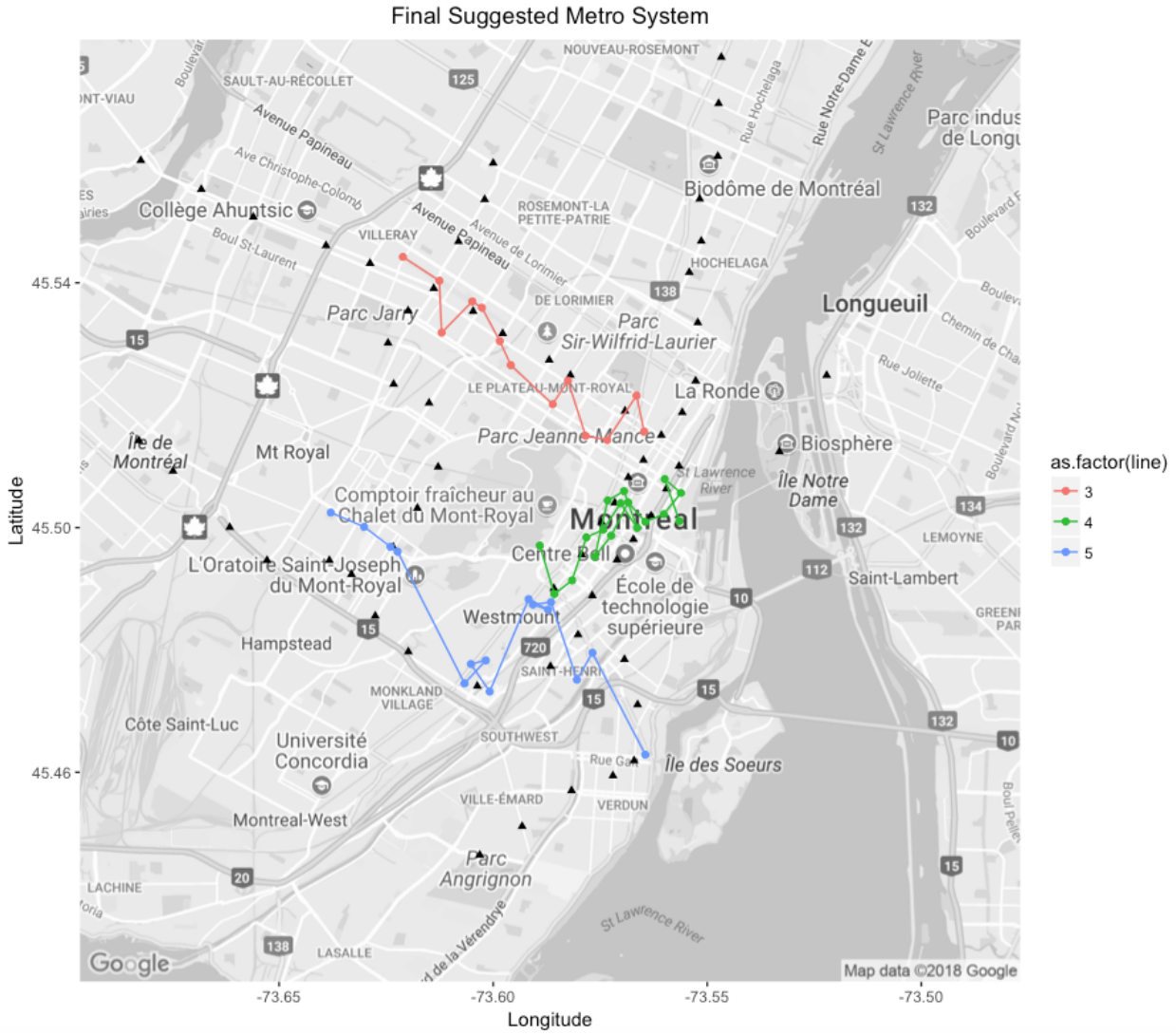


Figure 9: The final output of the program containing the suggested layout for the metro network given the input parameters. The black triangles indicate the locations of existing metro stations in Montreal, for comparison. Generated using the `ggmap` R package.

References

Ankerst, M., Breunig, M. M., Kriegel, H.-P., & Sander, J. (1999). *Optics: Ordering points to identify the clustering structure.* Oettingenstr. 67, D-80538 München, Germany: Institute for Computer Science, University of Munich.

Chopde, N. R., & Nichat, M. K. (2013, April). Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, *1*(2), 298-302.

Csárdi, G., & Nepusz, T. (2017, July). Package 'igraph' (1.1.2 ed.) [Computer software manual].

El-Geneidy, A., Grimsrud, M., Wasfi, R., Tétreault, P., & Surprenant-Legault, J. (2013, October). New evidence on walking distances to transit stops: Identifying redundancies and gaps using variable service areas. *Transportation*, *41*(1), 193-210.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise.* Oettingenstr. 67, D-80538 München, Germany: Institute for Computer Science, University of Munich.

Goldthwaite, W. M. (1892). *Goldthwaite's geographical magazine* (Vol. 3-4). New York City.

Lilly, J. (2016). *The variance ellipse.*

Places api web service [Computer software manual]. (2018, January).

Prim, R. C. (1957, November). Shortest connection networks and some generalizations. *Bell System Technical Journal*, *36*(6), 1389-1401.

Rojas, R. (2009, January). *The secret life of the covariance matrix.*

Sanderson, C., & Curtin, R. (2017). An open source c++ implementation of multi-threaded gaussian mixture models, k-means and expectation maximisation. In *The 11th international conference on signal processing and communication systems (icspcs 2017).*

Scrucca, L., Fop, M., Murphy, T. B., & Raftery, A. E. (2016, August). mclust 5: Clustering, classification and density estimation using gaussian finite mixture models [Computer software manual].

Simonton, D. K. (2008). Distribution, normal. In W. A. D. Jr. (Ed.), *International encyclopedia of the social sciences.* Gale.

Weisstein, E. W. (2018). *Eigenvector.* MathWorld.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*(2), 37-52.

A Montreal's Existing Metro System

Figure 10: Artistic map of Montreal's metro system. Taken from the Sociéte de Transport de Montréal.

B Program Code

```
library(dbscan)
library(ggplot2)
library(mclust)
library(geosphere)
library(proxy)
library(igraph)
library(ggmap)


# Domain remapping
fit.domain <- function(n, a, b, c, d) {
  (n-a)/(b-a) * (d-c) + c
}

# Returns latitude of coordinate distance metres to the direction of the current latitude
distance.to.latitude <- function(starting_latitude, distance, direction) {
  if (direction == "N") { coefficient <- 1 }
  if (direction == "S") { coefficient <- -1 }
  return(starting_latitude + (distance/111319.9) * coefficient )
}

# Returns longitude of coordinate distance metres to the direction of the current latitude & longitude
distance.to.longitude <- function(starting_longitude, starting_latitude, distance, direction) {
  if (direction == "E") { coefficient <- 1 }
  if (direction == "W") { coefficient <- -1 }
  divisor <- cos( starting_latitude * (pi/180) ) * 111319.9
  return(starting_longitude + (distance/divisor) * coefficient )
}

weight.given.normal.distribution <- function(weight, std_dev, distance) {
  weight * ( (1/(sqrt(2 * pi * std_dev ^ 2))) * exp( -( (distance)^2 )/(2 * std_dev)^2 ) )
}

path.cost <- function(distance, combined_weight, std_dev, a, b) {
  if (distance == 0) {
    return(0)
  }
  m <- a * (distance - 1.5*std_dev) ^ 2 # It's squared to favour going through many stations over skipping from each one
  n <- 1 / (10^(b * combined_weight))
  o <- exp(distance - 1.5*std_dev)
  return(m + n * (1 + 1/o))
}

magnitude <- function(x) sqrt(sum(x^2))


# Load data into dataframe
POIs <- read.csv("transit_planner.csv", header = FALSE)
colnames(POIs) <- c("id", "google_places_id", "name", "latitude", "longitude", "weight_determining_type", "weight")
POIs$id <- NULL # Delete the column for google_places_id
```

```
POIs$google_places_id <- NULL # Delete the column for google_places_id


# Only selected weighted POIs and change the domain of their weight
weighted.POIs <- POIs[POIs$weight > 0,]
weighted.POIs$weight <- fit.domain(weighted.POIs$weight, 0, max(weighted.POIs$
    weight), 0, 10)


# Perform DBSCAN and save cluster to POIs dataframe
x <- as.matrix(weighted.POIs[, 2:3])
dbscan.results <- dbscan(x, eps = 0.0003, minPts = 65, weights = as.numeric(
    weighted.POIs$weight))
weighted.POIs$cluster <- as.factor(dbscan.results$cluster) # So ggplot does not
    interpolate colors


# Plot all POIs
plot(weighted.POIs$longitude, weighted.POIs$latitude, main = "POIs in Montreal",
    xlab = "Longitude", ylab = "Latitude", col = adjustcolor("black", alpha=0.15),
    cex = 0.1)


# Plot and save map of first round of clusters
ggplot(weighted.POIs, aes(x = longitude, y = latitude, color = cluster)) +
  geom_point(shape = 16, size = 0.1, alpha = 0.15) +
  theme(panel.background = element_blank(), legend.position = "none") +
  coord_fixed() + labs(x = "Longitude", y = "Latitude")

ggplot() +
  geom_point(data = weighted.POIs[weighted.POIs$cluster %in% c(0),], aes(x =
      longitude, y = latitude), shape = 16, size = 0.1, alpha = 0.05, color = "
      black") +
  geom_point(data = weighted.POIs[!(weighted.POIs$cluster %in% c(0)),], aes(x =
      longitude, y = latitude, color = cluster), shape = 16, size = 0.1, alpha = 1)
       +
  theme(panel.background = element_blank(), legend.position = "none", plot.title =
       element_text(hjust = 0.5)) +
  coord_fixed() + labs(x = "Longitude", y = "Latitude") + ggtitle("Clusters of
      POIs Found by DBSCAN Algorithm")


# Loop through all clusters to find their centroids and weights
cluster.centroids <- data.frame(matrix(ncol = 3, nrow = 0))
colnames(cluster.centroids) <- c("cluster", "centroid_latitude", "centroid_
    longitude")
num.clusters <- max(dbscan.results$cluster) # Find the total number of clusters
acceptable.cluster.values <- seq(1, num.clusters)
for (this_cluster in acceptable.cluster.values) {
  centroid_latitude <- weighted.mean( weighted.POIs[weighted.POIs$cluster %in%
      this_cluster,]$latitude, weighted.POIs[weighted.POIs$cluster %in% this_
      cluster,]$weight )
  centroid_longitude <- weighted.mean( weighted.POIs[weighted.POIs$cluster %in%
      this_cluster,]$longitude, weighted.POIs[weighted.POIs$cluster %in% this_
      cluster,]$weight )
  row_to_add <- data.frame(cluster = this_cluster, centroid_latitude = centroid_
      latitude, centroid_longitude = centroid_longitude)
  cluster.centroids <- rbind(cluster.centroids, row_to_add)
}
```

```
93  rm(row_to_add, centroid_latitude, centroid_longitude, this_cluster)
94
95
96  # For each centroid:
97  #  Find all the POIs in a given radius and their weights
98  search.radius <- 1500 # in metres; Just over 5 standard deviations
99  cluster.centroids["weight"] <- 0
100 cluster.centroids["line"] <- 0
101 for (this_cluster in acceptable.cluster.values) {
102   this_centroid <- cluster.centroids[cluster.centroids$cluster %in% this_cluster,]
103
104   lat_north <- distance.to.latitude(this_centroid$centroid_latitude, search.radius
          , "N")
105   lat_south <- distance.to.latitude(this_centroid$centroid_latitude, search.radius
          , "S")
106   lon_east <- distance.to.longitude(this_centroid$centroid_longitude, this_
          centroid$centroid_latitude, search.radius, "E")
107   lon_west <- distance.to.longitude(this_centroid$centroid_longitude, this_
          centroid$centroid_latitude, search.radius, "W")
108
109   relevant.POIs.weights <- subset(weighted.POIs, latitude <= lat_north & latitude
          >= lat_south & longitude >= lon_west & longitude <= lon_east, select = weight
          )
110   relevant.POIs.coords <- subset(weighted.POIs, latitude <= lat_north & latitude
          >= lat_south & longitude >= lon_west & longitude <= lon_east, select = c(
          latitude, longitude))
111   relevant_POIs_distance_from_centroid <- distHaversine(as.vector(this_centroid[,2
          :3]), relevant.POIs.coords)
112
113   sum_of_weights <- sum( weight.given.normal.distribution(relevant.POIs.weights, 2
          97.37, relevant_POIs_distance_from_centroid) )
114   cluster.centroids[cluster.centroids$cluster %in% this_cluster,]$weight <- sum_of
          _weights
115 }; rm(sum_of_weights, this_cluster, this_centroid, lat_north, lat_south, lon_east,
        lon_west, relevant_POIs_distance_from_centroid)
116
117 region.westernmost.lon <- min(cluster.centroids$centroid_longitude)
118 region.easternmost.lon <- max(cluster.centroids$centroid_longitude)
119
120 ggplot(cluster.centroids, aes(x = centroid_longitude, y = centroid_latitude, color
        = weight)) + geom_point(shape = 16, size = 0.8, alpha = 1) + theme(panel.
      background = element_blank(), plot.title = element_text(hjust = 0.5)) + coord_
      fixed() + labs(x = "Longitude", y = "Latitude") + ggtitle("Weighted␣Clusters␣of
      ␣POIs␣in␣Montreal␣\n␣as␣Found␣by␣DBSCAN␣Algorithm,␣Isolated␣from␣Unclustered␣
      POIs")
121
122
123 # Populate distance and weight matrices
124 sum.of.weights <- function(x, y) { x + y }
125 distance.matrix <- distm(cluster.centroids[,2:3], fun = distHaversine)
126 combined.weight.matrix <- as.matrix( dist(cluster.centroids[,4], sum.of.weights) )
127
128 cost.matrix <- matrix(Inf, nrow = num.clusters^2, ncol = 3)
129 i <- 1
130 for (x in acceptable.cluster.values) {
131   for (y in acceptable.cluster.values) {
132     cost.matrix[i,] <- c(x, y, path.cost(distance.matrix[x, y], combined.weight.
          matrix[x, y], 300, 10^(-5), 100))
133     i <- i + 1
```

```
  }
}; rm(i, x, y)


# Find the transit lines in the network
gmm <- Mclust( rev(cluster.centroids[,2:3]) )
cluster.centroids$gmm_cluster <- gmm$classification
pdf("axes.pdf", width=11, height=8.5)
plot(gmm, what = "classification")
dev.off()

ggplot(cluster.centroids, aes(x = centroid_longitude, y = centroid_latitude, color
    = as.factor(gmm_cluster))) + geom_point(shape = 16, size = 0.2, alpha = 1) +
   theme(panel.background = element_blank()) + coord_fixed() + labs(x = "Longitude
   ", y = "Latitude")

# Prepare GMM cluster data
gmm.clusters <- list(clusters = 1:gmm$G, cluster.means = gmm$parameters$mean,
                     covariance.matrices = gmm$parameters$variance$sigma,
                        eigenvalues = NULL, eigenvectors = NULL,
                     termini = NULL, type.determining.magnitude = NULL, average.
                        magnitude = 0, types = NULL)

# Find eigenvalues and eigenvectors of each GMM cluster
for (G in gmm.clusters$clusters) {
  eigenstuff <- eigen(gmm.clusters$covariance.matrices[,,G])
  gmm.clusters$eigenvalues[G] <- list(eigenstuff$values)
  gmm.clusters$eigenvectors[G] <- list(eigenstuff$vectors)

  # Calculate the vectors
  scaled.eigen.vec.m <- eigenstuff$vectors[,1] * sqrt(eigenstuff$values[1])
  scaled.eigen.vec.n <- eigenstuff$vectors[,2] * sqrt(eigenstuff$values[2])
  vector.a <- as.vector(gmm.clusters$cluster.means[,G]) + scaled.eigen.vec.m
  vector.b <- as.vector(gmm.clusters$cluster.means[,G]) + scaled.eigen.vec.n
  vector.c <- as.vector(gmm.clusters$cluster.means[,G]) - scaled.eigen.vec.m
  vector.d <- as.vector(gmm.clusters$cluster.means[,G]) - scaled.eigen.vec.n

  # Plot eigenvectors
  plot(c(gmm.clusters$cluster.means[,G][1], vector.a[1], vector.b[1], vector.c[1],
       vector.d[1]), c(gmm.clusters$cluster.means[,G][2], vector.a[2], vector.b[2],
       vector.c[2], vector.d[2]))
  arrows(gmm.clusters$cluster.means[,G][1], gmm.clusters$cluster.means[,G][2],
      vector.a[1], vector.a[2])
  arrows(gmm.clusters$cluster.means[,G][1], gmm.clusters$cluster.means[,G][2],
      vector.b[1], vector.b[2])
  arrows(gmm.clusters$cluster.means[,G][1], gmm.clusters$cluster.means[,G][2],
      vector.c[1], vector.c[2])
  arrows(gmm.clusters$cluster.means[,G][1], gmm.clusters$cluster.means[,G][2],
      vector.d[1], vector.d[2])

  # Find relevant slope
  mag.a <- magnitude(gmm.clusters$cluster.means[,G] - vector.a)
  mag.b <- magnitude(gmm.clusters$cluster.means[,G] - vector.b)
  if (mag.a >= mag.b) {
    slope <- vector.a[2] / vector.a[1]
    gmm.clusters$type.determining.magnitude[G] <- list(mag.a)
  } else {
    slope <- vector.b[2] / vector.b[1]
    gmm.clusters$type.determining.magnitude[G] <- list(mag.b)
```

```
182   }
183
184   # Find points where line leaves region of interest (point-slope form)
185   lat.westernmost.endpoint <- slope * (region.westernmost.lon - gmm.clusters$
          cluster.means[,G][1]) + gmm.clusters$cluster.means[,G][2]
186   lat.easternmost.endpoint <- slope * (region.easternmost.lon - gmm.clusters$
          cluster.means[,G][1]) + gmm.clusters$cluster.means[,G][2]
187
188   # Find index of closest stations to endpoints (the termini)
189   current.closest.westernmost.index <- -1
190   current.closest.westernmost.distance <- Inf
191   current.closest.easternmost.index <- -1
192   current.closest.easternmost.distance <- Inf
193
194   current.searchable.clusters <- cluster.centroids[cluster.centroids$gmm_cluster
          == G,]
195   for (this_cluster in current.searchable.clusters$cluster) {
196     this_centroid <- current.searchable.clusters[current.searchable.clusters$
            cluster %in% this_cluster,]
197     western.distance <- distHaversine(this_centroid[,3:2], c(region.westernmost.
            lon, lat.westernmost.endpoint))
198     eastern.distance <- distHaversine(this_centroid[,3:2], c(region.easternmost.
            lon, lat.easternmost.endpoint))
199     if ( western.distance < current.closest.westernmost.distance ) {
200       current.closest.westernmost.index <- this_centroid$cluster
201       current.closest.westernmost.distance <- western.distance
202     }
203     if ( eastern.distance < current.closest.easternmost.distance ) {
204       current.closest.easternmost.index <- this_centroid$cluster
205       current.closest.easternmost.distance <- eastern.distance
206     }
207   }
208
209   gmm.clusters$termini[G] <- list(c(current.closest.westernmost.index, current.
          closest.easternmost.index))
210
211 }; rm(G, scaled.eigen.vec.m, scaled.eigen.vec.n, vector.a, vector.b, vector.c,
        vector.d, mag.a, mag.b, slope,
212        lat.westernmost.endpoint, lat.easternmost.endpoint, current.closest.
               westernmost.index,
213        current.closest.westernmost.distance, current.closest.easternmost.index,
               current.closest.easternmost.distance,
214        current.searchable.clusters, this_centroid, this_cluster, western.distance,
               eastern.distance)
215
216 # Prepare igraph data
217 cost.matrix.filtered <- cost.matrix[cost.matrix[,3] > 0,]
218 entire.possible.network <- graph.data.frame(cost.matrix.filtered[, 1:2], directed
        = FALSE) %>%
219   set_vertex_attr("gmm_cluster", value = cluster.centroids$gmm_cluster) %>%
220   set_vertex_attr("cluster_id", value = cluster.centroids$cluster)
221 E(entire.possible.network)$weight <- cost.matrix.filtered[,3]
222
223 # Prepare line graphing data table
224 line.data.for.drawing <- data.frame(matrix(ncol = 5, nrow = 0))
225 colnames(line.data.for.drawing) <- c("line", "x", "y", "xend", "yend")
226
227
228 # Get average magnitude of each line and determine whether line is metro or train
```

```
gmm.clusters$average.magnitude <- mean(unlist(gmm.clusters$type.determining.magnitude))
for (G in gmm.clusters$clusters) {
  if (gmm.clusters$type.determining.magnitude[G] <= gmm.clusters$average.magnitude) {
    gmm.clusters$types[G] <- list("metro")
    metro <- TRUE
  } else {
    gmm.clusters$types[G] <- list("train")
    metro <- FALSE
  }

  if (metro) {
    western.terminus <- gmm.clusters$termini[[G]][1]
    eastern.terminus <- gmm.clusters$termini[[G]][2]

    this.metro.line <- induced.subgraph(entire.possible.network, which( V(entire.possible.network)$gmm_cluster %in% c(G)) )

    this.shortest.path <- shortest_paths(this.metro.line, from = match(as.character(western.terminus), V(this.metro.line)$cluster_id), to = match(as.character(eastern.terminus), V(this.metro.line)$cluster_id), mode = "in")


    # Minimum Spanning Tree
    this.metro.line.for.tree <- induced.subgraph(this.metro.line,
                                                 which(V(this.metro.line) %in% this.shortest.path$vpath[[1]]) )

    min.spanning.tree <- minimum.spanning.tree(this.metro.line.for.tree)
    V(min.spanning.tree)$label.cex <- 0.2
    plot.igraph(min.spanning.tree,
      layout = as.matrix(cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, 3:2]),
      vertex.label.size = "",
      vertex.size = .2,
      rescale = FALSE,
      xlim = c(min(cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, 3]), max(cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, 3])),
      ylim = c(min(cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, 2]), max(cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, 2])))
    title(as.character(G))


    # Update clusters to line if more than 5 stations
    if (length(this.shortest.path$vpath[[1]]) > 5) {
      cluster.centroids[cluster.centroids$cluster %in% this.shortest.path$vpath[[1]]$name, ]$line <- G

      # Compile data for minimum spanning tree so it can be drawn in final map
      for (i in 1:length(E(min.spanning.tree))) {
        x <- cluster.centroids[cluster.centroids$cluster == as.integer(ends(min.spanning.tree, E(min.spanning.tree))[i,1]), 3]
        xend <- cluster.centroids[cluster.centroids$cluster == as.integer(ends(min.spanning.tree, E(min.spanning.tree))[i,2]), 3]
        y <- cluster.centroids[cluster.centroids$cluster == as.integer(ends(min.
```

```
            spanning.tree, E(min.spanning.tree))[i,1]), 2]
273         yend <- cluster.centroids[cluster.centroids$cluster == as.integer(ends(min
                .spanning.tree, E(min.spanning.tree))[i,2]), 2]
274         row.to.add <- data.frame(G, x, y, xend, yend)
275         colnames(row.to.add) <- c("line", "x", "y", "xend", "yend")
276         line.data.for.drawing <- rbind(line.data.for.drawing, row.to.add)
277       }; rm(x, xend, y, yend, row.to.add, i)
278     }
279
280   } else { # Otherwise, the train line must connect to either a metro station that
          reaches the central station or directly to the central station
281
282   }
283 }; rm(G, metro, this.metro.line, this.shortest.path, western.terminus, eastern.
      terminus)
284
285
286 # Plot all the lines on one map
287 geographical.map.data <- cluster.centroids[cluster.centroids$line > 0,]
288 geographical.map.average.longitude <- mean(geographical.map.data$centroid_
      longitude)
289 geographical.map.average.latitude <- mean(geographical.map.data$centroid_latitude)
290 geographical.map.center <- c(lon = geographical.map.average.longitude, lat =
      geographical.map.average.latitude)
291 geographical.map.constructor <- get_map(location = geographical.map.center, zoom =
       12, source="google", maptype = "roadmap", color="bw")
292 # Add existing stations (from STM)
293 all.existing.stations <- read.csv("stops.csv")
294 relevant.existing.stations <- all.existing.stations[all.existing.stations$location
      _type == 1,]
295 # Plot
296 overlay.map <- ggmap(geographical.map.constructor) +
297   geom_point(data = relevant.existing.stations,
298              aes(x = stop_lon, y = stop_lat), color = "black", shape = 17) +
299   geom_point(data = geographical.map.data,
300              aes(x = centroid_longitude, y = centroid_latitude, color = as.factor(
                  line))) +
301   geom_segment(data = line.data.for.drawing, aes(x = x, y = y, xend = xend, yend =
       yend, color = as.factor(line))) +
302   theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.5)
      ) +
303   labs(x = "Longitude", y = "Latitude") +
304   ggtitle("Final␣Suggested␣Metro␣System")
305 overlay.map
```