

# Optimized Design For GTA Metro System

Winter 2020

## **Team #19**

Ruirong Chen (20659065)

Lizheng He (20649609)

Zelu Li (20617017)

# 1. Introduction

Metro transit, also widely known as subway, is a high-capacity public transport in urban areas. Its advantages include “reduce reliance on cars, mitigate sprawl, and provide residents with access to affordable transportation” [1]. Recent research shows that the subway could also “have an economically insignificant effect on urban population growth” and help to expand the city [2]. The metro system is important to a city, its design should be carefully chosen and decided.

The main topic of the project is to redesign the metro system in Toronto so that this new system is able to satisfy as many passengers as possible with a lowest building cost. Prior to our work, we assume that the only transportation method in Toronto is the subway.

Our analysis has three major steps: In the first step, we will choose and decide the geometry location for the subway stations. Secondly, we will find those unsafe factors for running our subway system by doing research, in order to add constraints on the following station connection. Finally, among all the stations and constraints we get, we will compute and choose a subway system that connects each station with minimum building cost by building a complex net flow optimizer algorithm.

During the process, we will first build the subway system in Downtown Toronto, in order that we can compare and evaluate our system with the real Toronto subway transit system. Once the comparison result shows our system is feasible, we are able to apply our designing method outside Downtown Toronto and redesign a new subway transit system in GTA. We have divided the GTA area into four blocks, named block\_1, block\_2, block\_3 and block\_4, in order to enhance the data collection processing.

## 2. Data Collection

We need multiple aspects of data, as shown in the following table (table\_2.1):

Population	Construction Cost	Passenger Information
Permanent residents	Construction cost varies in different area	Distance to walk to the station

First, we will use the current subway in the Toronto Downtown area to estimate the population coverage for a single station. We used the 2016 Census profile from Statistics Canada [3] and the API and map from census mapper [4] to find the population. For permanent residents, we summed the population of districts near each subway station from the "Subway and Streeter Map" from ttc [5]. For tourist data, we looked at several attractions in Toronto Downtown and added capacity per day to the nearby subway station. (Appendix A)

After that, we get that a subway station covers about 10842 people. Then we will divide GTA into areas which have about 50000 population in each and locate 5 stations in each area. The map we used is based on the census mapper population density map.

Based on the Walking Distance Research by the Planning Commission TOD Committee, residents are willing to walk slightly longer distances to get to transit, between a quarter and a half-mile [8]. We will take a half-mile as the acceptable walking distance by the passengers in our new designs.

Once we have divided GTA into small areas, we are able to allocate stations in each area. Since each area is going to have around 5 stations, for some small areas (areas in dt), we will allocate 3 stations closely. And for some areas which have a lower population density and larger area, the allocation of stations would be based on the largest distance passengers are willing to walk. The stations would be located appropriately in order to maximize the number of potential passengers in that area.

In addition, for the construction cost on building subway lines in GTA. We decided to use three different costs per kilometer depending on the allocation of the line. In real life, the construction

fee is different for different areas, like the cost in Downtown(dt) Toronto is higher than other places. From the data[6][7], we choose C\$ 290 million/km for any connection of metro stations in dt Toronto, C\$ 270 million/km for connections in Toronto but outside dt and C\$ 250/km for connections in other areas of GTA.

Last step before running the optimization model is to calculate the distance between each station. In order to increase the accuracy, we print the GTA population density map with area division and stations location on it. On the printed map, we build a Cartesian coordinate system for each block, and measure the coordinate for each station in that block. Coordinate data sets are included in 'coordinate\_1.csv', 'coordinate\_2.csv', 'coordinate\_3.csv' and 'coordinate\_4.csv'. To measure the distance, we programmed a simple algorithm based on the formula: (Appendix B)

**For two points  $(x_1, y_1)$  and  $(x_2, y_2)$ , the distance  $d$  between these two points is:**

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

### 3. Model

As mentioned above, we divided the GTA in to four pieces, named by block\_1, block\_2, block\_3 and block\_4. What are we going to do is to redesign the subway line for each part.

Define graph  $G = (V, E)$  where  $V$  are the set of all the subway stations.

w.l.o.g. let subway stations are represented as vertices in graph  $G$ .

For each block:

The output of the mathematical model will be a new graph  $G' = (V, E')$ , where  $E' \subseteq E$ .

Such that:  $G'$  is a connected graph with minimizing the total cost for connecting  $V$  by  $E'$  and the distance between any two nodes  $u$  and  $v$  in  $G'$  should less than  $D$ . In addition, in order to facilitate the running of subway and be convinence for passengers, we also have restriction on the number of leaves that  $G'$  includes.

Let  $d_e$  be the distance of edge  $e \in E$ , and let  $D$  be the longest distance that people willing to spend on subway.

Let  $c_{u,v}$  be the cost of building the underground railway between nodes  $u$  and  $v$ .

Define  $x_{u,v}$  be a binary variable, with  $x_{u,v} = 1$  for edge  $uv \in E'$ , and 0 otherwise.

To add restriction on number of leaves for subway line in each part, we will choose some beginning(ending) points which are leaves in  $G'$  and restrict other vertices not to be leaves.

In addition, to ensure  $G'$  is a connected undirected graph, we are going to find a spanning tree in  $G$ . Therefor, we have two more constraints:

Let  $n$  be the number of nodes in  $G$ ,  $n = |V|$ .

Fisrt constraint represents  $n - 1$  edges in  $G'$ :  $\sum_{u,v \in E} x_{u,v} = n - 1$

Second constraint represents no cycles in  $G'$ : Any subset of  $k$  vertices must have at most  $k - 1$  edges contained in that subset.

$$\text{Min } \sum_{u,v \in V} x_{u,v} * c_{u,v}$$

s.t.

$$\sum_{u \in V} x_{u,v} \geq 2 \text{ for every not leaf vertex } v \in V$$

$$\sum_{u \in V} x_{u,v} = 1 \text{ for every leaf vertex } v \in V$$

$$\sum_{u,v \in E} x_{u,v} = n - 1$$

$$\sum_{u,v \in E: u \in S, v \in S} x_{u,v} \leq |S| - 1 \text{ for every subset } S \subseteq V$$

The programming model is in the file ‘Team19\_model.py’ also in appendix C.

The beginning/ending points are: 3, 8, 91, 97, 12, 35, 50, 74 for Block\_1, 1, 20, 39, 41 for Block\_2, 2, 28, 58, 61 for Block\_3 and 10, 25, 73, 61, 1, 9, 67 for Block\_4.

However, there is a time complexity issue in our model. Since the formulation for our model includes finding all the subsets of vertices in the graph, and it has a complexity equal to  $n * 2^n$ . In our data set, the smallest block still has 49 vertices, which means  $n = 49$  in the smallest case. This complexity is too large to run on our laptop (with CPU).


In this case, we decided to simplify our model when programming it, we did not add the last constraint in our programming model. As we can see from figure\_3.1, the outcome from the simplified model does have some bias, the graph in each block is not connected! These biases can be easily deleted by the last constraint. In order to prove that the last constraint is correct, we also build another programming model based on the same data set. It is going to be submitted in the file ‘Team19\_model\_MST’. This model would output a minimum spanning tree in each block. The output has been shown in figure\_4.1. The formulation for this model is in below, which contains the last two constraints in our project model:

$$\text{Min } \sum_{u,v \in V} x_{u,v} * c_{u,v}$$

s.t.

$$\sum_{u,v \in E} x_{u,v} = n - 1$$

$$\sum_{u,v \in E: u \in S, v \in S} x_{u,v} \leq |S| - 1 \text{ for every subset } S \subseteq V$$



The output are saved in 'subway\_line\_1.csv', 'subway\_line\_2.csv', 'subway\_line\_3.csv' and 'subway\_line\_4.csv' for our project model, and saved in 'minimum\_spanning\_tree\_1.csv', 'minimum\_spanning\_tree\_2.csv', 'minimum\_spanning\_tree\_3.csv' and 'minimum\_spanning\_tree\_4.csv' for the minimum spanning tree model.


## 4. Findings

After determining the stops and optimizing the subway lines by connecting stops, we are able to redesign the metro system in Toronto which maximizes the subway accommodation and minimizes the building cost at the same time. It also satisfies passenger's needs such as the most comfortable walking distance and time. In the three major steps, we are able to locate the subway stations geometrically and connect all the stations in which all the unsafe factors are avoided by adding the constraints on the station connection. We also make sure that the built station line is at minimum building cost by building a complex net flow optimizer algorithm.

We used Python to implement our code and the output for the algorithm is a connected undirected simple graph with vertices that represents metro stations and edges. The final output represents our redesigned metro lines. This new metro lines design effectively solves the following existing potential pitfalls of the existing subway lines:

1. Walking distance is too far for residences who live off the subway lines. A lot of them, especially those who live in places like Scarborough and Mississauga where the population density is not as high, have to suffer from walking too much especially in extreme weather. Our redesigned station lines have improved this weakness by locating the station stops based on the population distribution and passengers' favours. It will provide passengers the best riding experiences and thus attracts more people to choose to take the subway as their optimal traveling choice.
2. The current subway design may not be space-efficient. Even though it seems like the current subway line has covered most areas especially in downtown Toronto, there are still gaps between two stops which should be covered as new stops. Our new subway lines design improves this weakness by building a complex net flow optimizer algorithm which puts this weakness into consideration and makes sure that our connection between stops can accommodate more people with minimal cost and less space occupied.
3. The current subway design may not be cost-efficient. In our new design, we split different costs that may occur in detail and try to minimize the total cost. We find the average expense to construct underground subway per mile and estimate the time and workload to construct a one-mile underground subway. Furthermore, we also calculate the resources costs like the average salary for each worker. We also pay attention to the construction cost for different geologies.





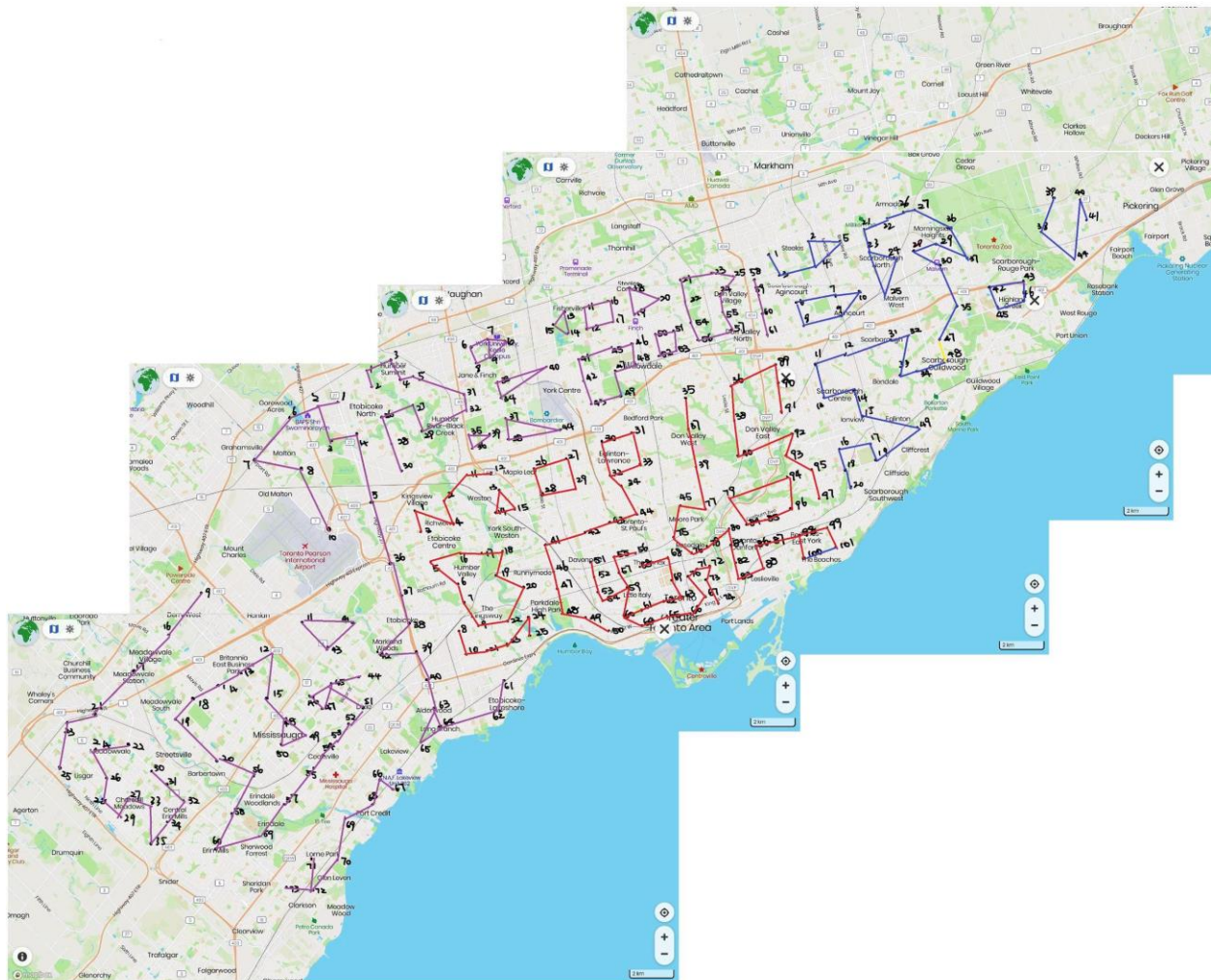
Therefore, our new proposed subway line designs effectively improve the potential weakness of the existing subway design and provide the passengers with the best riding experience with minimal cost. It will then attract more people to choose to take subways as they travel around which reduces reliance on private cars and provides residents with access to more affordable transportation. It is also environmentally friendly and helps with urban population growth and city expansion. Our proposal will be sustainable but may be subject to change in the future based on the population distribution, building cost, and passengers' favours.

## Reference

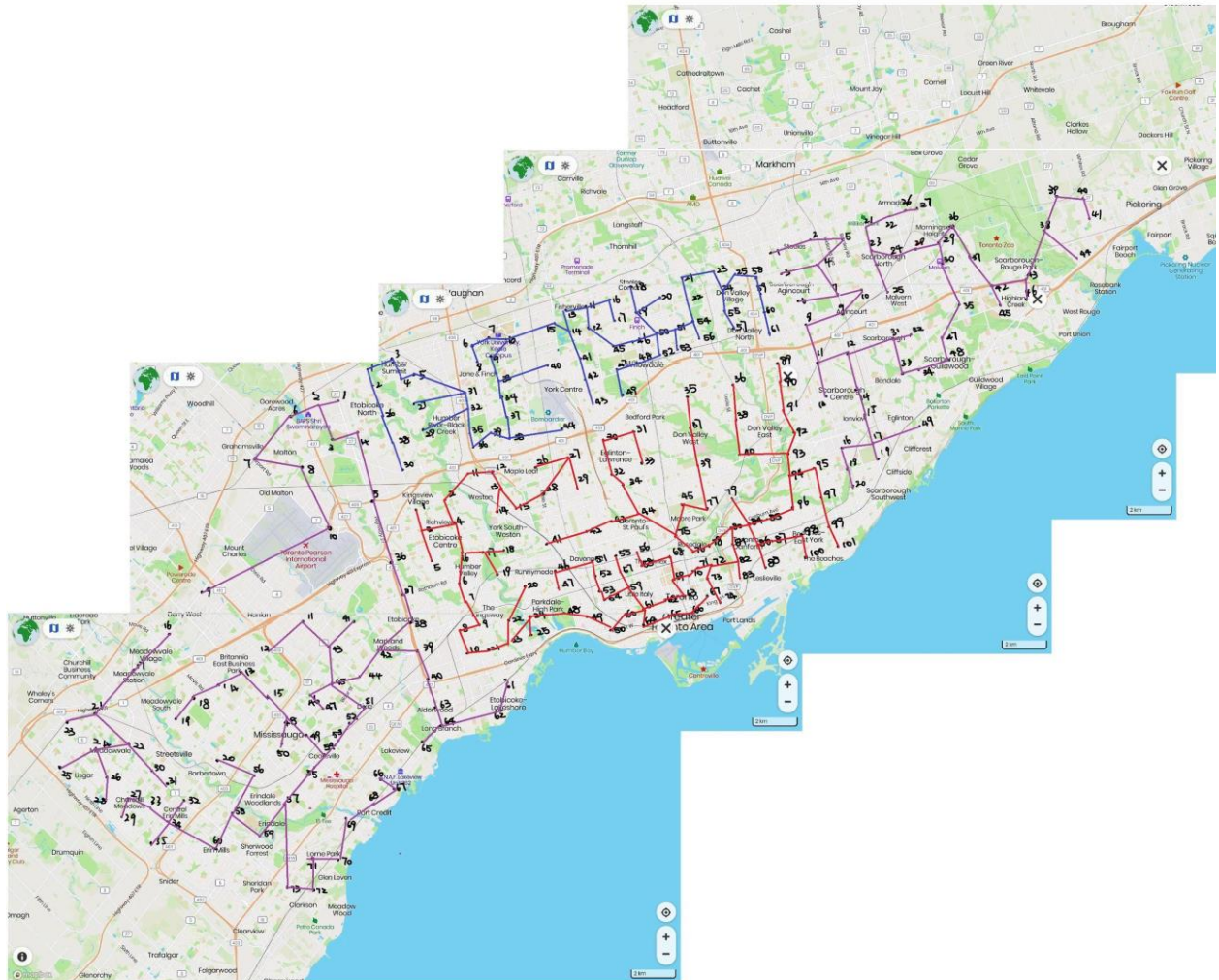
- [1]. Richard Florida, June 2, 2016, The Relationship Between Subways and Urban Growth, Citylab, Retrieved from <https://www.citylab.com/transportation/2016/06/the-relationship-between-subways-andurban-growth/485006/>
- [2]. Marco Gonzalez-Navarro, Matthew A. Turner, May 30 2016, Subways and Urban Growth: Evidence from Earth, Journal of Urban Economics, vol 108, pp 85-106, DOI: 10.3386/w24996
- [3]. Statistics Canada. (2020, March 31). Census Profile, 2016 Census. Retrieved from <https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/prof/index.cfm?Lang=E&HPA=1>
- [4]. (n.d.). Retrieved from <https://censusmapper.ca/api>
- [5]. Maps. (n.d.). Subway and Streetcar Map. Retrieved from. [https://www.ttc.ca/Routes/General\\_Information/Maps/index.jsp](https://www.ttc.ca/Routes/General_Information/Maps/index.jsp)
- [6]. Average construction cost in different areas. <http://lrt.daxack.ca/LRTvsHRT/CostCompare.html>
- [7]. Toronto Transit Commission Report. Scarborough Subway Options. [https://www.ttc.ca/About\\_the\\_TTC/Commission\\_reports\\_and\\_information/Commission\\_meetings/2013/September\\_25/Supplementary\\_Reports/Scarborough\\_Subway\\_O.pdf](https://www.ttc.ca/About_the_TTC/Commission_reports_and_information/Commission_meetings/2013/September_25/Supplementary_Reports/Scarborough_Subway_O.pdf)
- [8] Planning Commission TOD Committee - Walking Distance Research. <http://www.reconnectingamerica.org/assets/Uploads/20060908walkingdistanceabstracts.pdf>

# Appendix

Figure 4.1:



Figure\_4.2:



## A.

```
library(cancensus)

## Census data is currently stored temporarily.

##

## In order to speed up performance, reduce API quota usage, and reduce unnecessary network calls, please set up a persistent cache directory by
setting options(cancensus.cache_path = '<path to cancensus cache directory>')

##

## You may add this option, together with your API key, to your .Rprofile.

options(cancensus.api_key='CensusMapper_5d84f33015ae762d06a71b7d9a31862a')

options(cancensus.cache_path = '/Users/jerryzhou/workspace/school')

census_data_DA <- get_census(dataset='CA16',
regions=list(CSD=c("3520005","3519036","3518001","3521005","3524015","3524009","3524001","3521010")), vectors=c(), labels="detailed",
geo_format=NA, level='DA')

## Reading vectors data from local cache.

census_data_DA

## # A tibble: 6,242 x 11

##      GeoUID Type `Region Name` `Area (sq km)` Population Dwellings
##      <chr> <fct> <fct>          <dbl>          <dbl>    <dbl>
## 1 35240... DA   Oakville          0.115          492      148
## 2 35211... DA   Brampton          0.0693         1987      702
## 3 35200... DA   Toronto           0.0685          532      169
## 4 35204... DA   Toronto           0.0837         2622     1793
## 5 35200... DA   Toronto           0.0294          300      150
## 6 35240... DA   Milton            0.104           729      217
## 7 35204... DA   Toronto           0.0518          504      206
## 8 35200... DA   Toronto           0.0492          473      188
## 9 35204... DA   Toronto           0.102           551      214
## 10 35201... DA   Toronto           0.0772          438      170
## # ... with 6,232 more rows, and 5 more variables: Households <dbl>,
## # CSD_UID <chr>, CD_UID <chr>, CT_UID <chr>, CMA_UID <chr>

Union = c(35204896,35204650,35204826,35204897,35204650,35204900)
```

```

St_Andrew = c(35204845, 35204842, 35204843, 35200855, 35200859)

King = c(35204022, 35200822, 35204897, 35204650, 35200833, 35200826)

Osgoode = c(35204616, 35204615, 35204018, 35200900)

Queen = c(35204246, 35204245, 35204228, 35203176)

Dundas = c(35204608, 35204616, 35204615, 35204609, 35204614, 35204607)

St_Patric = c(35200900, 35200899, 35200898, 35200903, 35204018, 35200861, 35200863, 35200862)

Queens_Park = c(35204901, 35204157, 35200896, 35200903, 35200902, 35200901, 35204617)

College = c(35204606, 35204902, 35204592, 35204590, 35204589, 35204588, 35204586, 35204605, 35204591)

Museum = c(35204157, 35201035, 35201036, 35201039, 35202820, 35204603, 35204602)

Wellesy = c(35204623, 35204624, 35204583, 35204591, 35204581, 35204582, 35204584, 35204585, 35204004, 35200782)

Bloor_Yonge = c(35202820, 35204600, 35204618, 35204882, 35204620, 35204578, 35202821, 35204598)

St_George = c(35201034, 35201035, 35201036, 35201038, 35201039)

Spadina = c(35201031, 35201029, 35201059, 35201058, 35201033, 35201034, 35201035)

Rosedale = c(35202814, 35202812, 35202813, 35202879, 35202878)

Summerhill = c(35202811, 35202787, 35202788, 35204510, 35204509, 35204507)

St_Clair = c(35202790, 35204735, 35204734, 35204504, 35204503, 35204506, 35204505)

Davisville = c(35204468, 35204469, 35204465, 35202910, 35202911, 35202906, 35202928, 35203986)

Lawrence = c(35202347, 35202346, 35202344, 35202345, 35202371, 35202372, 35202373, 35202348)

York_Mills = c(35202363, 35204040, 35202398, 35202620, 35202608)

ROM_near_Museum = 13374

Ripley_Aquarium_near_Union = 32877

CN_Tower_near_Union = 4931

Art_Gallery_near_St_Andrew = 5000

caculate_population <- function(station_list) {

  area = 0

  for (each_DA in station_list){

    area = area + census_data_DA[census_data_DA$GeoUID == each_DA, ]$Population

  }

  return(area)

}

```

```

total_population = caculate_population(Union) + caculate_population(St_Andrew) + caculate_population(King) +
caculate_population(Osgoode) +

caculate_population(Queen) + caculate_population(Dundas) + caculate_population(St_Patric) + caculate_population(Queens_Park) +
caculate_population(College) + caculate_population(Museum) + caculate_population(Wellesy) + caculate_population(Bloor_Yonge) +
caculate_population(St_George) + caculate_population(Spadina) + caculate_population(Rosedale) + caculate_population(Summerhill) +
caculate_population(St_Clair) + caculate_population(Davisville) + caculate_population(Lawrence) + caculate_population(York_Mills) +
ROM_near_Museum + Ripley_Aquarium_near_Union + CN_Tower_near_Union + Art_Gallery_near_St_Andrew

avg_population = total_population / 19

avg_population

## [1] 10842

```

**R**

```
In [1]: import pandas as pd
        from math import *
```

```
In [2]: x = []
        y = []

        nodes = pd.read_csv('coordinate_3.csv')
        for i in nodes.index:
            x.append(nodes['x'][i])
            y.append(nodes['y'][i])
```

```
In [3]: u = []
        v = []
        dis = []
```

```
In [4]: i = 1
        while i <= 61:
            j = i+1
            while j <= 61:
                v.append(j)
                u.append(i)
                d = sqrt((x[i-1] - x[j-1])**2 + (y[i-1] - y[j-1])**2)
                d1 = d/5
                dis.append(d1)
                j += 1
            i += 1
```

```
In [5]: cost = []

        for i in range(len(u)):
            c = dis[i]*250
            cost.append(c)
```

```
In [6]: data = pd.DataFrame({'u': u, 'v': v, 'distance': dis, 'cost': cost})
        data.to_csv("Data_3.csv", index=False, sep=',')
```



## C.

```
In [1]: import pandas as pd
        from gurobipy import *
        import networkx as nx
```

```
In [19]: u = []
        v = []
        c = []

        g = pd.read_csv("Data_3.csv")
        for i in g.index:
            u.append(g['u'][i])
            v.append(g['v'][i])
            c.append(g['cost'][i])
```

```
In [20]: nodes = []
        for i in u:
            if i not in nodes:
                nodes.append(i)
        for i in v:
            if i not in nodes:
                nodes.append(i)
```

```
In [21]: multidict_input = {}

        for i in g.index:
            multidict_input[u[i],v[i]] = c[i]

        edges, cost = multidict(multidict_input)
```

```
In [5]: T = nx.Graph()
        T.add_edges_from(edges)
```

```
In [6]: leaf = [2,28,58,61]
        not_leaf = []

        for i in range(len(nodes)):
            if nodes[i] not in leaf:
                not_leaf.append(nodes[i])
```

```

In [7]: m = Model('Subway')

        Using license file C:\Users\HE\gurobi.lic
        Academic license - for non-commercial use only

In [8]: x = {}
        for u,v in edges:
            x[(u,v)] = m.addVar(vtype = GRB.BINARY, name = f'x_{u},{v}')

In [9]: m.setObjective(quicksum(cost[u,v]*x[u,v] for u,v in edges), GRB.MINIMIZE)

In [10]: for v in not_leaf:
            v_edges = []
            for i in range(len(edges)):
                if v in edges[i]:
                    v_edges.append(edges[i])
            m.addConstr(quicksum(x[a,b] for a,b in v_edges) >= 2)

            for u in leaf:
                u_edges = []
                for i in range(len(edges)):
                    if u in edges[i]:
                        u_edges.append(edges[i])
            m.addConstr(quicksum(x[a,b] for a,b in u_edges) == 1)

In [11]: m.addConstr(quicksum(x[u,v] for u,v in edges) == len(nodes)-1)

Out[11]: <gurobi.Constr *Awaiting Model Update*>

In [12]: # n = len(nodes)
        # set_nodes = [[]]

        # for u in nodes:
        #     set_nodes += [curr + [u] for curr in set_nodes]

        # for i in set_nodes:
        #     sub_edge = []
        #     for u,v in edges:
        #         if u in i and v in i:
        #             sub_edge.append((u,v))
        #     m.addConstr(quicksum(x[u,v] for u,v in sub_edge) <= len(i) - 1)

```

```
In [13]: m.optimize()
```

```
Gurobi Optimizer version 9.0.1 build v9.0.1rc0 (win64)
Optimize a model with 62 rows, 1830 columns and 5490 nonzeros
Model fingerprint: 0xdfaa18e3
Variable types: 0 continuous, 1830 integer (1830 binary)
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+02, 4e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 6e+01]
Presolve time: 0.01s
Presolved: 62 rows, 1830 columns, 5372 nonzeros
Variable types: 0 continuous, 1830 integer (1830 binary)

Root relaxation: objective 1.412059e+04, 68 iterations, 0.00 seconds

   Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time

  0  0 14120.5864  0  6   - 14120.5864  - - 0s
H  0  0           14198.664009 14120.5864 0.55% - 0s
H  0  0           14131.531505 14120.5864 0.08% - 0s
  0  0 14123.6640  0  4 14131.5315 14123.6640 0.06% - 0s

Cutting planes:
  Clique: 1
  Zero half: 1

Explored 1 nodes (71 simplex iterations) in 0.07 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 14131.5 14198.7

Optimal solution found (tolerance 1.00e-04)
Best objective 1.413153150511e+04, best bound 1.413153150511e+04, gap 0.00000%
```

```
In [14]: m.write('ans3.lp')
```

```
In [16]: solution = m.getAttr('X')
outcome = []

for i in range(len(edges)):
    if solution[i] != 0:
        outcome.append(edges[i])
```

```
In [17]: ans = pd.DataFrame({'edge': outcome})
ans.to_csv("subway_line_3.csv", index = False, sep=',')
```