Complex Network Project : The TCL Network

December 2020

Introduction

For this project we chose to work on the TCL network. It is one type of network that we use everyday and that we never consider with this approach. This is the occasion to put into practice theorical concepts on a real network. The advantages of a such network are that we can try to interpret the results we find in term of public-transport company network to see if this makes sense or not. The idea of this project is to study the network with tools that we learned in class to see if this allows us to find results about the organisation of the "Grand Lyon Metropole" that match reality. The problematic we choose for this project is then:

Can we get information about the "Grand Lyon metropole" organisation using network analysis on the TCL network?

DATA

The first step of the project was to find data about the TCL network. Then, we had to work on this data to create our own network with networkx on Python to study it.

Origin of the DATA

The first thing we have to do in order to study the transport network was to get the network in a graph form. We looked on the internet and we were not able to find such a network. Instead, we found several databases in dictionnaries formats (.json) on the website data.gouv.fr. Thus, we decided to extract a graph from these data. The two databases we found were :

- The first database we found contained all the stations of the network with some information on it. We picked in this database the names of the stations, the set of conveying lines (buses, tramways, metrolines...) which pass through the stations and the locations of the stations.
- The second database we used contained information on each line. Unfortunately, the list of stations served by each line was not part of the database. Nevertheless, for each line, there was the location of its route.

Creation of the first network

Building the edges from database

From these two databases, we built the appropriate graph. The first step was to rally for each line the list of all the stations it served. To do so, we iterated on all the stations and each time one specific line appeared, we put the station in the list attributed to this line. At the end of this operation, we had a list of list containing all lines and for all lines the list of stations served. Yet, the list of stations for each line had no reason to be sorted the right way.

The second step we performed to build the graph was to sort the stations served by each line. At this moment we used the second database. For each line, we extract the associated route. The route was discretized in several locations (1000 in order of magnitude). Thus, we iterated on these locations. For each location, we looked for the nearest station (thanks to the first database) and we sorted the list of station this way. This step leads to an issue in some situations. In windings roads, the algorithm found that a station may be closer to the station two steps further than the next stop.

Finally, we iterated on each line and we created an edge from station to station between two nearest stations on the route of each line, so as to obtain the desired graph.

To illustrate this process, here is the form the node Debourg would get if we just look at the metro and tram lines. In this case, all the lines don’t link to the same stations, however if a bus also goes from ENS de Lyon to Debourg, their will be only one edge between these 2 stations.

Attributes on edges

While building the graph, we saved for each edge the associated line. Thus, for each edge we attributed several numbers according to the associated mean of transport (Metro, Tramway, Bus...). We arbitrarily attributed a capacity (mean number of person in the transport), a time scale (period of time between two passages) and a weight (capacity divided by time scale). Those numbers will be useful to do simulations.

Here is the detail of the attributes on edges in our python code :

- 'contenance' : capacity, mean number of person in the transport
- 'period' : average period of time between two passages
- 'weight' : 'contenance' divided by 'period'

Attributes on nodes

After building the graph, we wanted to add some information on the different nodes. Firstly, we added the location of each node (thanks to the first dataset). Secondly, we wanted to add the city (or the district) of each station. In order to fulfill this purpose, we found a third database (still from the same source) in which there were for each city (or district) the location of its border. We extracted those locations and we calculated a barycenter of those positions for each city. Then, we associated a city (or district) to each station by looking for the closest barycenter to the location of the station. This method is not perfectly precise but we thought it would be enough for our future study. Finally, we added an approximation of the number of people living close to each station. Indeed, we found on wikipedia the population of each city (or district), then we divided this population by the number of station in this city (or district) and we attributed this ratio to each station in the corresponding city (or district).

Here is the detail of the attributes on nodes in our python code :

- 'location' : 2D coordinates of the station (Geojson coordinates)
- 'commune' : city or district of the station
- 'densite_arret' : density of people that we affected to the station

Creation of the second network

We built a second network to estimate the time to travel from one station to another. Thus, we created a directed "multilayer" graph. We started from the list of lines with sorted stations. However, instead of having a station in which the connection between lines is instantaneous, here in each station we created sub-stations for each line. With this modelisation, it coasts time to change line, which is closer to reality. We arbitrarily chose the time-scale associated to each kind of change (bus to metro, metro to tramway...) and set edges attributes the same way as in the first network. Here is an exemple for the Debourg station with only the tram and metro lines.

Modelisation of the debourg station in the second network. We represented in color the lines and in grey the connections between lines: it represents the time taken to walk from a stop to another plus the average waiting time.

First analysis: How is builded the TCL network?

For the first part of the analysis we worked on the first network. With this first modelisation we were able to better characterise the network.

Preparation

Librairies importation

In [1]:
```
# Import librairies
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from cdlib import algorithms, viz, NodeClustering
from collections import Counter


%matplotlib inline
```

DATA importation

In [2]:
```
G = nx.read_graphml('tcl_v3_.graphml')
```

Change location from str to tuple

In [3]:
```
loca_dic = dict(nx.get_node_attributes(G,'location'))
locations = list(loca_dic.values())

#create tuples
for i in range(len(locations)):
    locations[i] = tuple(map(float, locations[i].split(',')))

#create dictionnary
i=0
for key in loca_dic:
    loca_dic[key] = locations[i]
    i+=1

#add new attribute
nx.set_node_attributes(G, loca_dic, "location")

#example of one node
print('Jean Macé Stop')
G.nodes()['Jean MacÃ©']
```

```
Jean Macé Stop
```

Out[3]:
```
{'location': (4.842293258518623, 45.74612138997321),
 'commune': 'LYON 7EME',
 'densite_arret': 2628.3870967741937}
```

Contraction of the Bellecour and Part-dieu stops

As we want to make measurements on the network, we need to contract all the Bellecour and Part-dieu stops in one stop to better represent the "hub" behavior of this 2 stops. However, this approximation seems to be less accurate for Part-Dieu as it is more of a neighborhood and some of the Part-Dieu stations are quite far. We let here the code if you want to try our program with this contraction but we won't use it in the following study.

In [4]:
```
G_contracte = G.copy()
```

Contraction for Part-Dieu stop:

In [5]:
```
#First we get all the node with 'part-dieu' in their name
nodes_l = list(G.nodes)
part_dieu =[]
for name in nodes_l :
    if 'Part-Dieu' in name :
        part_dieu.append(name)

part_dieu
```

Out[5]:
```
['Part-Dieu Jules Favre',
 'Gare Part-Dieu Vivier Merle',
 'Part-Dieu Renaudel',
 'Part-Dieu Auditorium',
 'Part-Dieu Servient',
 'Gare Part-Dieu Villette']
```

RUN IT ONLY IF YOU WANT TO CONTRACTE ALL THE PART-DIEU STATIONS IN ONE

In [148]:
```
#then we contract them together
G_contracte = nx.contracted_nodes(G_contracte, part_dieu[0], part_d
for i in range(2,len(part_dieu)):
    G_contracte = nx.contracted_nodes(G_contracte, part_dieu[0], pa
```

Contraction for Bellecour stops:

In [6]:
```
nodes_l = list(G.nodes)
bellc =[]
for name in nodes_l :
    if 'Bellecour' in name or 'bellecour' in name:
        bellc.append(name)

bellc
```

Out[6]:
```
['Bellecour Charite',
 'Bellecour Chambonnet',
 'Bellecour Le Viste',
 'Bellecour A. Poncet',
 'Bellecour St Exupery',
 'Bellecour']
```

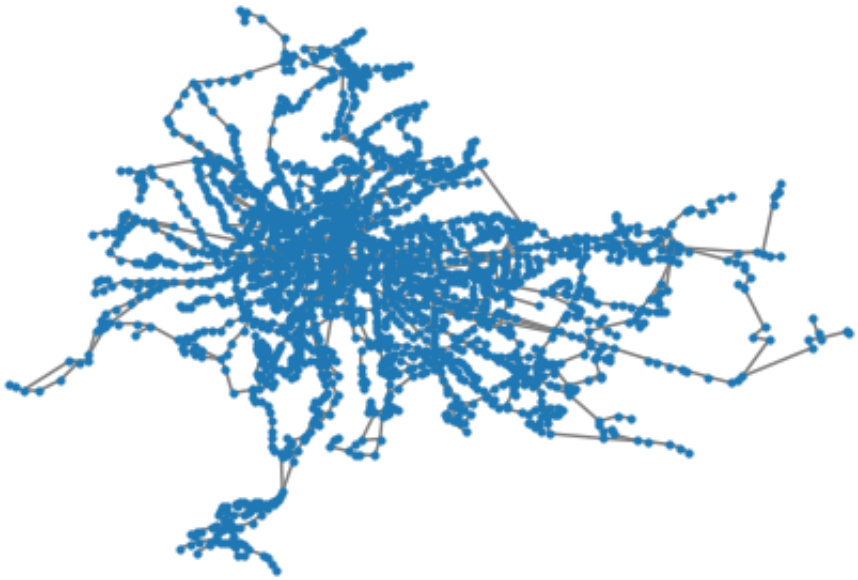
In [7]:
```
#Contraction of Bellecour
liste = bellc
G_contracte = nx.contracted_nodes(G_contracte, liste[0], liste[1], 
for i in range(2,len(liste)):
    G_contracte = nx.contracted_nodes(G_contracte, liste[0], liste[
```

First vizualisation

Now that our network is ready, we can vizualise it to make sure it is as expected and to get a first impression.

A first basic representation:

In [8]:
```
%matplotlib inline
#%matplotlib qt
nx.draw(G_contracte, pos = nx.get_node_attributes(G_contracte,'loca
```



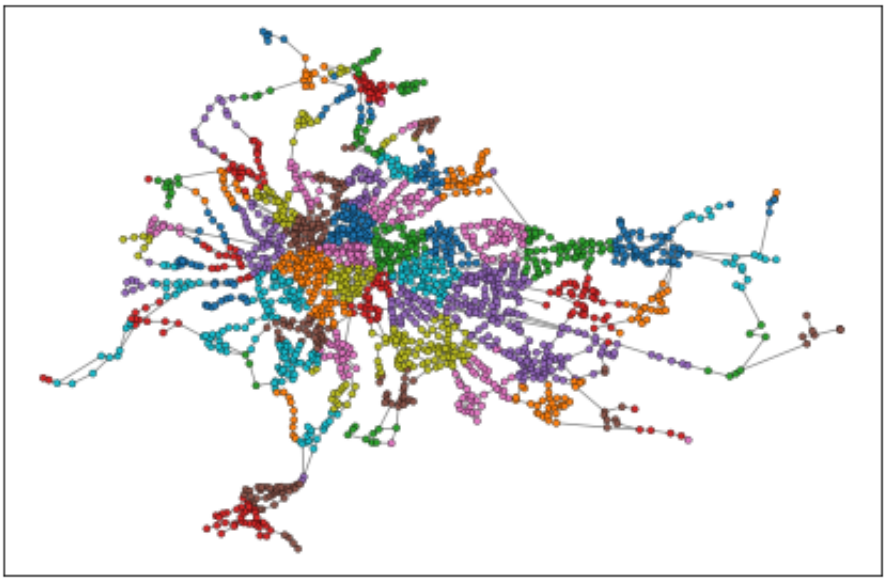
Now we can try to observe the network with the cities of the stations.

In [9]:
```
# Give a color to the cities and the bus-stop

couleurs = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:
n_col = len(couleurs)
communes_nodes = dict(nx.get_node_attributes(G_contracte,'commune')
communes = list(communes_nodes.values())
communes = np.array(communes)
communes_u = np.unique(communes)
communes_u
couleurs_nodes = dict()

for key in communes_nodes :
    #print(key)
    #print(np.where(communes_u==communes_nodes[key]))
    couleurs_nodes[key] = couleurs[np.where(communes_u==communes_no

#couleurs_nodes
```

In [10]:
```
#Plot it
nx.draw(G_contracte, pos = nx.get_node_attributes(G_contracte,'loca
```

This representation with cities seems to be working on our network. Now we can better work on the vizualisation to make it easier to read.

In [11]:
```
nodes = nx.draw_networkx_nodes(G_contracte, pos = nx.get_node_attri
# Set edge color to black
nodes.set_edgecolor('black')
nodes.set_linewidth(0.3)
nx.draw_networkx_edges(G_contracte, pos = nx.get_node_attributes(G_

plt.tight_layout()
plt.show()
```



First analysis

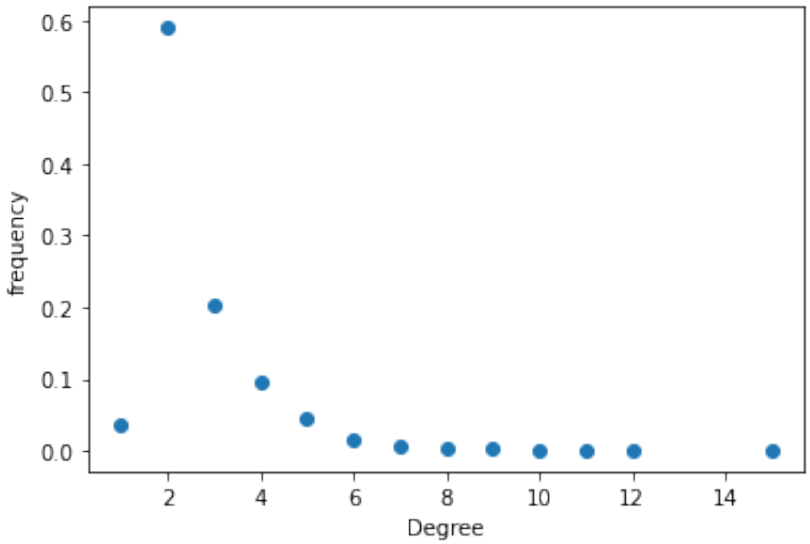
Now that we got the network and that we were able to vizualise it, we can start the analysis of the network. In this part we will make a first approach analysis to get aquainted with the network and try to understand the meaning of the results in term of transport network.

In [12]:
```
#Number of nodes and edges

N_nodes = len(G_contracte.nodes())
N_edges = len(G_contracte.edges())

print('This graph gets ', N_nodes, ' bus-tram-subway stations and',
```

```
This graph gets  2058  bus-tram-subway stations and 2739  liaisons
between stations.
```

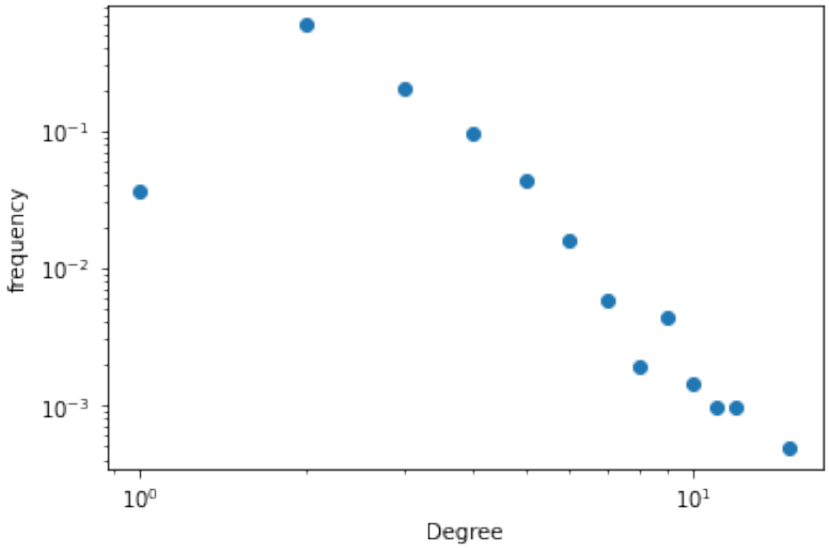
Now we will look at the degree distribution of the network.

In [13]:
```
# Plotting the degree distribution

c = Counter(np.array(list(dict(G_contracte.degree()).values())))
degree = np.array(list(dict(c).keys()))
freq = np.array(list(dict(c).values()))/sum(list(dict(c).values()))

plt.plot(degree, freq, 'o')
plt.xlabel('Degree')
plt.ylabel('frequency')

plt.show()
```



It appears that most of the stations have only two connections, which means that there is only one line crossing them. The ones with one connection are the first/last stops of the lines. We can see that most of the stations have less than 6 connections. The decrease seems to be strongly exponential as we can see in this loglog plot:

In [14]:
```
plt.loglog(degree, freq, 'o')
plt.xlabel('Degree')
plt.ylabel('frequency')

plt.show()
```



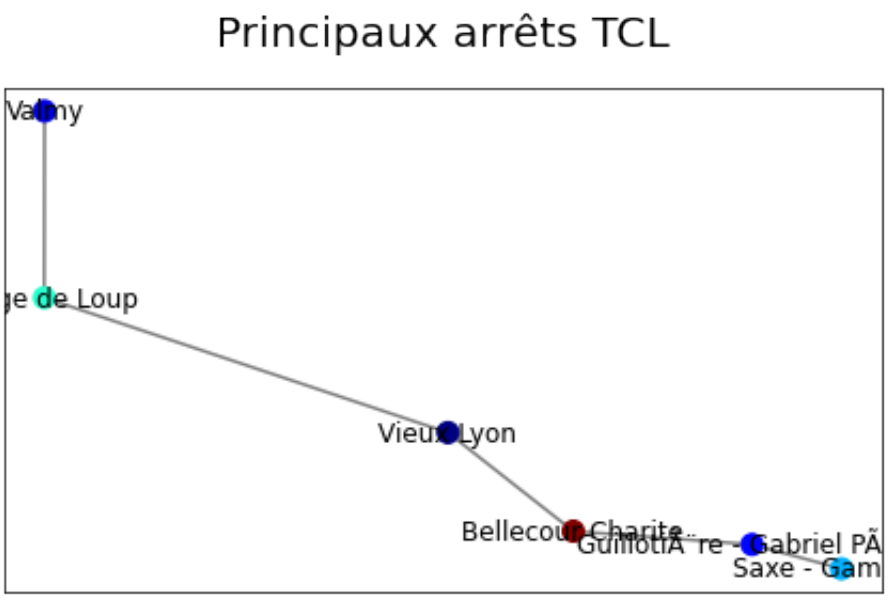
Going further: How to recover real TCL hubs?

Now that we got a first graph that seems to represent well the TCL network, we want to see if networks tools allow us to recover important information that have sense in the 'real network'.

Betweeness Centrality

In [15]:
```
#get centrality
centrality = nx.betweenness_centrality(G_contracte)

#get only the stops with the highest centrality coef
data = dict(centrality)
data_a = np.array(list(data))
data_values_a = np.array(list(data.values()))

seuil = 0.2

data_f = data_a[data_values_a > np.max(data_values_a) - seuil]
data_val = data_values_a[data_values_a > np.max(data_values_a) - se

G_principal = nx.subgraph(G_contracte, data_f)
graph = G_principal

#we get the colors
couleur = []
for node in G_principal.nodes():
    i = np.where(data_f == node)[0][0]
    couleur.append(data_val[i])

print(data_f)
print(data_val)

#plot
nx.draw_networkx(graph, node_size=100, width=1.5, node_color=couleu
plt.suptitle('Principaux arrêts TCL', fontsize = 20)
plt.tight_layout()
```

```
['Bellecour Charite' 'Gorge de Loup' 'Saxe - Gambetta' 'Valmy'
 'GuillotiÃ¨re - Gabriel PÃ©ri' 'Vieux Lyon']
[0.4284305  0.32220751 0.30401297 0.26691756 0.27417294 0.25121106
]
```

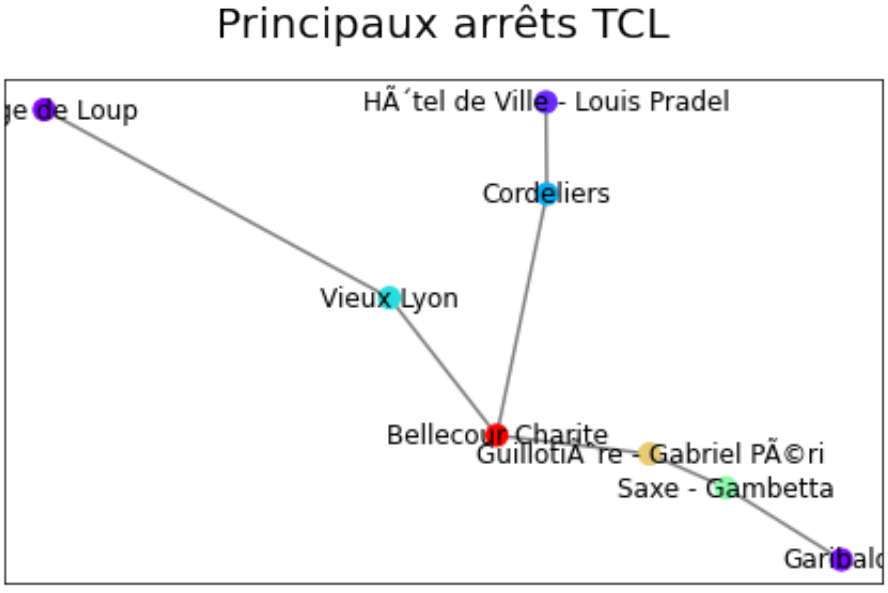


This method gives us metro D stations. As this metro-line has been created after the other, it appears that TCL had chosen to link all the important hub of the network to improve the trajectories.

Closeness centrality

In [16]:
```
#get centrality
centrality = nx.closeness_centrality(G_contracte)

#get only the stops with the highest centrality coef
data = dict(centrality)
data_a = np.array(list(data))
data_values_a = np.array(list(data.values()))

seuil = 0.004

data_f = data_a[data_values_a > np.max(data_values_a) - seuil]
data_val = data_values_a[data_values_a > np.max(data_values_a) - se

G_principal = nx.subgraph(G_contracte, data_f)
graph = G_principal


couleur = []
for node in G_principal.nodes():
    i = np.where(data_f == node)[0][0]
    couleur.append(data_val[i])

print(data_f)
print(data_val)

#plot
nx.draw_networkx(graph, node_size=100, width=1.5, node_color=couleu
plt.suptitle('Principaux arrêts TCL', fontsize = 20)
plt.tight_layout()
```

```
['Cordeliers' 'HÃ´tel de Ville - Louis Pradel' 'Bellecour Charite'
 'Gorge de Loup' 'Saxe - Gambetta' 'GuillotiÃ¨re - Gabriel PÃ©ri'
 'Vieux Lyon' 'Garibaldi']
[0.07972868 0.07900296 0.08273671 0.07878811 0.08084421 0.08156872
 0.08010749 0.07883642]
```

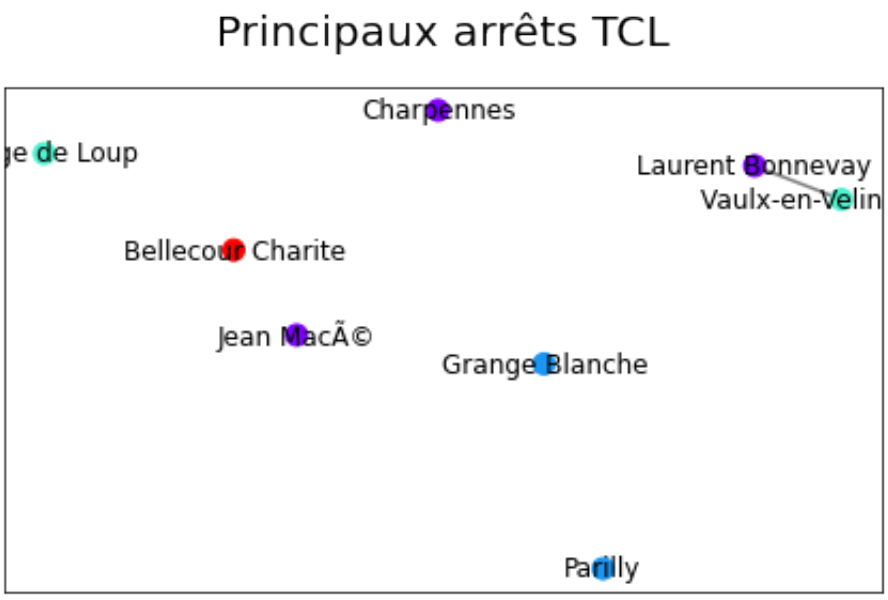


As for the betweeness centrality method gives us most of the metro line D and important stations of the "Presqu'Ile".

Degree

In [17]:
```
centrality = nx.degree(G_contracte)

data = dict(centrality)

data_a = np.array(list(data))
data_values_a = np.array(list(data.values()))

seuil = 6

data_f = data_a[data_values_a > np.max(data_values_a) - seuil]
data_val = data_values_a[data_values_a > np.max(data_values_a) - se

G_principal = nx.subgraph(G_contracte, data_f)
graph = G_principal

couleur = []
for node in G_principal.nodes():
    i = np.where(data_f == node)[0][0]
    couleur.append(data_val[i])

print(data_f)
print(data_val)

#plot
nx.draw_networkx(graph, node_size=100, width=1.5, node_color=couleu
plt.suptitle('Principaux arrêts TCL', fontsize = 20)
plt.tight_layout()
```

```
['Jean MacÃ©' 'Bellecour Charite' 'Grange Blanche' 'Charpennes' 'P
arilly'
 'Gorge de Loup' 'Laurent Bonnevay' 'Vaulx-en-Velin La Soie']
[10 15 11 10 11 12 10 12]
```

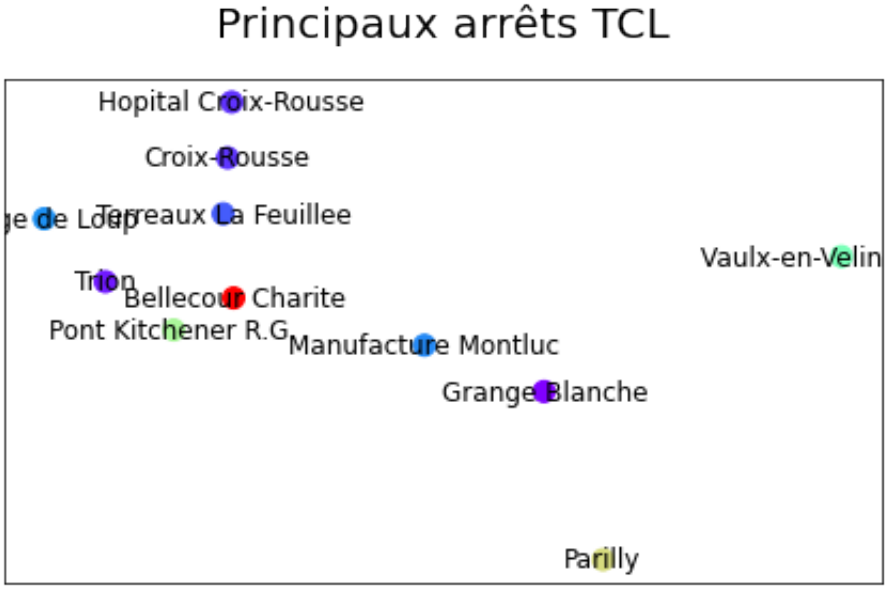


The degree method gives none linked stops (except for 2 stations). This is due to the fact that degree distribution only depends on a node and its number of neighbours, and this property does not propagate around the network as it could be the case for some other properties. This measurement gives us independant local hubs and not a central hub composed of several stops.

Pagerank

In [18]:
```
Dd=nx.get_node_attributes(G, 'densite_arret')
centrality = nx.pagerank(G_contracte,weight='weight') #nx.pagerank(

data = dict(centrality)

data_a = np.array(list(data))
data_values_a = np.array(list(data.values()))

seuil = 0.00055 #0.00055

data_f = data_a[data_values_a > np.max(data_values_a) - seuil]
data_val = data_values_a[data_values_a > np.max(data_values_a) - se

G_principal = nx.subgraph(G_contracte, data_f)
graph = G_principal

couleur = []
for node in G_principal.nodes():
    i = np.where(data_f == node)[0][0]
    couleur.append(data_val[i])

print(data_f)
print(data_val)


nx.draw_networkx(graph, node_size=100, width=1.5, node_color=couleu
plt.suptitle('Principaux arrêts TCL', fontsize = 20)
plt.tight_layout()
```

```
['Bellecour Charite' 'Grange Blanche' 'Parilly' 'Gorge de Loup'
 'Terreaux La Feuillee' 'Pont Kitchener R.G.' 'Trion'
 'Manufacture Montluc' 'Vaulx-en-Velin La Soie' 'Croix-Rousse'
 'Hopital Croix-Rousse']
[0.00170137 0.00116254 0.00152452 0.00126777 0.00122663 0.00147725
 0.00117951 0.00126126 0.00142484 0.00119994 0.00119811]
```



As pagefrank is based on a random walk, we expect to get with this method local hubs of the network.This method seems to be comparable to the degree method : we get local hubs which are not linked.

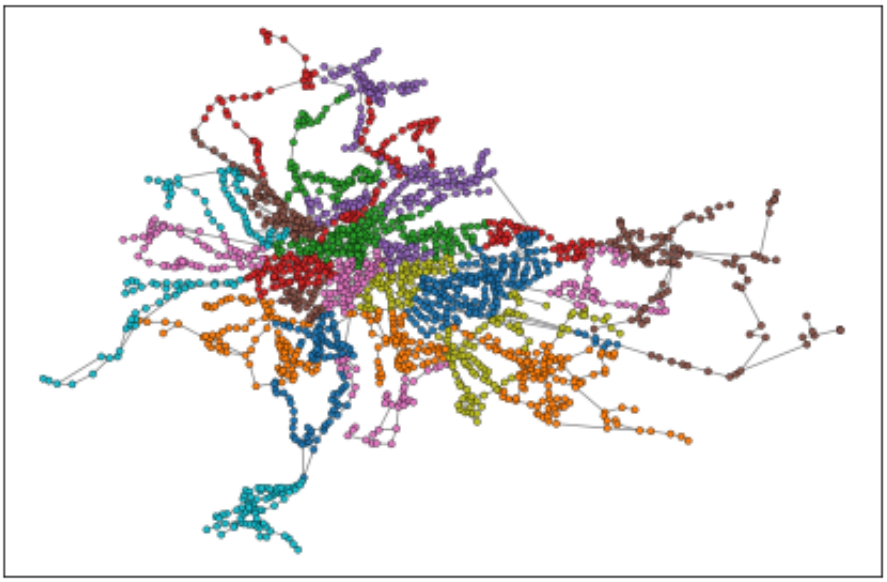
Communities

In this part, we will see if we can find communities that fit with spatial pattern of our network (like cities).

We first use the Louvain Algorithm.

In [19]:
```
G_com = G_contracte

#Louvain communities detection
com = algorithms.louvain(G_com)

com_list = NodeClustering.to_node_community_map(com)
nx.set_node_attributes(G_com, com_list, 'com')


#We get the communities that we are interested in
com_stops = np.array(list(com_list))
com_com = np.array(list(com_list.values()))[:,0]

com_seuil = 34
arrets = com_stops[com_com < com_seuil]

#create subgraph
G_com_seuil = nx.subgraph(G_com, arrets)
#nx.get_node_attributes(G_com_seuil,'commune')
```

In [20]:
```
#vizualization
##############

#get communities colors
couleurs = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:
n_col = len(couleurs)
com_nodes = dict(nx.get_node_attributes(G_com_seuil,'com'))
com_v = list(com_nodes.values())
com_v = np.array(com_v)
com_u = np.unique(com_v)
com_u
#np.where(communes_u=='BRIGNAIS')
couleurs_nodes = dict()
for key in com_nodes :
    #print(key)
    #print(np.where(communes_u==communes_nodes[key]))
    couleurs_nodes[key] = couleurs[np.where(com_u==com_nodes[key])[

nodes = nx.draw_networkx_nodes(G_com_seuil, pos = nx.get_node_attri
# Set edges color to red
nodes.set_edgecolor('black')
nodes.set_linewidth(0.3)
nx.draw_networkx_edges(G_com_seuil, pos = nx.get_node_attributes(G_
# Uncomment this if you want your labels
## draw_networkx_labels(G, pos)
plt.tight_layout()
plt.show()
```

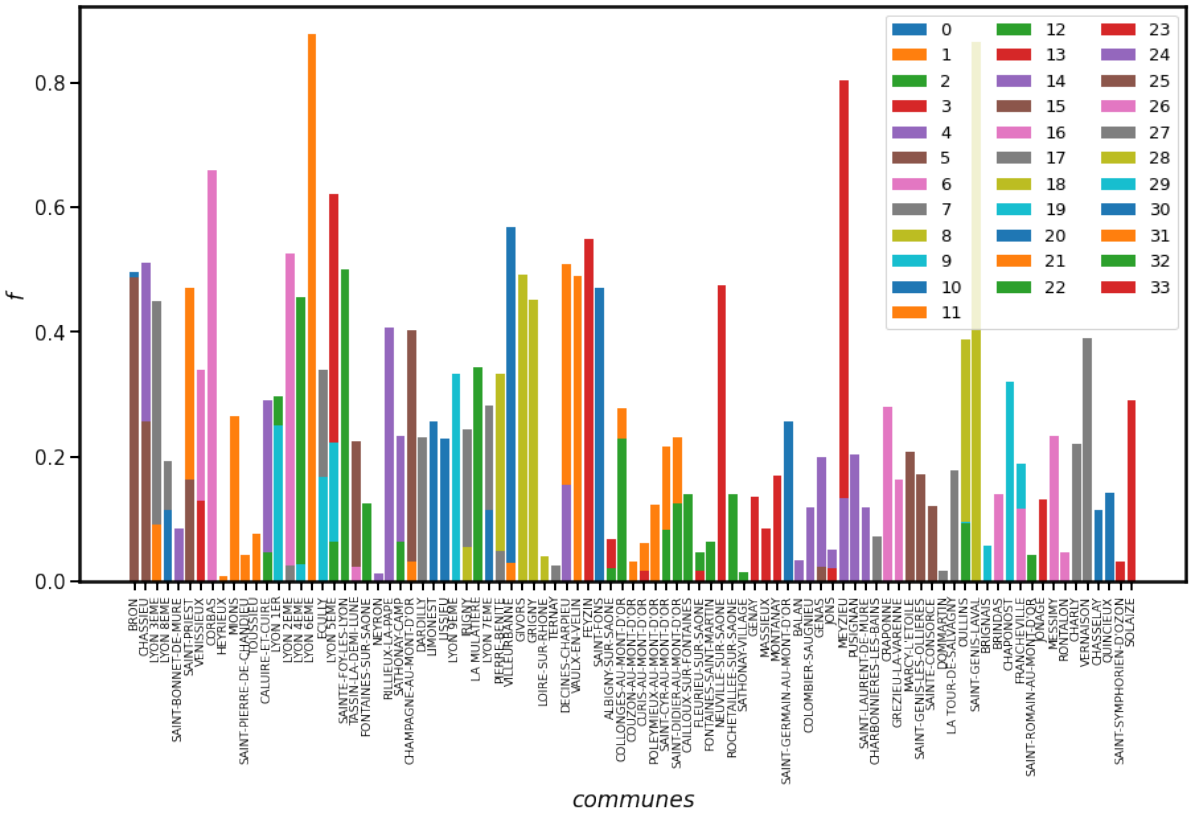


The Louvain algorithm gives us 34 communities that appear to be defined by a spatial distribution. We want to see if it is linked with the cities distributions.

In [21]:
```
def hist_com(communes, normalisation=True):
    """
    INPUT
    -----
    communes : list of str with communes names
    _____________________________________________
    OUTPUT:
    ------
    commune_u : list of unique communes
    frequency : frequency of appearance
    """

    #creation of the data
    communes_u = np.unique(np.array(communes))
    freq_com = []

    #iterations on the list
    for name in communes_u :
        freq = communes.count(name)
        freq_com.append(freq)

    freq_com = np.array(freq_com)

    #normalization
    if normalisation:
        freq_com = freq_com/np.sum(freq_com)

    return communes_u, freq_com
```

In [22]:
```
#We create all the histogram distribution for each community
communes_tot = []
communes_freq_tot = []
#we iterate on the communities
for com_name in range(len(com_u)): #10
    arrets_loc = com_stops[com_com==com_name]
    G_loc = nx.subgraph(G_com,arrets_loc)
    commune_loc = nx.get_node_attributes(G_loc,'commune')
    communes_u, freq_com = hist_com(list(commune_loc.values()))
    communes_tot.append(communes_u)
    communes_freq_tot.append(freq_com)
```

In [23]:
```
#Plot of the histograms

fig =plt.figure(figsize = (15,8))
plt.rcParams['axes.linewidth']=2.5

ax = fig.add_subplot(1,1,1)

for i in range(34):#range(len(communes_tot)):
    #ax.hist(communes_tot[i], communes_freq_tot[i], label = i)
    ax.bar(communes_tot[i],communes_freq_tot[i],align='center', lab

ax.set_xlabel('communes', style = 'italic', fontsize = 17)
ax.set_ylabel('f', style = 'italic', fontsize = 17)
ax.legend(ncol = 3, fontsize = 13)
ax.tick_params(axis = 'x', which = 'both',labelsize = 9, width = 2,
ax.tick_params(axis = 'y', which = 'both' ,labelsize = 15, width =
```



With this graph we can not conclude anything. The first thing is that we got 34 communities for 92 cities, if there are some correlations, communities should contain several cities. Some of the communities appear to be alone on their cities but for others like in Lyon they cover several districts and each district is in several communities. At first, we can have the impression that this works better for cities outside Lyon. This is coherent since suburbs may contain town with a local organisation while in Lyon everything is interconnected.

In order to give more sense to the community detection, we will uniformize it in each district and city: we will put each district/city in the community that is in majority in the city/district.

In [24]:
```
#Creation of a dictionary to give to each city-district a community
communes_dic = {}
commune_u = np.unique(np.array(list(dict(nx.get_node_attributes(G_c

#Get cities of the stations and communities
communes_tot = np.array(list(dict(nx.get_node_attributes(G_com,'com
communities_tot = np.array(list(dict(nx.get_node_attributes(G_com,'

#iteration on the city-districts names
for commune_name in commune_u :
    communities_loc = communities_tot[communes_tot == commune_name]
    communities_u, freq_com = hist_com(list(communities_loc), norma
    community_loc = communities_u[np.argmax(freq_com)]
    communes_dic[commune_name] = community_loc

print(communes_dic)

#count new number of communities
num_b_comu = len(np.unique(list(dict(communes_dic).values())))
print('\n New number of communities : ', num_b_comu)
```

```
{'ALBIGNY-SUR-SAONE': 13, 'BALAN': 14, 'BRIGNAIS': 19, 'BRINDAS':
26, 'BRON': 0, 'CAILLOUX-SUR-FONTAINES': 12, 'CALUIRE-ET-CUIRE': 4
, "CHAMPAGNE-AU-MONT-D'OR": 5, 'CHAPONOST': 19, 'CHARBONNIERES-LES
-BAINS': 17, 'CHARLY': 27, 'CHASSELAY': 30, 'CHASSIEU': 24, "COLLO
NGES-AU-MONT-D'OR": 11, 'COLOMBIER-SAUGNIEU': 14, 'CORBAS': 16, "C
OUZON-AU-MONT-D'OR": 11, 'CRAPONNE': 26, "CURIS-AU-MONT-D'OR": 11,
'DARDILLY': 17, 'DECINES-CHARPIEU': 21, 'DOMMARTIN': 17, 'ECULLY':
17, 'FEYZIN': 33, 'FLEURIEU-SUR-SAONE': 12, 'FONTAINES-SAINT-MARTI
N': 12, 'FONTAINES-SUR-SAONE': 12, 'FRANCHEVILLE': 19, 'GENAS': 14
, 'GENAY': 13, 'GIVORS': 8, 'GREZIEU-LA-VARENNE': 26, 'GRIGNY': 8,
'HEYRIEUX': 1, 'IRIGNY': 27, 'JONAGE': 23, 'JONS': 14, 'LA MULATIE
RE': 6, 'LA TOUR-DE-SALVAGNY': 17, 'LIMONEST': 5, 'LISSIEU': 30, '
LOIRE-SUR-RHONE': 8, 'LYON 1ER': 2, 'LYON 2EME': 6, 'LYON 3EME': 7
, 'LYON 4EME': 2, 'LYON 5EME': 3, 'LYON 6EME': 31, 'LYON 7EME': 7,
'LYON 8EME': 0, 'LYON 9EME': 5, "MARCY-L'ETOILE": 15, 'MASSIEUX':
13, 'MESSIMY': 26, 'MEYZIEU': 23, 'MIONS': 1, 'MONTANAY': 13, 'NEU
VILLE-SUR-SAONE': 13, 'NEYRON': 4, 'OULLINS': 18, 'PIERRE-BENITE':
18, "POLEYMIEUX-AU-MONT-D'OR": 11, 'PUSIGNAN': 14, 'QUINCIEUX': 30
, 'RILLIEUX-LA-PAPE': 4, 'ROCHETAILLEE-SUR-SAONE': 12, 'RONTALON':
26, 'SAINT-BONNET-DE-MURE': 14, "SAINT-CYR-AU-MONT-D'OR": 11, "SAI
NT-DIDIER-AU-MONT-D'OR": 11, 'SAINT-FONS': 10, 'SAINT-GENIS-LAVAL'
: 28, 'SAINT-GENIS-LES-OLLIERES': 15, "SAINT-GERMAIN-AU-MONT-D'OR"
: 30, 'SAINT-LAURENT-DE-MURE': 14, 'SAINT-PIERRE-DE-CHANDIEU': 1,
'SAINT-PRIEST': 1, "SAINT-ROMAIN-AU-MONT-D'OR": 22, "SAINT-SYMPHOR
IEN-D'OZON": 33, 'SAINTE-CONSORCE': 15, 'SAINTE-FOY-LES-LYON': 3,
'SATHONAY-CAMP': 4, 'SATHONAY-VILLAGE': 12, 'SOLAIZE': 33, 'TASSIN
-LA-DEMI-LUNE': 15, 'TERNAY': 8, 'TOUSSIEU': 1, 'VAULX-EN-VELIN':
21, 'VENISSIEUX': 16, 'VERNAISON': 27, 'VILLEURBANNE': 20}

 New number of communities :  30
```

We lost 3 communities with this approximation.

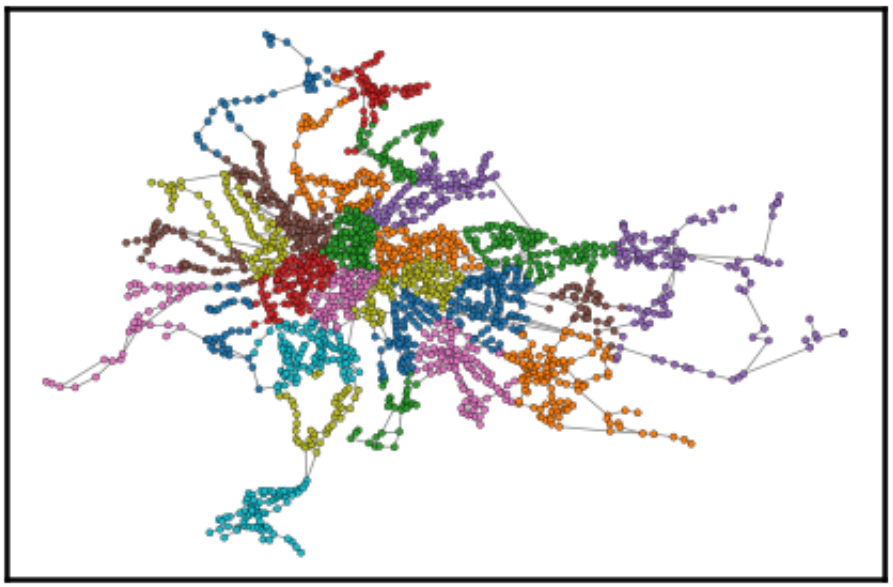
Now, we plot the new communities distribution.

In [25]:
```
G_com.nodes()['Debourg']['commune']
```

Out[25]:
```
'LYON 7EME'
```

In [26]:
```
#Iteration on all the nodes of the graph
#########################################

#dictionnary of the new communities for the nodes
new_com = {}

nodes_tot = list(G_com.nodes())

for name in nodes_tot :
    city_name = G_com.nodes()[name]['commune']
    new_com[name] = communes_dic[city_name]

#print(new_com)

#now we add those new communities to the graph
nx.set_node_attributes(G_com, new_com, 'com_bis')
```

In [27]:
```
#vizualization
##############

#get communities colors
couleurs = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:
n_col = len(couleurs)
com_nodes = dict(nx.get_node_attributes(G_com_seuil,'com_bis'))
com_v = list(com_nodes.values())
com_v = np.array(com_v)
com_u = np.unique(com_v)
com_u
#np.where(communes_u=='BRIGNAIS')
couleurs_nodes = dict()
for key in com_nodes :
    #print(key)
    #print(np.where(communes_u==communes_nodes[key]))
    couleurs_nodes[key] = couleurs[np.where(com_u==com_nodes[key])[

nodes = nx.draw_networkx_nodes(G_com_seuil, pos = nx.get_node_attri
# Set edge color to red
nodes.set_edgecolor('black')
nodes.set_linewidth(0.3)
nx.draw_networkx_edges(G_com_seuil, pos = nx.get_node_attributes(G_
# Uncomment this if you want your labels
## draw_networkx_labels(G, pos)
plt.tight_layout()
plt.show()
```



Using this graph it is difficult to see a difference with the first one. We want to know where are the nodes that changed from one community to another to quantify how much there is and to see where they are in majority.

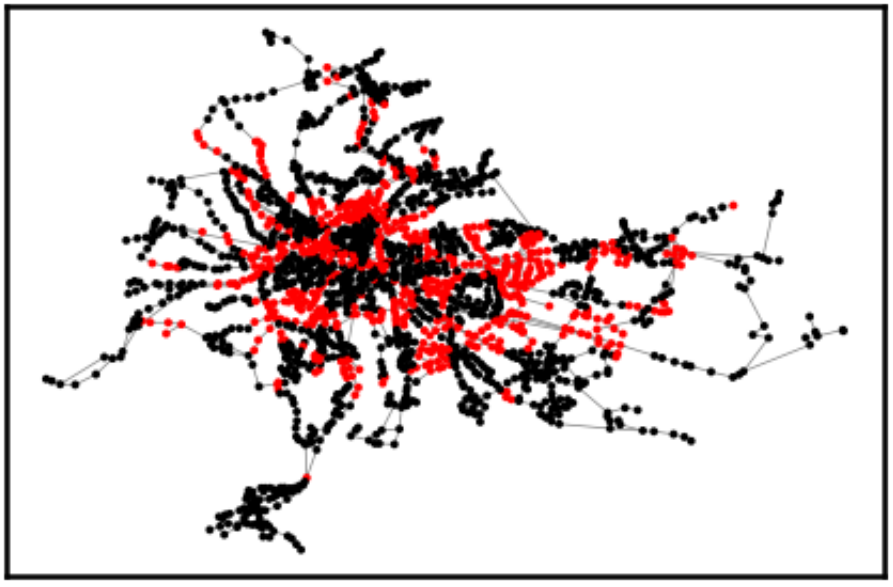
In [28]:
```
communities = np.array(list(dict(nx.get_node_attributes(G_com, 'com
communities = np.reshape(communities, len(communities))
communities_bis = np.array(list(dict(nx.get_node_attributes(G_com,
```

In [29]:
```
communities_indice = (communities==communities_bis)

print('There is ', np.sum(communities_indice)/len(communities_indic
```

```
There is  69.67930029154519  % of the stops that did not change of
community
```

In [30]:
```
#Creation of the color
color_indice = []
for item in communities_indice :
    if item :
        color_indice.append('black')
    else :
        color_indice.append('red')
```

In [31]:
```
nodes = nx.draw_networkx_nodes(G_com, pos = nx.get_node_attributes(
# Set edge color to red
nodes.set_edgecolor('black')
nodes.set_linewidth(0)
nx.draw_networkx_edges(G_com, pos = nx.get_node_attributes(G_com,'l
# Uncomment this if you want your labels
## draw_networkx_labels(G, pos)
plt.tight_layout()
plt.show()
```



In this graph the stations in red are the stations that changed community with this new approximation. It is difficult to conclude a correlation between network communities and cities/districts. There are a lot of red stations (stations that changed community with the new method), this means that the network seems not to take into account the city/district of the stations. However, the further away from the centre of Lyon we get, the more the community detection seems to be correlated with the city/district. This is coherent with our intuition : We get away from Lyon, each line becomes a city to deliver. In addition, for the cities/districts distribution we gave to each station the one with its barycenter the closest to the station. As this is an approximation, the result with the communities could be biased by it.

Shortest path: Does-it make sense?

As we are working on a public transport network, the question of the shortest path seems to be relevant. In this network we can calculate it.

In [32]:
```
#get average shortest path :
shortest_path_av = nx.average_shortest_path_length(G_com)

print('In this graph the average shortest path is ', shortest_path_
```

```
In this graph the average shortest path is  20.248198452934894 edg
es long.
```

In general, you have to wait 20 stations to go from a bus stop to another in Lyon. This result is not false, but not adapted to the question of itinerary that we encounter in reality. This shortest path does not take into account the bus/metro/tram lines and to use this shortest path one might have to change the line at several stations which is not very convenient. To compute a more realistic shortest path taking into account the lines we need to update the network: this is what we do in the next part of the project.

Second analysis: How to use the TCL network?

As we saw in the first network ananlysis, we can get some results about the network organisation and spatial organisation, but we can not use it to work on real routes. To be able to find itineraries, we created a second network, more complex but closer to reality.

Importation of the librairies and the DATA

In [33]:
```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import os
#os.chdir('C:\\Users\\KadBa\\Desktop\\Complex_Networks')

G=nx.readwrite.graphml.read_graphml('tcl2.graphml')
```

Finding the best itineraries

We will test the directed graph to find several itineraries on the TCL map.

Here, we will give you 3 examples. For each example we give itineraries that we get with the shortest path and the network with the stations it goes through and the time it takes.

Debourg - ENS Lyon

First, we choose a trivial example.

In [34]:
```
# Itinerary
nx.shortest_path(G,'T1A:Debourg','T1A:ENS Lyon',weight='period')
```

Out[34]:
```
['T1A:Debourg', 'T1A:ENS Lyon']
```

In [35]:
```
# shortest path length : time
nx.shortest_path_length(G,'T1A:Debourg','T1A:ENS Lyon',weight='peri
```

Out[35]:
```
1.0
```

La Doua - Gaston Berger-ENS Lyon

In [36]:
```
# Itinerary
list(nx.all_shortest_paths(G,'T4A:La Doua - Gaston Berger','T1A:ENS
```

Out[36]:
```
[['T4A:La Doua - Gaston Berger',
  'T4A:UniversitÃ© Lyon 1',
  'T4A:Condorcet',
  'T4A:Le Tonkin',
  'T4A:Charpennes',
  'MB:Charpennes',
  'MB:Brotteaux',
  'MB:Gare Part-Dieu Vivier Merle',
  'MB:Place Guichard',
  'MB:Saxe - Gambetta',
  'MB:Jean MacÃ©',
  'MB:Place Jean JaurÃ¨s',
  'MB:Debourg',
  'T1A:Debourg',
  'T1A:ENS Lyon']]
```

In [37]:
```
# shortest path length : time
nx.shortest_path_length(G,'T4A:La Doua - Gaston Berger','T1A:ENS Ly
```

Out[37]:
```
19.0
```

For this trajectory, the itinerary is what the TLC app would give us. The time is estimated to be half of the real travel time.

C16A:Surville Route de Vienne - MC:Croix-Rousse

In [38]:
```
# Itineraries
list(nx.all_shortest_paths(G,'C16A:Surville Route de Vienne','MC:Cr
```

Out[38]:
```
[['C16A:Surville Route de Vienne',
  'C16A:Rue du Sablon',
  'C16A:Professeur Roux',
  'C16A:Place Ennemond Romand',
  'C16A:College Balzac',
  'C16A:Beauvisage - PressensÃ©',
  'C16A:Beauvisage CISL',
  'C16A:Etats-Unis Tony Garnier',
  'C16A:LycÃ©e LumiÃ¨re',
  'C16A:Cazeneuve - Berthelot',
  'C16A:Cazeneuve - Berliet',
  'C16A:Saint-Mathieu',
  'C16A:Place Ambroise Courtois',
  'C16A:Feuillat Freres Lumiere',
  'C16A:Feuillat - Albert Thomas',
  'C16A:Monplaisir LumiÃ¨re',
  'MD:Monplaisir LumiÃ¨re',
  'MD:Sans Souci',
  'MD:Garibaldi',
  'MD:Saxe - Gambetta',
  'MD:GuillotiÃ¨re - Gabriel PÃ©ri',
  'MD:Bellecour',
```

```
 'MA:Bellecour',
 'MA:Cordeliers',
 'MA:HÃ´tel de Ville - Louis Pradel',
 'MC:HÃ´tel de Ville - Louis Pradel',
 'MC:Croix Paquet',
 'MC:Croix-Rousse'],
['C16A:Surville Route de Vienne',
 'C16A:Rue du Sablon',
 'C16A:Professeur Roux',
 'C16A:Place Ennemond Romand',
 'C16A:College Balzac',
 'C16A:Beauvisage - PressensÃ©',
 'C16A:Beauvisage CISL',
 'C16A:Etats-Unis Tony Garnier',
 'C16A:LycÃ©e LumiÃ¨re',
 'T4A:LycÃ©e LumiÃ¨re',
 "T4A:Jet d'Eau - Mendes France",
 'T4A:Lycee Colbert',
 'T4A:Manufacture Montluc',
 'T4A:Archives Departementales',
 'T4A:Gare Part-Dieu Villette',
 'T4A:Thiers - Lafayette',
 'T4A:CollÃ¨ge Bellecombe',
 'T4A:Charpennes',
 'MA:Charpennes',
 'MA:MassÃ©na',
 'MA:Foch',
 'MA:HÃ´tel de Ville - Louis Pradel',
 'MC:HÃ´tel de Ville - Louis Pradel',
 'MC:Croix Paquet',
 'MC:Croix-Rousse'],
['C16A:Surville Route de Vienne',
 'C16A:Rue du Sablon',
 'C16A:Professeur Roux',
 'C16A:Place Ennemond Romand',
 'C16A:College Balzac',
 'C16A:Beauvisage - PressensÃ©',
 'C16A:Beauvisage CISL',
 'C16A:Etats-Unis Tony Garnier',
 'T4A:Etats-Unis Tony Garnier',
 'T4A:LycÃ©e LumiÃ¨re',
 "T4A:Jet d'Eau - Mendes France",
 'T4A:Lycee Colbert',
 'T4A:Manufacture Montluc',
 'T4A:Archives Departementales',
 'T4A:Gare Part-Dieu Villette',
 'T4A:Thiers - Lafayette',
 'T4A:CollÃ¨ge Bellecombe',
 'T4A:Charpennes',
 'MA:Charpennes',
 'MA:MassÃ©na',
 'MA:Foch',
 'MA:HÃ´tel de Ville - Louis Pradel',
 'MC:HÃ´tel de Ville - Louis Pradel',
 'MC:Croix Paquet',
 'MC:Croix-Rousse'],
['C16A:Surville Route de Vienne',
```

```
 'C16A:Rue du Sablon',
 'C16A:Professeur Roux',
 'C16A:Place Ennemond Romand',
 'C16A:College Balzac',
 'C16A:Beauvisage - PressensÃ©',
 'C16A:Beauvisage CISL',
 'T4A:Beauvisage CISL',
 'T4A:Etats-Unis Tony Garnier',
 'T4A:LycÃ©e LumiÃ¨re',
"T4A:Jet d'Eau - Mendes France",
 'T4A:Lycee Colbert',
 'T4A:Manufacture Montluc',
 'T4A:Archives Departementales',
 'T4A:Gare Part-Dieu Villette',
 'T4A:Thiers - Lafayette',
 'T4A:CollÃ¨ge Bellecombe',
 'T4A:Charpennes',
 'MA:Charpennes',
 'MA:MassÃ©na',
 'MA:Foch',
 'MA:HÃ´tel de Ville - Louis Pradel',
 'MC:HÃ´tel de Ville - Louis Pradel',
 'MC:Croix Paquet',
 'MC:Croix-Rousse']]
```

In this case we get 3 shortest paths that are quite equivalent.

In [39]:
```
# shortest path length : time
nx.shortest_path_length(G,'C12A:Surville Route de Vienne','MC:Croix
```

Out[39]:
```
24.0
```

We can see that the chosen path is quite representative of the truth (see citymaper App or TCL website) but once again the time estimation is about half as much as the real time. Studying a large amount of itineraries may confirm or not this factor 2 between the estimation and the realistic estimate. If this factor would not change much, we may get a reasonnable approximation by changing the time scales arbitrarily chosen. We did not perform this study.

Then, you can test the network for any initinerary that you want.

In [40]:
```
#here are the name of all the station, with their line number befor
G.nodes()
```

Out[40]:
```
NodeView(('77A:Montessuy Gutenberg', '77A:Caluire Place Foch', '
77A:Caluire - Pl. Bascule', '77A:Andre Lassagne', '77A:Cedres',
'77A:Le Vernay', '77A:Bois Roux', '77A:Marronniers', '77A:Les Br
uyeres', '77A:Camp Militaire', '77A:Sathonay Mutualite', '77A:Sa
thonay Gare', '77A:Boutarey', '77A:Sathonay Village Eglise', '77
A:Centre Village', '77A:Les Epinettes', '77A:Chemin de Rivery',
'77A:La Buissonniere', '77A:Noailleux', '77A:Cailloux Chateau',
'77A:Cailloux Mairie', '77A:Diligences', '77A:La Dangereuse', '7
7A:GAEC de la Grive', '77A:Le Four', '77A:Les Pervenches', '77A:
Cailloux Centre', '77A:Cailloux Bellevue', '77A:Cailloux La Croi
x', '77A:Treve Oray', '77A:Fontaines-St Martin Ctre', '77A:Le Ca
ntin', '77A:Les Mollieres', '77A:Norenchal', '77A:Fontaines Cent
re', '77A:Pont de Fontaines', '9A:Cordeliers', '9A:HÃ´tel de Vil
le - Louis Pradel', '9A:Pont De Lattre RD', '9A:Pont Churchill R
D', '9A:Cours Aristide Briand', '9A:Bellevue', '9A:Petit Versail
les', '9A:Les Eaux', '9A:Grande Rue de St Clair', '9A:St-Clair S
quare Brosset', '9A:Montee des Soldats', '9A:Caluire Place Foch'
, '9A:Caluire - Pl. Bascule', '9A:Andre Lassagne', '9A:Cedres',
'9A:Le Vernay', '9A:Bois Roux', '9A:Marronniers', '9A:Les Bruyer
```

In [41]:
```
list(nx.all_shortest_paths(G,'MD:Grange Blanche','T1A:ENS Lyon',wei
```

Out[41]:
```
[['MD:Grange Blanche',
  'MD:Monplaisir LumiÃ¨re',
  'MD:Sans Souci',
  'MD:Garibaldi',
  'MD:Saxe - Gambetta',
  'MB:Saxe - Gambetta',
  'MB:Jean MacÃ©',
  'MB:Place Jean JaurÃ¨s',
  'MB:Debourg',
  'T1A:Debourg',
  'T1A:ENS Lyon']]
```

Random walk and spatial organisation

A question that we tried to answer with this study was: can we find spatial patterns in Lyon where people work and/or where people live. In this part, we use a biased random walk to try to answer this question.

Importation of the DATA

In [42]:
```
#We open the graph G and set the list of nodes, edges and dictionna
G=nx.readwrite.graphml.read_graphml('tcl.graphml')
Ln=list(G.nodes())
N=nx.number_of_nodes(G)
Dd=nx.get_node_attributes(G, 'densite_arret')
Dv=nx.get_node_attributes(G, 'commune')

Ne=nx.number_of_edges(G)
Le=list(G.edges())
Dw=nx.get_edge_attributes(G,'weight')
```

Preparation of the simulation

We will distribute randomly initial positions of people on the graph with respect to the approximation of population around each station.

In [43]:
```
pop_init_position=[]
for k in range(N):
    E=int(Dd[Ln[k]])
    for i in range(E):
        pop_init_position.append(k)
```

The goal will be to see where those people go after a certain amount of time.

First, we create a list neighbors in which we put all the neighbors of each nodes. We have to correct the tuple as the order given by the function G.edges(node) is not necessarily the same as the one in the whole list of edges Le.

In [44]:
```
neighbors=[[] for k in range(N)]
for i in range(N):
    neighbors[i]=list(G.edges(Ln[i]))
    nb_neighbors=len(neighbors[i])
    errors=[]
    for k in range(nb_neighbors):
        if neighbors[i][k] not in Le:
            neighbors[i].append((neighbors[i][k][1],neighbors[i][k]
            errors.append(k)
    for l in range(len(errors)):
        neighbors[i].remove(neighbors[i][errors[l]-l])
```

Then, we create a list connection in which we put the index of each neighbors of each node with an occurence depending on the weight of the connection. For example, if you are in a metro station, we state that you are more likely to go on with a metro than a simple bus.

In [45]:
```
connection=[[] for k in range(N)]
for i in range(N):
    index_neighbors=[]
    nodes_neighbors=list(G.neighbors(Ln[i]))
    for k in range(N):
        if Ln[k] in nodes_neighbors:
            index_neighbors.append(k)
    nb_neighbors=len(index_neighbors)
    for j in range(nb_neighbors):
        w=int(10/Dw[neighbors[i][j]])
        for k in range(w):
            connection[i].append(index_neighbors[j])
```

Simulation

Now, we do the simulation. Let's choose a value for the total time T and the population "doing" the experiment.

In [46]:
```
T=60
pop=10000
```

For each "person" we track the node in which one is at each time t. We impose that each "person" does not go twice in the same station.

In [47]:
```
chemins=np.zeros((pop,T))
chemins[:,0]=np.random.choice(pop_init_position,pop)
chemins=chemins.astype(int)
for t in range(T-1):
    for k in range(pop):
        id_neighb=connection[chemins[k,t]].copy()
        nb_neighbors=len(id_neighb)
        for i in range(t):
            while chemins[k,i] in id_neighb:
                id_neighb.remove(chemins[k,i])
        if len(id_neighb)==0:
            chemins[k,t+1]=chemins[k,t]
        else:
            chemins[k,t+1]=np.random.choice(id_neighb,1).astype(int
```

We replace the index by the name of the station to be clearer.

In [ ]:
```
```

In [48]:
```
chemins_name=[]
for k in range(pop):
    chemin_k=[]
    for t in range(T):
        chemin_k.append(Ln[chemins[k,t]])
    chemins_name.append(chemin_k)
```

Results

To interpret the results, it is important to compare the distribution at the initial state and the final one. We first look at the initial situation. We sort the city in which there are the most people at the initial time.

In [49]:
```
init_city=[]
init_city_occ=[]
for k in range(pop):
    if Dv[chemins_name[k][0]] not in init_city:
        conteur=0
        for i in range(k,pop):
            if Dv[chemins_name[k][0]]==Dv[chemins_name[i][0]]:
                conteur+=1
        init_city.append(Dv[chemins_name[k][0]])
        init_city_occ.append(conteur)

init_bilan=[]
for k in range(len(init_city)):
    init_bilan.append([init_city[k],init_city_occ[k]])

sorted(init_bilan, key=lambda x: x[1], reverse=True)[:30]
```

Out[49]:
```
[['VILLEURBANNE', 1012],
 ['LYON 3EME', 697],
 ['LYON 8EME', 591],
 ['LYON 7EME', 589],
 ['VENISSIEUX', 428],
 ['LYON 5EME', 340],
 ['LYON 9EME', 340],
 ['VAULX-EN-VELIN', 331],
 ['LYON 6EME', 312],
 ['CALUIRE-ET-CUIRE', 308],
 ['SAINT-PRIEST', 294],
 ['BRON', 259],
 ['LYON 4EME', 251],
 ['MEYZIEU', 232],
 ['RILLIEUX-LA-PAPE', 199],
 ['LYON 2EME', 193],
 ['OULLINS', 185],
 ['LYON 1ER', 179],
 ['DECINES-CHARPIEU', 171],
 ['TASSIN-LA-DEMI-LUNE', 148],
 ['SAINT-GENIS-LAVAL', 140],
 ['SAINT-FONS', 134],
 ['SAINTE-FOY-LES-LYON', 127],
 ['GIVORS', 126],
 ['FRANCHEVILLE', 123],
 ['ECULLY', 118],
 ['GENAS', 89],
 ['CORBAS', 81],
 ['FEYZIN', 79],
 ['MIONS', 78]]
```

Now, we compare with the distribution at the end of the experiment.

In [50]:
```
final_city=[]
final_city_occ=[]
for k in range(pop):
    if Dv[chemins_name[k][-1]] not in final_city:
        conteur=0
        for i in range(k,pop):
            if Dv[chemins_name[k][-1]]==Dv[chemins_name[i][-1]]:
                conteur+=1
        final_city.append(Dv[chemins_name[k][-1]])
        final_city_occ.append(conteur)

final_bilan=[]
for k in range(len(final_city)):
    final_bilan.append([final_city[k],final_city_occ[k]])

sorted(final_bilan, key=lambda x: x[1], reverse=True)[:30]
```

Out[50]:
```
[['BRON', 714],
 ['LYON 3EME', 616],
 ['VILLEURBANNE', 611],
 ['LYON 6EME', 407],
 ['LYON 4EME', 358],
 ['VENISSIEUX', 309],
 ['LYON 1ER', 306],
 ['LYON 2EME', 303],
 ['LYON 5EME', 298],
 ['LYON 7EME', 277],
 ['VAULX-EN-VELIN', 275],
 ['CALUIRE-ET-CUIRE', 265],
 ['SAINT-FONS', 232],
 ['RILLIEUX-LA-PAPE', 216],
 ['FEYZIN', 211],
 ['MIONS', 205],
 ['MEYZIEU', 203],
 ['SAINT-GENIS-LAVAL', 191],
 ['CORBAS', 188],
 ['LYON 8EME', 168],
 ['DECINES-CHARPIEU', 165],
 ['OULLINS', 159],
 ['ECULLY', 157],
 ['CHASSIEU', 157],
 ['LYON 9EME', 154],
 ['SAINT-PRIEST', 133],
 ['PIERRE-BENITE', 127],
 ['SAINTE-FOY-LES-LYON', 127],
 ['GRIGNY', 125],
 ['FONTAINES-SUR-SAONE', 122]]
```

We see a slightly different ranking. The changes in the ranking concur with our inituition of living places and working places in Lyon. It would be interesting to compare it with actual demographic data but we did not find such information.

Going further

Let's try to go further by choosing the traveling time of each "person" randomly. Assuming that each person does not travel the same amount of time to go from home to work, we use a Gaussian distribution to model this travel time.

In [51]:
```
end_times=5+(np.random.randn(pop)*5+30).astype(int)
for i in range(pop):
    if end_times[i]<0:
        end_times[i]=0

final_city=[]
final_city_occ=[]
for k in range(pop):
    if Dv[chemins_name[k][-end_times[k]]] not in final_city:
        conteur=0
        for i in range(k,pop):
            if Dv[chemins_name[k][-end_times[k]]]==Dv[chemins_name[
                conteur+=1
        final_city.append(Dv[chemins_name[k][-end_times[k]]])
        final_city_occ.append(conteur)

final_bilan=[]
for k in range(len(final_city)):
    final_bilan.append([final_city[k],final_city_occ[k]])

sorted(final_bilan, key=lambda x: x[1], reverse=True)[:30]
```

Out[51]:
```
[['BRON', 745],
 ['LYON 3EME', 619],
 ['VILLEURBANNE', 588],
 ['LYON 6EME', 436],
 ['LYON 4EME', 332],
 ['LYON 2EME', 331],
 ['VENISSIEUX', 312],
 ['LYON 1ER', 307],
 ['LYON 5EME', 305],
 ['VAULX-EN-VELIN', 299],
 ['LYON 7EME', 288],
 ['CALUIRE-ET-CUIRE', 250],
 ['SAINT-FONS', 242],
 ['LYON 8EME', 213],
 ['MEYZIEU', 201],
 ['RILLIEUX-LA-PAPE', 198],
 ['DECINES-CHARPIEU', 191],
 ['ECULLY', 188],
 ['CORBAS', 186],
 ['SAINT-GENIS-LAVAL', 180],
 ['MIONS', 180],
 ['FEYZIN', 175],
 ['LYON 9EME', 172],
 ['SAINT-PRIEST', 168],
 ['SAINTE-FOY-LES-LYON', 141],
 ['CHASSIEU', 138],
 ['OULLINS', 134],
 ['PIERRE-BENITE', 120],
 ['GRIGNY', 117],
 ['GIVORS', 108]]
```

We see little differences in the ranking compare to the simulation without the gaussian travel time. Once again, it could be interesting to see if this analysis fit better the reality with actual data. Besides, we may adapt the parameters of the distribution to see which ones correspond the best to the reality.

Conclusion

In this report we worked on the TCL public transport network with Python. First, we created a simple modelisation of the graph enabling us to get a first impression of the graph, highlighting hubs of the network and performing communities detetection on the graph.

After that, we used our second modelisation in order to better simulate trajectories on the network taking into acount lines changing and giving us the best itinerary.

We finished with a random walk simulation on the graph in order to find the working places.

In order to improve our simulation we could add "Points of interest" in our graph such as Schools, railway station or museum for instance. We found the DATA to do this but we did not have the time to finish the analysis.

To better interpret our results, we could also try to find studies about the working/leaving areas in the 'Grand Lyon" as already mentionned.