# Relationships Analysis in a French High School

The project is based on 5 friendship datasets (available here (http://www.sociopatterns.org/datasets/high-school-contact-and-friendship-networks/)) collected in a French High School for 5 days in December 2013.

The dataset is composed of 4 different graphs and their corresponding dataframes :

- FRIENDSHIP is the graph of related physical interactions between students, based on the *friendship* dataframe
- FACEBOOK is the graph of facebook friendships, based on the *facebook* dataframe. There is a connection between two nodes if their are "friends" on Facebook
- CONTACT is an other graph of physical interactions which classifies the interactions depending on their duration (t < 5min, 5min < t <15min, 15min < t < 1h, t > 1h). It is based on the *contact* dataframe. The graph has been acquired by measuring the proximity (<1.5m) between students, during a day, using sensors.
- *ContactDynamic* is the dataframe relating interactions over 20 seconds timesteps
- *Meta* is the dataframe containing the metadata of each node

After the following preliminary operations we will try to characterize the relationships for the entire graphs. Then using, the dynamic graphs, we will characterize the overall relationships over time, and conclude by studying the dynamic relationships at the single person level.

# I. Preliminary Steps

In [263]:
```
import networkx as nx
import os
import pandas as pd

import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable  #set color
import matplotlib.cm as cm  #used in colorbars
import matplotlib.patches as mpatches  #bar legends
plt.style.use('seaborn-white')

from cdlib import algorithms,viz
import tnetwork as tn
import math
import numpy as np
import scipy
from scipy.stats import gaussian_kde
import collections
```

To import the graphs, you have to download them from here (http://www.sociopatterns.org/datasets/high-school-contact-and-friendship-networks/), store the .csv in the same directory without changing their names :

In [264]:
```
### Load the dataframes
os.chdir(r"/home/clement-hallopeau/Bureau/cours/M2/Complex Networks/

friendship = pd.read_csv("Friendship-network_data_2013.csv",sep=' '
facebook = pd.read_csv("Facebook-known-pairs_data_2013.csv",sep=' '
contact = pd.read_csv("Contact-diaries-network_data_2013.csv",sep='
ContactDynamics = pd.read_csv("High-School_data_2013.csv",sep=' ',na
Meta = pd.read_csv("metadata_2013.txt", sep='\t',names=["i","sectio
```

Then we build the graphs from each csv file. All the files contain columns named "i" and "j" meaning that student i had contacts with student j. The detailed structures of each dataframe is detailed on the website.

In [265]:
```
### Build the graphs
# FRIENDSHIP graph
FRIENDSHIP=nx.DiGraph()

E=list()
for k in range (len(friendship.i)):
    E.append((friendship.i[k],friendship.j[k]))

FRIENDSHIP.add_nodes_from(list(friendship.i))
FRIENDSHIP.add_edges_from(E)


# FACEBOOK graph
FACEBOOK=nx.Graph()

E=list()
V=list()
for k in range (len(facebook.i)):
  if facebook.w[k]==1:    #if there is a relationship
    E.append((facebook.i[k],facebook.j[k]))
    V.append(facebook.i[k])  #it's a directed graph, somes nodes do
    V.append(facebook.j[k])  # in column i but appear in column j

FACEBOOK.add_edges_from(E)
FACEBOOK.add_nodes_from(V)


# CONTACT graph
CONTACT=nx.DiGraph()

E=list()
for k in range (len(contact.i)):
    E.append((contact.i[k],contact.j[k],contact.w[k]))

CONTACT.add_nodes_from(list(contact.i))
CONTACT.add_weighted_edges_from(E)
```

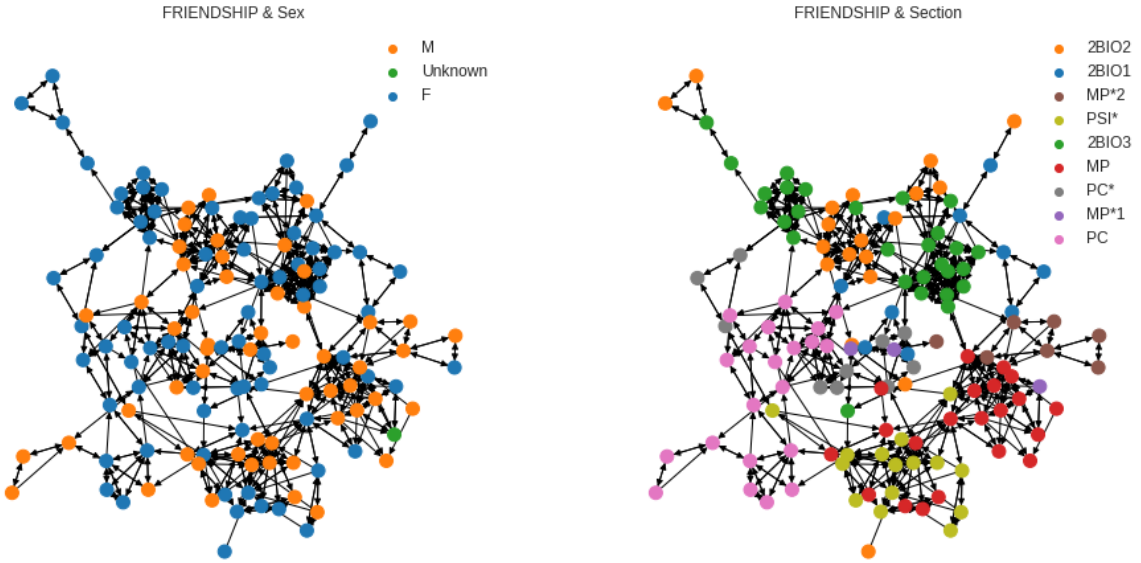
From the metadata file, we add "sex" and "section" attribute to each node of each graph :

In [266]:
```
### We add attributes (section & sex) to each nodes of each graph
for k in list(FACEBOOK.nodes()):
    FACEBOOK.nodes[k]['section']= str(list(Meta.section[Meta.i==k])
    FACEBOOK.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])

for k in list(FRIENDSHIP.nodes()):
    FRIENDSHIP.nodes[k]['section']= str(list(Meta.section[Meta.i==k
    FRIENDSHIP.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])

for k in list(CONTACT.nodes()):
    CONTACT.nodes[k]['section']= str(list(Meta.section[Meta.i==k])[
    CONTACT.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])
```

To finally plot the graphs with node color depending on sex or section :
The *kamada kawai* projections are used since they visually lead to the most sparse representation compared to other projections available in networkx.

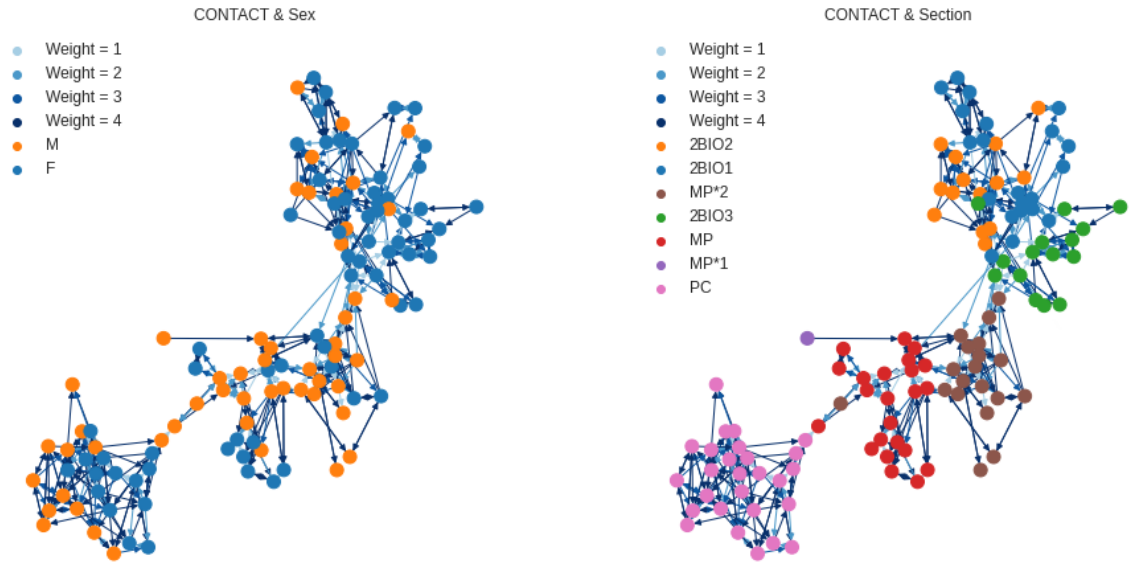
In [267]:
```
nodesize=100
figsize =(16,8)
dpi = 800
leg_prop = {'size' :12}
plt.figure(figsize=figsize)  # FRIENDSHIP & sex
plt.subplot(121)
fig5 = nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP), n
for v in list(set(list(nx.get_node_attributes(FRIENDSHIP,'sex').val
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend(prop = leg_prop)
plt.title('FRIENDSHIP & Sex')

plt.subplot(122) #FRIENDSHIP and section
fig2 = nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP), n
for v in list(set(list(nx.get_node_attributes(FRIENDSHIP,'section')
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend(prop = leg_prop)
plt.title('FRIENDSHIP & Section')
```

Out[267]: Text(0.5, 1.0, 'FRIENDSHIP & Section')



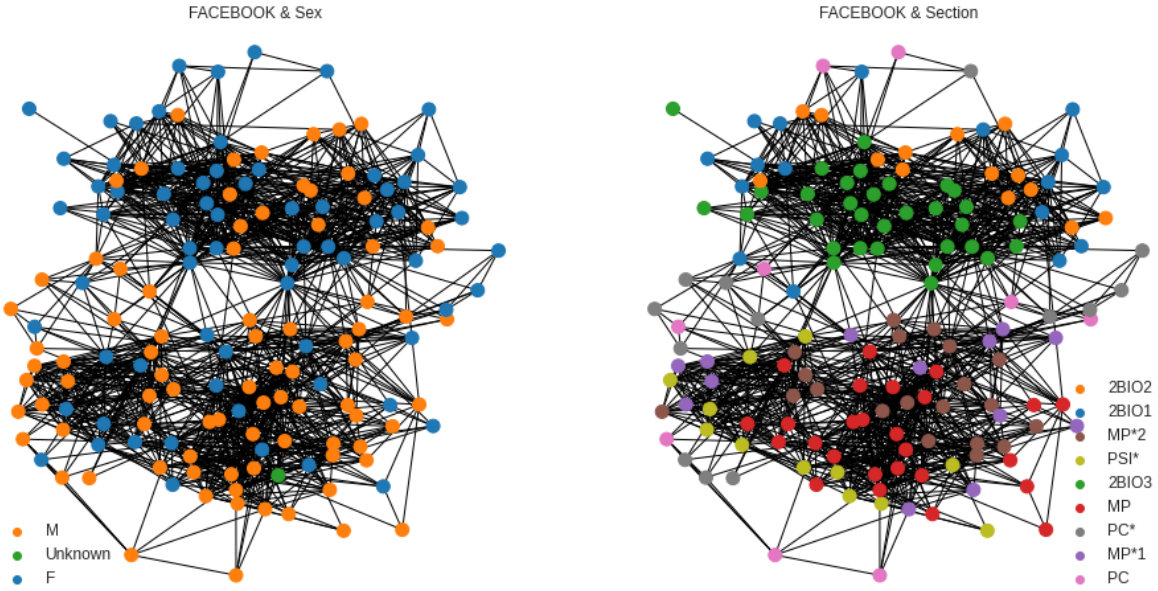
In [268]:
```
nodesize=100
figsize =(16,8)
dpi = 800
leg_prop = {'size' :12}
plt.figure(figsize=figsize)  # CONTACT & sex
plt.subplot(121)
fig6= nx.draw_kamada_kawai(CONTACT,width=1,node_size=nodesize,node_
for v in list(set(contact.w)):
    plt.scatter([],[], color=[plt.cm.Blues(v/max(contact.w)+0.1) fo
plt.legend(prop = leg_prop)
for u in list(set(list(nx.get_node_attributes(CONTACT,'sex').values
    plt.scatter([],[], color=[plt.cm.tab10(u) for u in list(pd.Data
plt.legend(prop = leg_prop)
plt.title('CONTACT & Sex')

plt.subplot(122) #CONTACT and section
fig3 = nx.draw_kamada_kawai(CONTACT,width=1,node_size=nodesize,node_
for v in list(set(contact.w)):
    plt.scatter([],[], color=[plt.cm.Blues(v/max(contact.w)+0.1) fo
for u in list(set(list(nx.get_node_attributes(CONTACT,'section').va
    plt.scatter([],[], color=[plt.cm.tab10(u) for u in list(pd.Data
plt.legend(prop = leg_prop)
plt.title('CONTACT & Section')
```

Out[268]:
```
Text(0.5, 1.0, 'CONTACT & Section')
```



In [269]:
```
nodesize=100
figsize =(16,8)
dpi = 800
leg_prop = {'size' :12}
plt.figure(figsize=figsize)  # FACEBOOK & sex
plt.subplot(121)
fig4 = nx.draw(FACEBOOK, node_size = nodesize, pos=nx.kamada_kawai_
for v in list(set(list(nx.get_node_attributes(FACEBOOK,'sex').value
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend(prop = leg_prop)
plt.title('FACEBOOK & Sex')

plt.subplot(122) # FACEBOOK and section
fig1 = nx.draw(FACEBOOK,pos=nx.kamada_kawai_layout(FACEBOOK), node_
for v in list(set(list(nx.get_node_attributes(FACEBOOK,'section').v
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.title('FACEBOOK & Section')
plt.legend(prop = leg_prop)
```

Out[269]:
```
<matplotlib.legend.Legend at 0x7f6d5ce69dc0>
```




Looking at theses three graphs raises questions of different natures. First, since they don't represent the same relationships (real, online), : **do the graphs share the same structural features**, such as degree. It is clear that these graphs contain communities : **can we find nodes attributes explaining these communities ?** On the FACEBOOK graph, two nodes seem highly connected and seem to bridge the two main communities : **can we find central people within the relationship networks ?**.

# II. Measures on Static Graphs

## 2.1 Can we find popular people ?

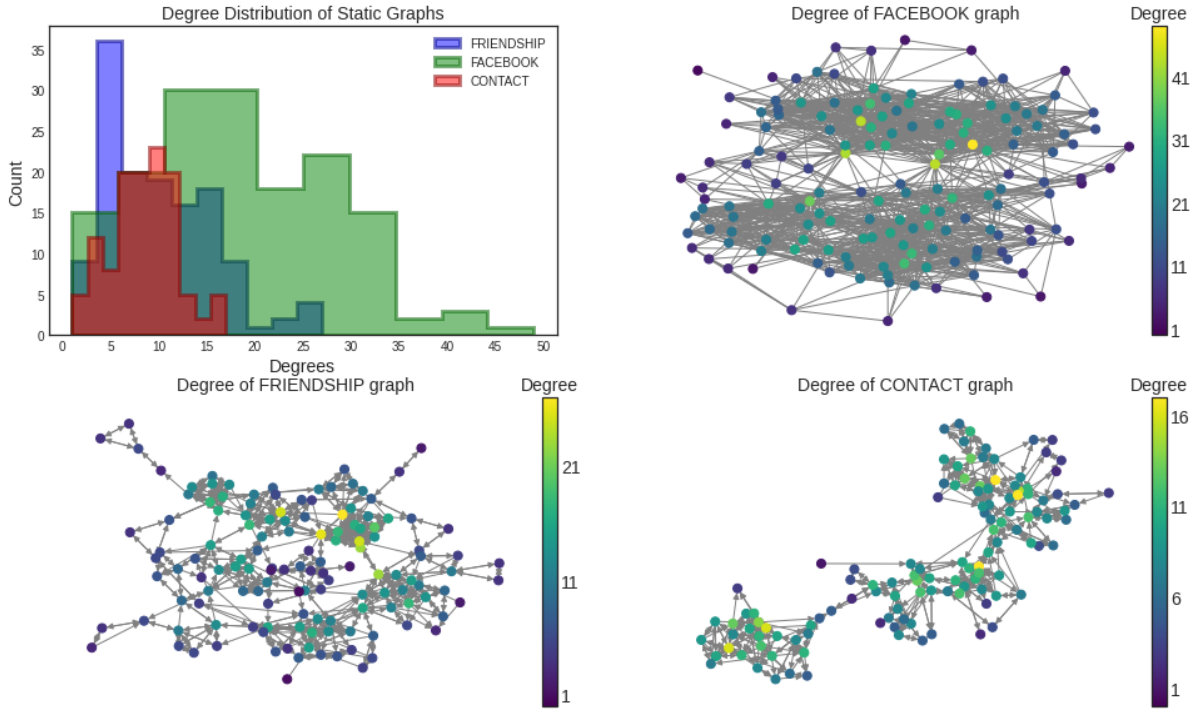
In [270]:
```
nodesize = 50
```

```
fontsize = 14
bins = 10
plt.figure(figsize=(16,10))
plt.subplot(221)
plt.hist(list(dict(FRIENDSHIP.degree()).values()),bins = bins, alph
plt.hist(list(dict(FACEBOOK.degree()).values()), bins = bins,alpha =
plt.hist(list(dict(CONTACT.degree()).values()), bins = bins,alpha =
plt.title('Degree Distribution of Static Graphs', fontsize=fontsize
plt.xlabel('Degrees', fontsize=fontsize)
plt.ylabel('Count', fontsize=fontsize)
plt.xticks(np.arange(0,55,5))
plt.legend()

plt.subplot(222)
b = list(dict(FACEBOOK.degree()).values())
plt.title('Degree of FACEBOOK graph', fontsize=fontsize)
nx.draw(FACEBOOK,pos=nx.kamada_kawai_layout(FACEBOOK),edge_color="g
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i) for i in range(int(min(b)),int(max(b)
cb.ax.set_title('Degree', fontsize = fontsize)

plt.subplot(223)
b = list(dict(FRIENDSHIP.degree()).values())
plt.title('Degree of FRIENDSHIP graph', fontsize=fontsize)
nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP),edge_colo
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i) for i in range(int(min(b)),int(max(b)
cb.ax.set_title('Degree', fontsize = fontsize)

plt.subplot(224)
b = list(dict(CONTACT.degree()).values())
plt.title('Degree of CONTACT graph', fontsize=fontsize)
nx.draw(CONTACT,pos=nx.kamada_kawai_layout(CONTACT),edge_color="gra
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i) for i in range(int(min(b)),int(max(b)
cb.ax.set_title('Degree', fontsize = fontsize)
```

```
<ipython-input-270-ed7b8611ca86>:19: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
3%")
<ipython-input-270-ed7b8611ca86>:28: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
3%")
<ipython-input-270-ed7b8611ca86>:37: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
```

```
3%")
```

Out[270]: Text(0.5, 1.0, 'Degree')




**Degree Distribution**

The degree distribution (histogram) shows that **people make more connections on Facebook** compared to the real life. Indeed, the FACEBOOK degree histogram has a Gaussian shape centered on ~20 people, whereas CONTACT and FRIENDSHIP histograms are centered on ~10 people.

One thing to mention is the tail on the right of the histograms, especially on the FACEBOOK (degree > 35) and CONTACT (degree > 20) histogram, showing that there is a **minority of high degree nodes**.

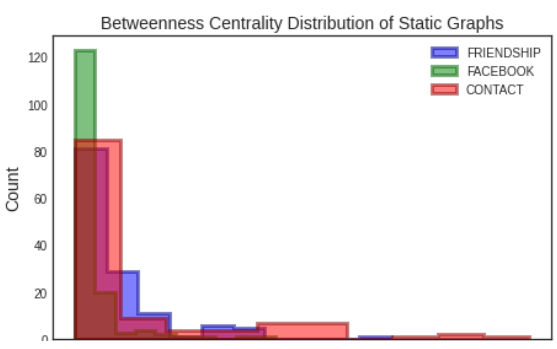
These high degree nodes corresponding to students making a lot of connections are visible on the graphs were a minority of nodes appear to be yellow/light green. Some high-degree nodes are located in the middle of node clusters (*e.g.* bottom left of CONTACT graph, thus they could correspond to **people highly integrated in their community**. The other high-degree nodes are located at the edges of nodes clusters (*e.g.* FACEBOOK graph between the top and bottom clusters, FRIENDSHIP the cluster on the top-right), they could correspond **people involved in different social communities**. Thus, they might appear with the highest betweenness centrality in the following figure.

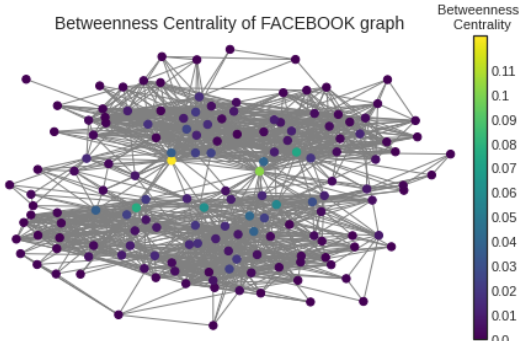
In [271]:
```
nodesize = 40
fontsize = 14
bins = 10
plt.figure(figsize=(16,10))
plt.subplot(221)
plt.hist(list(nx.betweenness_centrality(FRIENDSHIP).values()),bins =
plt.hist(list(nx.betweenness_centrality(FACEBOOK).values()),bins = 
plt.hist(list(nx.betweenness_centrality(CONTACT).values()),bins = b
plt.title('Betweenness Centrality Distribution of Static Graphs', f
plt.xlabel('Betweenness Centrality', fontsize=fontsize)
```

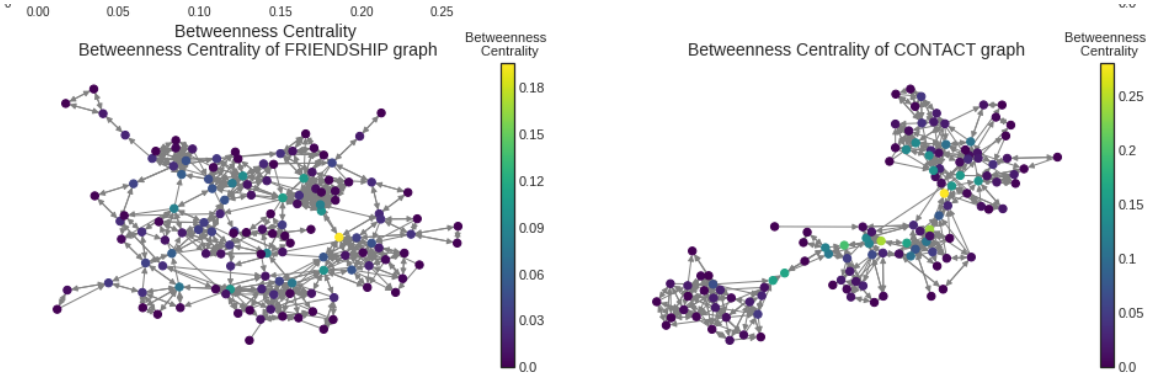
```
plt.ylabel('Count', fontsize=fontsize)
plt.legend()

plt.subplot(222)
b = list(nx.betweenness_centrality(FACEBOOK).values())
plt.title('Betweenness Centrality of FACEBOOK graph', fontsize=font
nx.draw(FACEBOOK,pos=nx.kamada_kawai_layout(FACEBOOK),edge_color="g
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i/100) for i in range(int(min(b)),int(ma
cb.ax.set_title('Betweenness \n Centrality', fontsize = int(fontsiz

plt.subplot(223)
b = list(nx.betweenness_centrality(FRIENDSHIP).values())
plt.title('Betweenness Centrality of FRIENDSHIP graph', fontsize=fo
nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP),edge_colo
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i/100) for i in range(int(min(b)),int(ma
cb.ax.set_title('Betweenness \n Centrality', fontsize = int(fontsiz

plt.subplot(224)
b = list(nx.betweenness_centrality(CONTACT).values())
plt.title('Betweenness Centrality of CONTACT graph', fontsize=fonts
nx.draw(CONTACT,pos=nx.kamada_kawai_layout(CONTACT),edge_color="gra
cax = make_axes_locatable(plt.gca()).append_axes("right", size="3%"
cb = plt.colorbar(cm.ScalarMappable(norm=None, cmap=plt.cm.viridis)
cb.ax.set_yticklabels([str(i/100) for i in range(int(min(b)),int(ma
cb.ax.set_title('Betweenness \n Centrality', fontsize = int(fontsiz
```

```
<ipython-input-271-7e4105f367f4>:18: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
3%")
<ipython-input-271-7e4105f367f4>:27: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
3%")
<ipython-input-271-7e4105f367f4>:36: MatplotlibDeprecationWarning:
In a future version, 'pad' will default to rcParams['figure.subplo
t.wspace'].  Set pad=0 to keep the old behavior.
  cax = make_axes_locatable(plt.gca()).append_axes("right", size="
3%")
```

Out[271]: Text(0.5, 1.0, 'Betweenness \n Centrality')






**Betweenness Centrality**

The **betweenness centrality distribution is different than the degree distribution** since the three graphs have their histogram centered approximatly on the same value, and present a tail on their right were there are no true differences between them considering the small height of the bars and/or the large width of the bins.

As expected, the **nodes with the highest betweenness are located on the borders of the groups**, especially on the FACEBOOK and FRIENDSHIP graph. On the CONTACT graph, the distribution is more surprising since the two blue nodes at the bottom left are the only one to connect two community clearly separated. Surprisingly they don't have the highest betweenness centrality.

In this high school context, these **high betweenness nodes might be involved in associative activities** such as the student office, which would explain the diversity of their connections.

## 2.3 Can We Detect Communities Using Modularity Algorithms ?

A first look at the graphs reveals obvious communities more likely defined by sections : **is the *Louvain Algorithm* efficient to detect communities** on these graphs ?

In [272]:
```
# DIRECTED graphs -> UNDIRECTED graphs
FRIENDSHIPundir=nx.Graph()
FRIENDSHIPundir.add_nodes_from(list(friendship.i))
E=list()
for k in range (len(friendship.i)):
    E.append((friendship.i[k],friendship.j[k]))
FRIENDSHIPundir.add_edges_from(E)

CONTACTundir=nx.Graph()
CONTACTundir.add_nodes_from(list(contact.i))
E=list()
for k in range (len(contact.i)):
    E.append((contact.i[k],contact.j[k]))
CONTACTundir.add_edges_from(E)

for k in list(FRIENDSHIPundir.nodes()):
    FRIENDSHIPundir.nodes[k]['section']= str(list(Meta.section[Meta
    FRIENDSHIPundir.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[

for k in list(CONTACTundir.nodes()):
    CONTACTundir.nodes[k]['section']= str(list(Meta.section[Meta.i==
```

```
    CONTACTundir.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])


plt.figure(figsize=(16,0))
plt.title('Ratio #Detected component/#Sections for differents lambd
plt.axis('off')

#RATIOS
I = np.arange(0.4,2,0.05) #lambda range
II = np.arange(0.4,2,0.1) #xticks range
ylim = [0.0,2]  #ylim
plt.figure(figsize=(16,3))
plt.subplot(131)
plt.plot([I[0],I[-1]], [1,1], color="gray")
for k in I : #friendship : ratio between detected communities & nbr
    plt.scatter([k],[len(set([i[0] for i in list(algorithms.louvain
plt.grid(which='major', axis='both',)
plt.title('FRIENDSHIP')
plt.xlabel("Lambda")
plt.ylabel('Detected Components/Sections')
plt.xticks(II, rotation=45)
plt.ylim(ylim)


plt.subplot(132)
plt.plot([I[0],I[-1]], [1,1], color="gray")
for k in I: #contact : ratio between detected communities & nbr of
    plt.scatter([k],[len(set([i[0] for i in list(algorithms.louvain
plt.grid(which='major', axis='both',)
plt.xlabel("Lambda")
plt.xticks(II, rotation=45)
plt.ylim(ylim)
plt.title('CONTACT')

plt.subplot(133)
plt.plot([I[0],I[-1]], [1,1], color="gray")
for k in I: #contact : ratio between detected communities & nbr of
    plt.scatter([k],[len(set([i[0] for i in list(algorithms.louvain
plt.grid(which='both', axis='both')
plt.xticks(II, rotation=45)
plt.xlabel("Lambda")
plt.ylim(ylim)
plt.title('FACEBOOK')

plt.figure(figsize=(16,0))
plt.title('Graphs with detected communities (Louvain) :', fontsize=
plt.axis('off')

#RESULTING graphs
plt.figure(figsize = (16,5))
nodesize= 20
plt.subplot(131)
l = 1.1  #lambda
nx.set_node_attributes(FRIENDSHIPundir,algorithms.louvain(FRIENDSHI
nx.draw(FRIENDSHIPundir, pos=nx.kamada_kawai_layout(FRIENDSHIPundir
plt.title(str('FRIENDSHIP : lambda = '+str(l)))
```

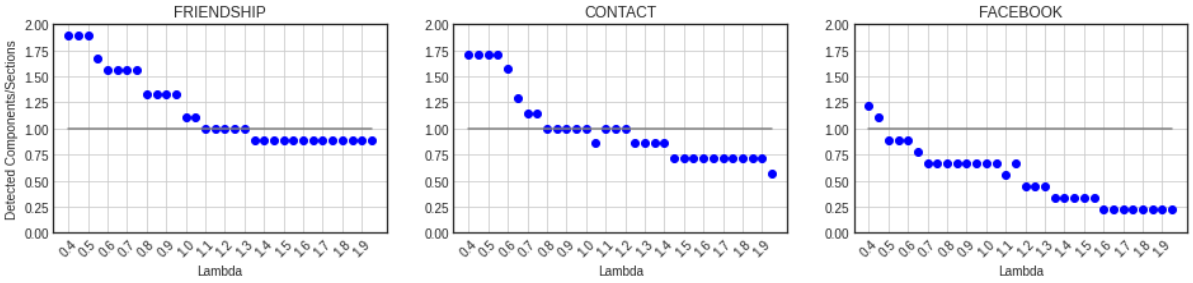
```
plt.subplot(132)
l = 0.8
nx.set_node_attributes(CONTACTundir,algorithms.louvain(CONTACTundir
nx.draw(CONTACTundir, pos=nx.kamada_kawai_layout(CONTACT),node_colo
plt.title(str('CONTACT : lambda = '+str(l)))

plt.subplot(133)
l=0.48
nx.set_node_attributes(FACEBOOK,algorithms.louvain(FACEBOOK,resolut
nx.draw(FACEBOOK, pos=nx.kamada_kawai_layout(FACEBOOK),node_color =
plt.title(str('FACEBOOK : lambda = '+str(l)))

plt.figure(figsize=(16,0))
plt.title('Original graphs :', fontsize=18)
plt.axis("off")

#ORIGINAL graphs
plt.figure(figsize = (16,5))
plt.title('Original Graphs')
leg_prop = {'size' :10}
plt.subplot(131)
fig2 = nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP), n
for v in list(set(list(nx.get_node_attributes(FRIENDSHIP,'section')
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
#plt.legend(prop = leg_prop, loc = 'upper left')
plt.title('FRIENDSHIP & Sections')

plt.subplot(132)
fig3 = nx.draw_kamada_kawai(CONTACT,width=1,node_size=nodesize,node_
for v in list(set(contact.w)):
    plt.scatter([],[], color=[plt.cm.Blues(v/max(contact.w)+0.1) fo
for u in list(set(list(nx.get_node_attributes(CONTACT,'section').va
    plt.scatter([],[], color=[plt.cm.tab10(u) for u in list(pd.Data
#plt.legend(prop = leg_prop, loc = 'upper left')
plt.title('CONTACT & Sections')

plt.subplot(133)
fig1 = nx.draw(FACEBOOK,pos=nx.kamada_kawai_layout(FACEBOOK), node_
for v in list(set(list(nx.get_node_attributes(FACEBOOK,'section').v
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.title('FACEBOOK & Sections')
#plt.legend(prop = leg_prop, loc = 'upper left')
```

Out[272]: Text(0.5, 1.0, 'FACEBOOK & Sections')

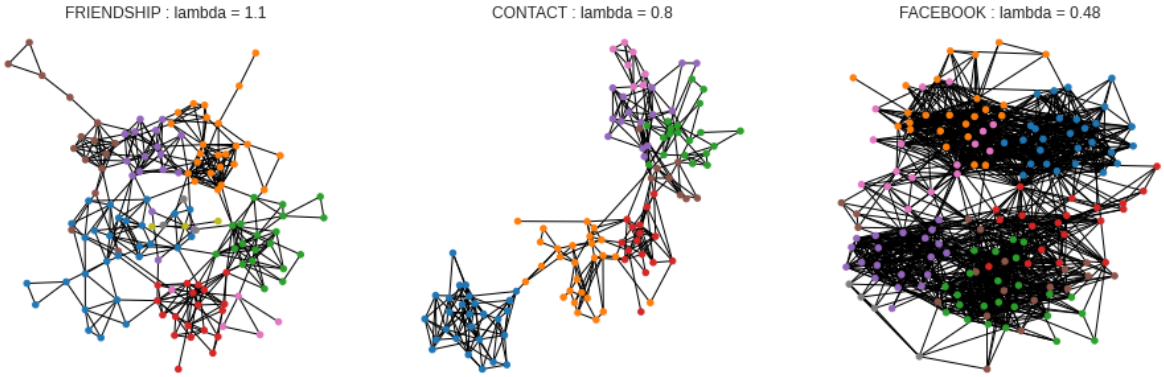
```
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/tight_bbox.py:71: RuntimeWarning: divide by zero encountere
d in double_scalars
  fig.patch.set_bounds(x0 / w1, y0 / h1,
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/tight_bbox.py:72: RuntimeWarning: divide by zero encountere
d in double_scalars
  fig.bbox.width / w1, fig.bbox.height / h1)
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/patches.py:781: RuntimeWarning: invalid value encountered i
n double_scalars
  self._y1 = self._y0 + self._height
```

```
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matplotlib/transforms.py:1978: RuntimeWarning: invalid value encountered in double_scalars
  self._mtx[1, 2] += ty
```

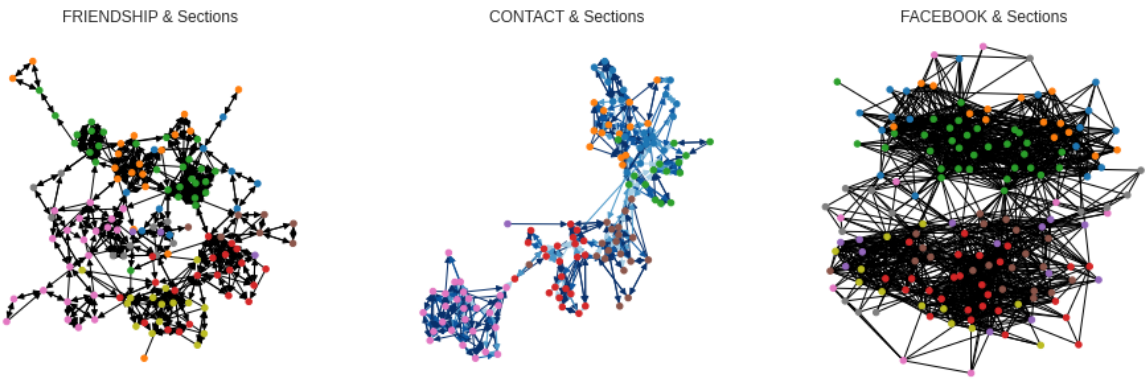
Ratio #Detected component/#Sections for differents lambda :




Graphs with detected communities (Louvain) :




Original graphs :



The Louvain Algorithm required as input the graph of interest and a resolution parameter $\lambda$. To find the correct value of $\lambda$, several detections are ran with different values. For each community, the number of detected communities is divided by the number of expected communities. The number of expected communities is the number of sections since the graph communities seems to be defined by the sections (especially on the CONTACT and FRIENDSHIP graph). The ratios for differents values of $\lambda$ are shown on the first line of the figure. When the ratio is equal to 1, the number of detected communities is equal to the number of sections, and the corresponding graphs are plotted on the second line. The original graphs with node colored by sections are plotted on the third line.

On the CONTACT graph, the Louvain method identifies communities which are almost the same than the section clusters of the original graph, which **suggest that the communities are defined by the sections on the CONTACT graphs.**

On the FRIENDSHIP graph, **the link between the sections and the detected communities is less obvious**. For instance the green population at the top of the original graph is splitted in two by the algorithm. Conversely, the red+yellow population at the bottom of the original graph is merged in a single population. Indeed, if we only consider the node distribution, nothing indicates that theses communities share the same section, thus they must be explained by other attributes.

On the FACEBOOK graph, it is hard to find a trend on both the original graph and graph from Louvain algorithm. Indeed sections don't seem to direct the observed clusters.

For the three graphs, the sex attribute hasn't been tested since visually it don't seems to drive the graphs clusters. Except in the Biology sections (FACEBOOK and FRIENDSHIP graphs) where some clusters of women appear, but it is known from our personnal experiences that there are more women than men in biology sections.

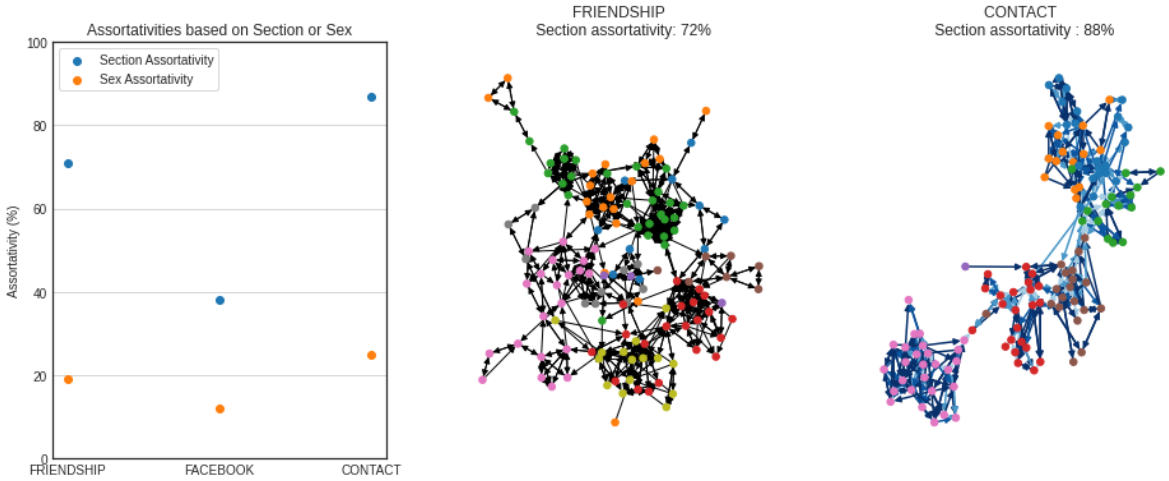
In the next section, assortativity measures will bring a more quantitative overview of section and sex impact on graph clusters.

## 2.2 Using Nodes Attributes to Find Communities

As previously mentionned, some graphs shows distinct clusters of nodes, especially the CONTACT graph and its bottom-left cluster. On the CONTACT and FRIENDSHIP graph, the clusters seems to be defined by the section rather than their sex, which. It was already suggested by the Louvain detection where the choice of $\lambda$ took into account the number of sections. We will see that it is less obvious on the FACEBOOK graph. Assortativity measures will help to have a more quantitative vision.

In [273]:
```
plt.figure(figsize=(16,6))
nodesize = 30
plt.subplot(131)
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_a
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_a
plt.ylim([0,100])
plt.grid(axis = "y")
plt.legend(frameon=True, loc = "upper left")
plt.ylabel('Assortativity (%)')
plt.title('Assortativities based on Section or Sex')

plt.subplot(132) #friendship and section
nx.draw(FRIENDSHIP,pos=nx.kamada_kawai_layout(FRIENDSHIP), node_col
plt.title(str(str('FRIENDSHIP \n Section assortativity: ')+str(roun

plt.subplot(133) # and nodes color depending on section
nx.draw_kamada_kawai(CONTACT,node_size=nodesize,node_color = [plt.c
plt.title(str(str('CONTACT \n Section assortativity : ')+str(round(
```

Out[273]: Text(0.5, 1.0, 'CONTACT \n Section assortativity : 88%')



Indeed, **section is the attribute leading to the highest assortativity** on CONTACT (88%), FRIENDSHIP (72%) and FACEBOOK (38%) graphs. This result can explain and confirm that communities in the real life are mostly defined by sections, which is visually understandable on CONTACT and FRIENDSHIP.

However, **FACEBOOK graph has the lowest section and sex assortativities compaired to the FRIENDSHIP and CONTACT graphs**. Indeed, FACEBOOK graph is based on Facebook connections between people, and it is a way more **easier to ask someone as a friend than talking to him/her in the real life**. As a consequence, the degrees are higher on the FACEBOOK graph (seen previously) and the assortativities are lower since people mix easily compared to the real life.

However, the section assortativities of the real-life interaction graphs are different : **section assortativity of the CONTACT graph is higher than the on the FRIENDSHIP graph**. Furthermore, we know that the *FRIENDSHIP graph is based on the all the friendships reported by the students*, whereas the *CONTACT graph is based on the physical proximities measured during a day*. The most plausible explanation would be that during a day of class, a student doesn't have time to interact with all the people he/her knows. **When the students are in class, they are less mixed with the other sections** explaining the higher section assortativity compared to the FRIENDSHIP graph

The **sex assortativity follows the same trend than the section assortativity** (FRIENDSHIP = 19%, FACEBOOK = 12%, CONTACT = 25%) which could be due to the previous phenomenons : easy mixing on Facebook, less interactions during a day (CONTACT) compared to the absolute relationships (FRIENDSHIP).

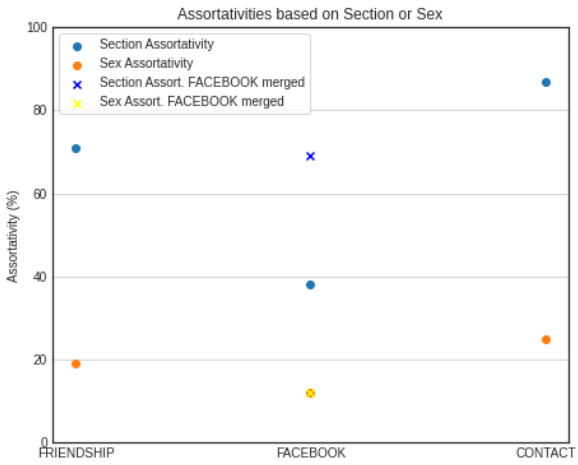
Finally, on the FACEBOOK graph, two main groups appear (*c.f.* previous graph) : at the top there are mainly biologists (2BIO1, 2BIO2, 2BIO3), and at the bottom there are mainly mathematicians (MP*1, MP*2, MP), physicist and chemist (PSI*,PC*, PC). Furthermore, our personal experiences converge to say that in this kind of high school, biologists tend to form a community, whereas "harder" sciences such as mathematics and physics tend to form an other community. Thus, **the same measurement is conducted by merging the node attributes depending on the scientific field** (biologists, mathematicians, physicists/chemists) :

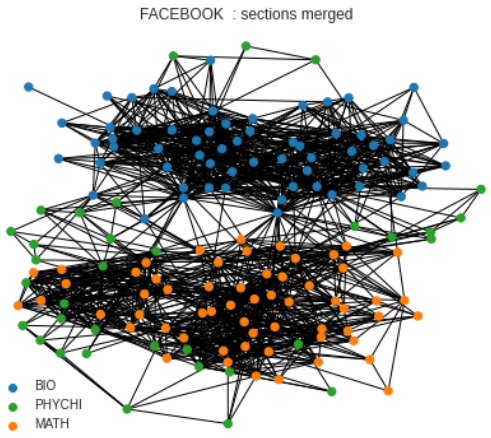
- **BIO** : 2BIO1, 2BIO2, 2BIO3
- **MATH** : MP*1, MP*2, MP
- **PHYCHI** : PSI*, PC*, PC

In [274]:
```
FACEBOOKg=nx.Graph()
E=list()  #edges
V=list()  #vertices
for k in range (len(facebook.i)):
  if facebook.w[k]==1:    #if there is a relationship
    E.append((facebook.i[k],facebook.j[k]))
    V.append(facebook.i[k])  #it's a directed graph, somes nodes do
    V.append(facebook.j[k])  # in column i but appear in column j

FACEBOOKg.add_edges_from(E)
FACEBOOKg.add_nodes_from(V)
```

```
nx.set_node_attributes(FACEBOOKg,float('nan'),"section")
nx.set_node_attributes(FACEBOOKg,float('nan'),"sex")

#Group sections
for k in list(FACEBOOKg.nodes()) :
    if str(list(Meta.section[Meta.i==k])[0]) =='2BIO1' or str(list(M
        FACEBOOKg.nodes[k]['section'] = 'BIO'
    if str(list(Meta.section[Meta.i==k])[0]) =='MP*1' or str(list(M
        FACEBOOKg.nodes[k]['section'] = 'MATH'
    if str(list(Meta.section[Meta.i==k])[0]) =='PC*' or str(list(Me
        FACEBOOKg.nodes[k]['section'] = 'PHYCHI'
    FACEBOOKg.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])

plt.figure(figsize=(16,6))
nodesize=40
plt.subplot(121)
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_a
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_a
plt.scatter(['FACEBOOK'], [int(nx.attribute_assortativity_coefficie
plt.scatter(['FACEBOOK'], [int(nx.attribute_assortativity_coefficie
plt.ylim([0,100])
plt.grid(axis = "y")
plt.legend(loc = "upper left", frameon=True)
plt.ylabel('Assortativity (%)')
plt.title('Assortativities based on Section or Sex')

plt.subplot(122) #friendship and section
nx.draw(FACEBOOKg,pos=nx.kamada_kawai_layout(FACEBOOKg), node_color
plt.title('FACEBOOK  : sections merged')
for v in list(set(list(nx.get_node_attributes(FACEBOOKg,'section').
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend()
```

Out[274]: <matplotlib.legend.Legend at 0x7f6d46a0c730>



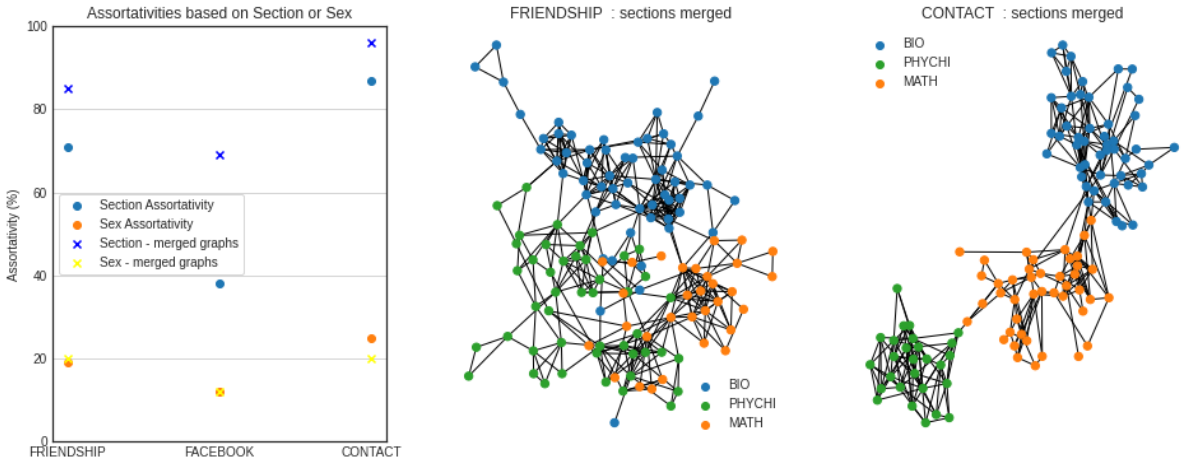
When the nodes attributes are merged depending on the scientific field, the section assortativity of FACEBOOK increase by 1.8 (up to 69%). It indicates that even if people tend to be more mixed on Facebook, the observed **communities are not exactly driven by sections but more by the scientific fied** (Biology, Mathematics, Physics & Chemistry).

The same operation is done the the FRIENDSHIP and CONTACT graph.

In [275]:
```
FRIENDSHIPg=nx.Graph()
E=list()  #edges
for k in range (len(friendship.i)):
    E.append((friendship.i[k],friendship.j[k]))

FRIENDSHIPg.add_nodes_from(list(friendship.i))
FRIENDSHIPg.add_edges_from(E)

nx.set_node_attributes(FRIENDSHIPg,float('nan'),"section")
nx.set_node_attributes(FRIENDSHIPg,float('nan'),"sex")

#Group sections
for k in list(FRIENDSHIPg.nodes()) :
    if str(list(Meta.section[Meta.i==k])[0]) =='2BIO1' or str(list(M
        FRIENDSHIPg.nodes[k]['section'] = 'BIO'
    if str(list(Meta.section[Meta.i==k])[0]) =='MP*1' or str(list(Me
        FRIENDSHIPg.nodes[k]['section'] = 'MATH'
    if str(list(Meta.section[Meta.i==k])[0]) =='PC*' or str(list(Met
        FRIENDSHIPg.nodes[k]['section'] = 'PHYCHI'
    FRIENDSHIPg.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])

CONTACTg=nx.Graph()
E=list()  #edges
for k in range (len(contact.i)):
    E.append((contact.i[k],contact.j[k]))

CONTACTg.add_nodes_from(list(contact.i))
CONTACTg.add_edges_from(E)

nx.set_node_attributes(CONTACTg,float('nan'),"section")
nx.set_node_attributes(CONTACTg,float('nan'),"sex")

#Group sections
for k in list(CONTACTg.nodes()) :
    if str(list(Meta.section[Meta.i==k])[0]) =='2BIO1' or str(list(M
        CONTACTg.nodes[k]['section'] = 'BIO'
    if str(list(Meta.section[Meta.i==k])[0]) =='MP*1' or str(list(Me
        CONTACTg.nodes[k]['section'] = 'MATH'
    if str(list(Meta.section[Meta.i==k])[0]) =='PC*' or str(list(Met
        CONTACTg.nodes[k]['section'] = 'PHYCHI'
    CONTACTg.nodes[k]['sex']= str(list(Meta.sex[Meta.i==k])[0])

plt.figure(figsize=(16,6))
nodesize=40
plt.subplot(131)
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_as
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'],[int(nx.attribute_as
```

```
plt.scatter(['FRIENDSHIP','FACEBOOK','CONTACT'], [int(nx.attribute_
plt.ylim([0,100])
plt.grid(axis = "y")
plt.legend(loc = "center left", frameon=True)
plt.ylabel('Assortativity (%)')
plt.title('Assortativities based on Section or Sex')

plt.subplot(132) #friendship and section
nx.draw(FRIENDSHIPg,pos=nx.kamada_kawai_layout(FRIENDSHIP), node_co
plt.title('FRIENDSHIP  : sections merged')
for v in list(set(list(nx.get_node_attributes(FRIENDSHIPg,'section'
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend()

plt.subplot(133)
nx.draw(CONTACTg,pos=nx.kamada_kawai_layout(CONTACT), node_color =
plt.title('CONTACT  : sections merged')
for v in list(set(list(nx.get_node_attributes(CONTACTg,'section').v
    plt.scatter([],[], color=[plt.cm.tab10(v) for v in list(pd.Data
plt.legend()
```

Out[275]: <matplotlib.legend.Legend at 0x7f6d20ee7fd0>




Merging the nodes attributes depending on their **scientific field increase the section assortativity** by 1.18 on the FRIENDSHIP graph (85%), and 1.09 on the CONTACT graph (96%). In the same way than the FACEBOOK graph, scientific fields drives the communities of the real interaction graphs. But since the increase of assortativity is less important compared to the FACEBOOK graph (1.8), **sections remain the main parameter driving graphs communities on real-interaction graphs**.

Whatever the graph attributes (merged or not), the sex assortativity remains approximatly the same and on the three graphs it is lower than 30%, meaning that there is an **easy mixing between men and women**. But these measure are made at the high school level : **are there differences of sex assortativites at the single-section level ?**

## 2.3 Single Section Level : Can We Distinguish Different Behaviors ?

The previous graph-levels studies emphasizes different social behaviors between the real life and the social network life. The following section investigate whether there are different social behaviors between the sections, and between real and online lifes.

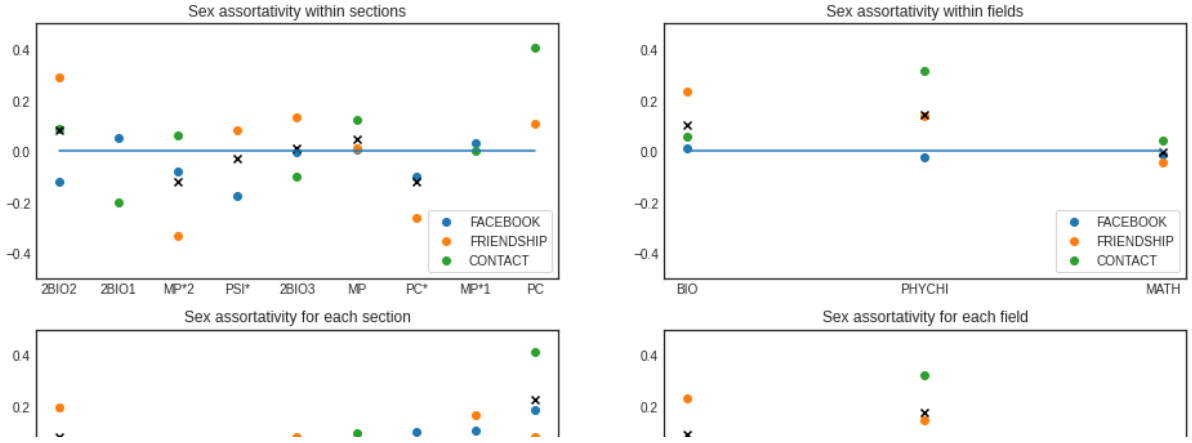
### 2.3.1 Is the People' Sex Driving Social Interactions ?

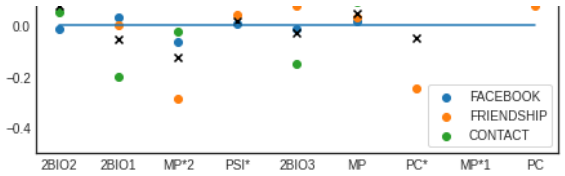
In [276]:
```
I = [-0.5,0.5]
plt.figure(figsize=(16,8))
plt.subplot(221) #within sections
plt.title('Sex assortativity within sections')
X = list(set(list(nx.get_node_attributes(FACEBOOK,'section').values
plt.scatter(X,[nx.attribute_assortativity_coefficient(FACEBOOK.subg

X = list(set(list(nx.get_node_attributes(FRIENDSHIP,'section').valu
plt.scatter(X,[nx.attribute_assortativity_coefficient(FRIENDSHIP.su

X = list(set(list(nx.get_node_attributes(CONTACT,'section').values(
plt.scatter(X,[nx.attribute_assortativity_coefficient(CONTACT.subgr

plt.plot([X[0],X[-1]],[0,0])
plt.legend(loc="lower right", frameon = True)
plt.ylim(I)
#mean
X = list(set(list(nx.get_node_attributes(FACEBOOK,'section').values
for x in X :
    plt.scatter(x,[sum([nx.attribute_assortativity_coefficient(FRIE



plt.subplot(222) #within fields
plt.title('Sex assortativity within fields')
X = list(set(list(nx.get_node_attributes(FACEBOOKg,'section').value
plt.scatter(X,[nx.attribute_assortativity_coefficient(FACEBOOKg.sub

X = list(set(list(nx.get_node_attributes(FRIENDSHIPg,'section').val
plt.scatter(X,[nx.attribute_assortativity_coefficient(FRIENDSHIPg.s

X = list(set(list(nx.get_node_attributes(CONTACTg,'section').values
plt.scatter(X,[nx.attribute_assortativity_coefficient(CONTACTg.subg
plt.plot([X[0],X[-1]],[0,0])
plt.legend(loc="lower right", frameon = True)
plt.ylim(I)
#mean
X = list(set(list(nx.get_node_attributes(FACEBOOKg,'section').value
for x in X :
    plt.scatter(x,[sum([nx.attribute_assortativity_coefficient(FRIE



#sex assortativity for a given section (not IN a given section):
plt.subplot(223)
```

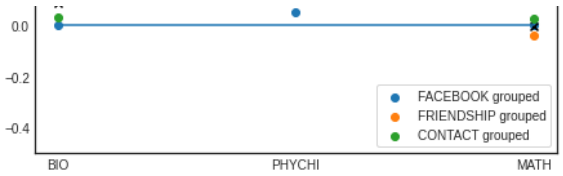
```
plt.title('Sex assortativity for each section')
X = list(set(list(nx.get_node_attributes(FACEBOOK,'section').values
plt.scatter(X,[nx.attribute_assortativity_coefficient(FACEBOOK,'sex

X = list(set(list(nx.get_node_attributes(FRIENDSHIP,'section').valu
plt.scatter(X,[nx.attribute_assortativity_coefficient(FRIENDSHIP,'s

X = list(set(list(nx.get_node_attributes(CONTACT,'section').values(
plt.scatter(X,[nx.attribute_assortativity_coefficient(CONTACT,'sex'

plt.legend(loc="lower right", frameon = True)
plt.plot([X[0],X[-1]],[0,0])
plt.ylim(I)
#mean
X = list(set(list(nx.get_node_attributes(FACEBOOK,'section').values
plt.scatter(X,[sum([nx.attribute_assortativity_coefficient(CONTACT,


plt.subplot(224) #grouped scientific fields
plt.title('Sex assortativity for each field')
X = list(set(list(nx.get_node_attributes(FACEBOOKg,'section').value
plt.scatter(X,[nx.attribute_assortativity_coefficient(FACEBOOKg,'se

X = list(set(list(nx.get_node_attributes(FRIENDSHIPg,'section').val
plt.scatter(X,[nx.attribute_assortativity_coefficient(FRIENDSHIPg,'

X = list(set(list(nx.get_node_attributes(CONTACTg,'section').values
plt.scatter(X,[nx.attribute_assortativity_coefficient(CONTACTg,'sex
plt.legend(loc="lower right", frameon = True)
plt.plot([X[0],X[-1]],[0,0])
plt.ylim(I)
X = list(set(list(nx.get_node_attributes(FACEBOOKg,'section').value
#for x in X :
    #plt.scatter(x,[sum([nx.attribute_assortativity_coefficient(FR
plt.scatter(X,[sum([nx.attribute_assortativity_coefficient(CONTACTg
```

```
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/netw
orkx/algorithms/assortativity/correlation.py:263: RuntimeWarning:
invalid value encountered in double_scalars
  r = (t - s) / (1 - s)
```

Out[276]: <matplotlib.collections.PathCollection at 0x7f6d462a59a0>






To understand the mixing between genres within the classes, the sex assortativity is computed within each section or each scientific field (*i.e.* on the sections subgraphs of each graph), represented on the first line.

The second line shows the same assortativity measures for each section and scientific field, but it integrates the interaction of each node with other sections.

While looking a each section and their means (black cross), there are no clear trends : each section is more or less near zero. However, when the scientific fields are grouped, whatever we consider the assortativity within a class or not, the mathematicians appear to have a sex assortativity near zero meaning their relationships do not depend on the genre. For the **biologists and physicists, their relationships are slightly assortative meaning that women tend to socialize with women, men with men**.

### 2.4.2 Are There Different Social Behaviors Between the Sections ?

In [277]:
```
xlim = [-1,30]
ylim = [0,0.65]
figsize = (16,2)
plt.figure(figsize=(16,0))
plt.title('FACEBOOK :')
plt.axis('off')

plt.figure(figsize = figsize)
plt.subplot(131)
for x in ['2BIO1','2BIO2','2BIO3'] :
    subgraph = FACEBOOK.subgraph([k for k in list(FACEBOOK.nodes())
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(132)
for x in ['MP*1','MP*2','MP'] :
    subgraph = FACEBOOK.subgraph([k for k in list(FACEBOOK.nodes())
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
```

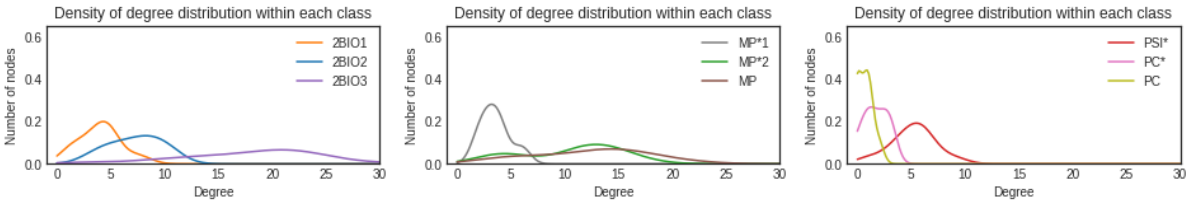
```
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(133)
for x in ['PSI*','PC*','PC'] :
    subgraph = FACEBOOK.subgraph([k for k in list(FACEBOOK.nodes())
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.figure(figsize=(16,0))
plt.title('FRIENDSHIP :')
plt.axis('off')

plt.figure(figsize = figsize)
plt.subplot(131)
for x in ['2BIO1','2BIO2','2BIO3'] :
    subgraph = FRIENDSHIPundir.subgraph([k for k in list(FRIENDSHIP
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(132)
for x in ['MP*1','MP*2','MP'] :
    subgraph = FRIENDSHIPundir.subgraph([k for k in list(FRIENDSHIP
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(133)
for x in ['PSI*','PC*','PC'] :
    subgraph = FRIENDSHIPundir.subgraph([k for k in list(FRIENDSHIP
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
```

```
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.figure(figsize=(16,0))
plt.title('CONTACT :')
plt.axis('off')


plt.figure(figsize = figsize)
plt.subplot(131)
for x in ['2BIO1','2BIO2','2BIO3'] :
    subgraph = CONTACTundir.subgraph([k for k in list(CONTACTundir.
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(132)
for x in ['MP*2','MP'] :
    subgraph = CONTACTundir.subgraph([k for k in list(CONTACTundir.
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)

plt.subplot(133)
for x in ['PC'] :
    subgraph = CONTACTundir.subgraph([k for k in list(CONTACTundir.
    data = list(dict(subgraph.degree()).values())
    density = gaussian_kde(data)
    xs = np.linspace(0,30,200)
    plt.plot(xs,density(xs), color=plt.cm.tab10(list(set(Meta.secti
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('Density of degree distribution within each class')
plt.legend()
plt.ylim(ylim)
plt.xlim(xlim)
```

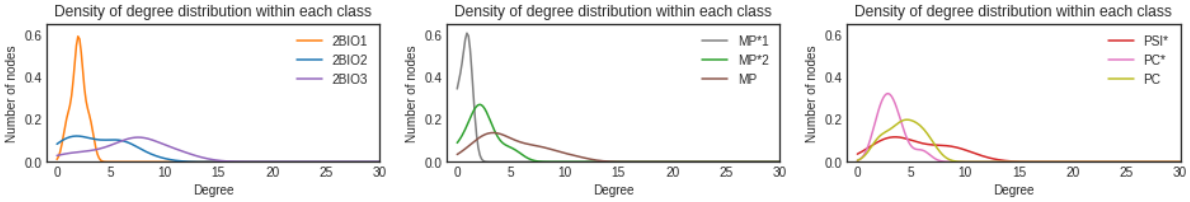
Out[277]: (-1.0, 30.0)

```
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/tight_bbox.py:71: RuntimeWarning: divide by zero encountere
d in double_scalars
  fig.patch.set_bounds(x0 / w1, y0 / h1,
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/tight_bbox.py:72: RuntimeWarning: divide by zero encountere
d in double_scalars
  fig.bbox.width / w1, fig.bbox.height / h1)
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/patches.py:781: RuntimeWarning: invalid value encountered i
n double_scalars
  self._y1 = self._y0 + self._height
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/matp
lotlib/transforms.py:1978: RuntimeWarning: invalid value encounter
ed in double_scalars
  self._mtx[1, 2] += ty
```

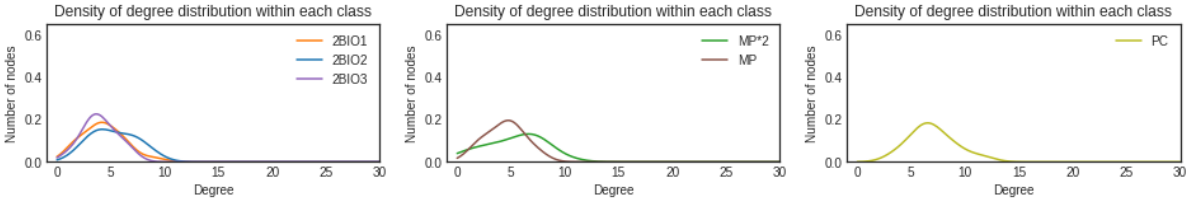
FACEBOOK :




FRIENDSHIP :




CONTACT :



Above of the density of degree for each class, each line corresponding to a graph. As expected, the densities of most sections decreases between the FACEBOOK graph and the CONTACT/FRIENDSHIP graphs. Except for the physicist/chemist whose degree even tend to increase on the real-life interaction graph (PC)

To conclude this section, these static graph allow to draw some features of this social network from a high school.

- This network form communities mainly based on section and scientific field.
- These communities are structured around high degree nodes, and surrounded by high betweenness centrality nodes
- The FACEBOOK graph shows different features on most of the analysis, compared to the real life interaction graphs (CONTACT and FRIENDSHIP) showing how different is the virtual life : people makes more contact, the communities are partly dissolved

The following sections integrates the time dimension on the previous high school network in order to gain accuracy in the description of this social network

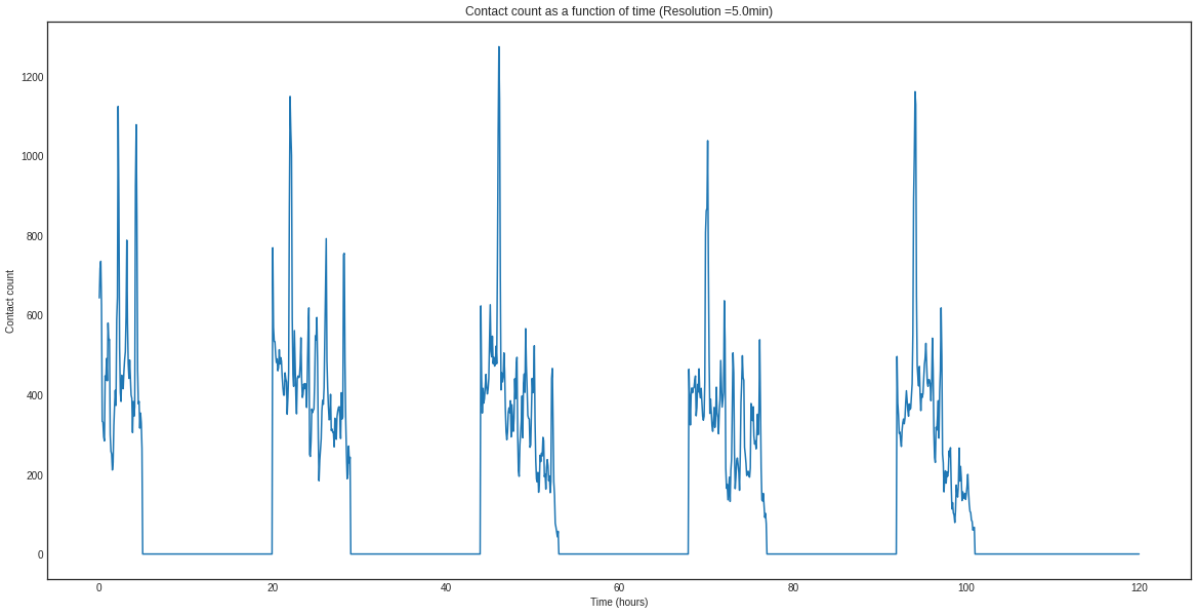
# III. Time evolution of the network

In this section we will investigate the evolution of the network properties over time using the contact dynamics dataset. The latter contains a tab-separated list representing the active contacts during 20-second intervals of the data collection. For the following general analysis we count a contact an edge between two students over 20 seconds. In short, each line of the contact dynamics file yiels a contact. Since 20 seconds is a very short time laps, we agregate the contact count over 5 minutes to track the evolutoin of the network and we obtain the graph below.

## 3.1 Analysis of the periodic properties of the network

In [278]:
```
ContactDynamics=pd.read_csv("High-School_data_2013.csv",sep=' ',nam
ContactDynamics.t = ContactDynamics.t - ContactDynamics.t[0]



Meta=pd.read_csv("metadata_2013.txt", sep='\t',names=["i","section"


def ContactCount(t_start, t_stop, df = ContactDynamics):
    masque = (df["t"] >= t_start) & (df["t"] < t_stop)
    filtered_df = df[masque]

    return len(filtered_df)

def TimeEvol(t_start, step, Nb_step, df=ContactDynamics, plot=True)
    t = np.arange(t_start, t_start + step*Nb_step, step)
    y = np.zeros(Nb_step)
    for k in range(Nb_step):
        y[k] = (ContactCount(t_start + k*step, t_start + (k+1)*step

    if plot :
        plt.figure(figsize=(20,10))
        plt.title("Contact count as a function of time (Resolution =
        plt.ylabel("Contact count")
        plt.xlabel("Time (hours)")
        plt.plot(t/3600,y)
        plt.show()

    return t, y
```

In [279]:
```
TimeEvol(0,300, 24*12*5) ;
```



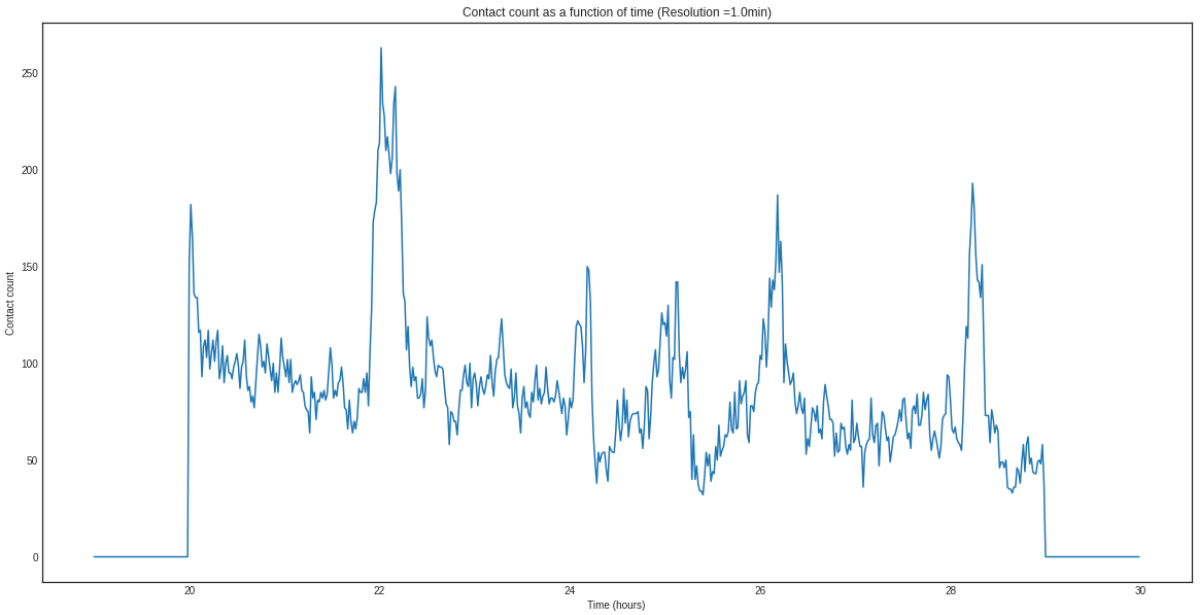
The graph above represents the temporal evolution of the number of contacts during 5 days. Each point corresponds to the number of contacts cumulated over 5 min, i.e. the number of interactions recorded between t and t+5min.

The day is defined in the following as a 24-hour period made up of the day (period of high activity) and the night (period of zero activity). The first thing we observe is that there is a periodicity of 24h corresponding naturally to the duration of a day. The latter comprises two phases, a period of intense activity probably corresponding to a day, followed by a period of zero activity certainly corresponding to the night. By analyzing the duration of a day, one realizes that it invariably lasts 9.1 hours over the 5 days of collection, which leaves the night 14.9 hours. This is surprising for two reasons. First, while it is logical that all students wake up at approximately the same time, one would expect that the last student to send a message would not do so at exactly the same time each day, thus changing the length of the day. On Fridays, for example, it is possible to go to bed later since there is no class the next day. Secondly, it is surprising that the night period lasts so long, a prep student doesn't sleep 15 hours a night!

A reasonable assumption is that data collection is not continuous over the 5 days but starts and stops at fixed times. This would be consistent with 9-hour days of attendance at the high school divided into 4 hours of class in the morning, a lunch break and 4 hours of class in the afternoon. It would also explain why the variations in the number of contacts are so abrupt at the beginning and end of the day.

Let's now analyze one day in more detail. In everything that follows we will keep the same reference day, the one that starts at t = 8 pm.

In [280]:
```
TimeEvol(19*3600, 60, 11*60) ;
```



On the graph above representing the first full day of classes, there are peaks of activity that are clearly above average. There are six at time t = 20, 22, 24, 25, 26 and 28 hours approximately. Considering all the peaks except the one at t = 25h, we notice that there is a periodicity of 2 hours. This corresponds to the periodicity of the courses which are given in 2-hour intervals. In addition, the 10h break (at t = 22h) is a privileged discussion period. The isolated peak at t = 25 could correspond to the lunch break, which lasts only one hour.

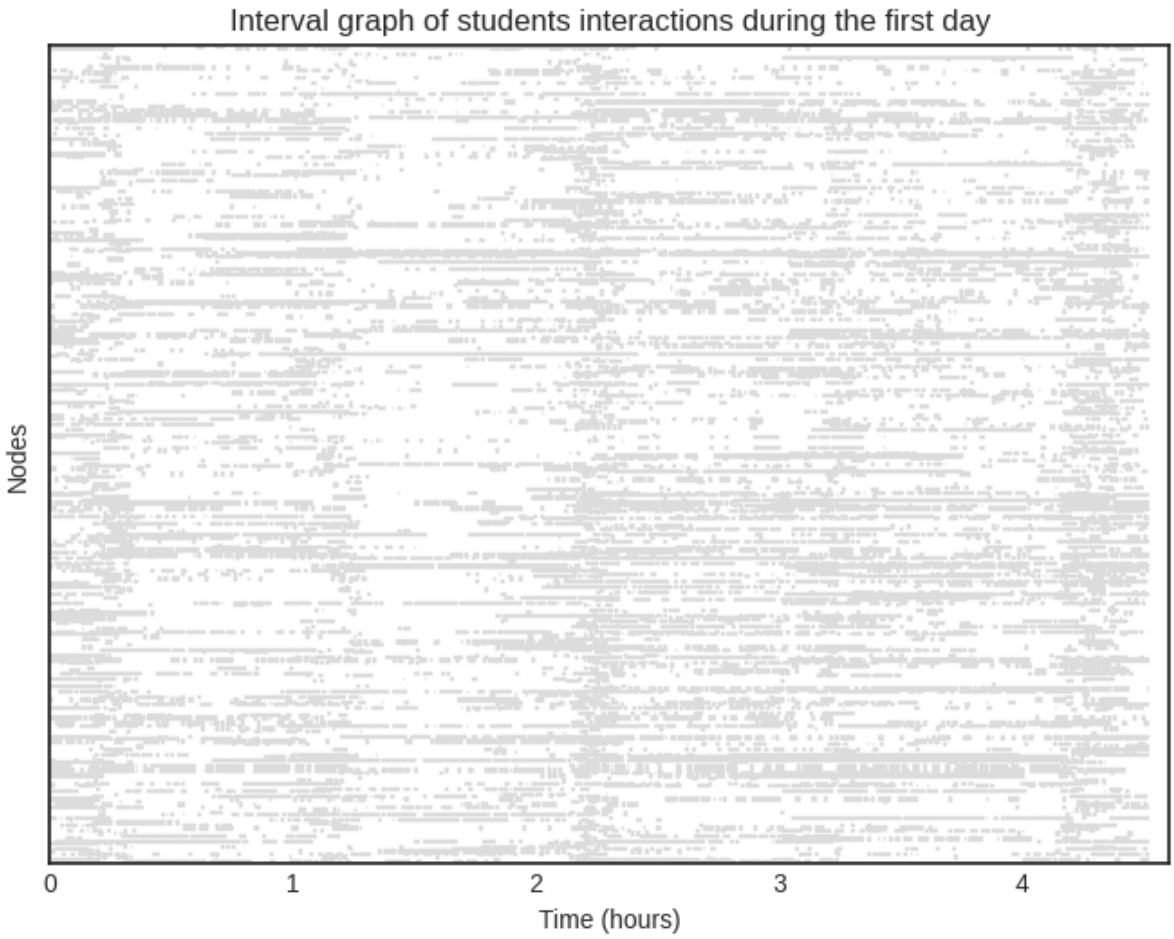
This 2-hour period, corresponding to class hours, is found on the other days as well as the more important peak of the 10h break. On the other hand, there is a variability from one day to another. The peaks do not have the same intensity and their width evolves as well.

These peaks of activity can also be seen very well on the dynamic graph. Let's look at the one of the day starting at t=20h over a period of 5 hours.

In [281]:
```
def Graph(t_start, t_stop, df=ContactDynamics):
    DynamicContact = tn.DynGraphIG()

    masque = (df["t"] >= t_start) & (df["t"] < t_stop)
    filtered_df = df[masque]

    for c in range(len(filtered_df)):
        DynamicContact.add_interaction(str(ContactDynamics.i[c]), s

    return DynamicContact
```

In [282]:
```
g = Graph(19.5*3600, 19.5*3600 + 5*3600);
tn.plot_longitudinal(g);
plt.xlabel("Time (hours)");
plt.ylabel("Nodes");
plt.title("Interval graph of students interactions during the first
```

```
/home/clement-hallopeau/anaconda3/lib/python3.8/site-packages/nump
y/core/numeric.py:2378: FutureWarning: elementwise comparison fail
ed; returning scalar instead, but in the future will perform eleme
ntwise comparison
  return bool(asarray(a1 == a2).all())
```



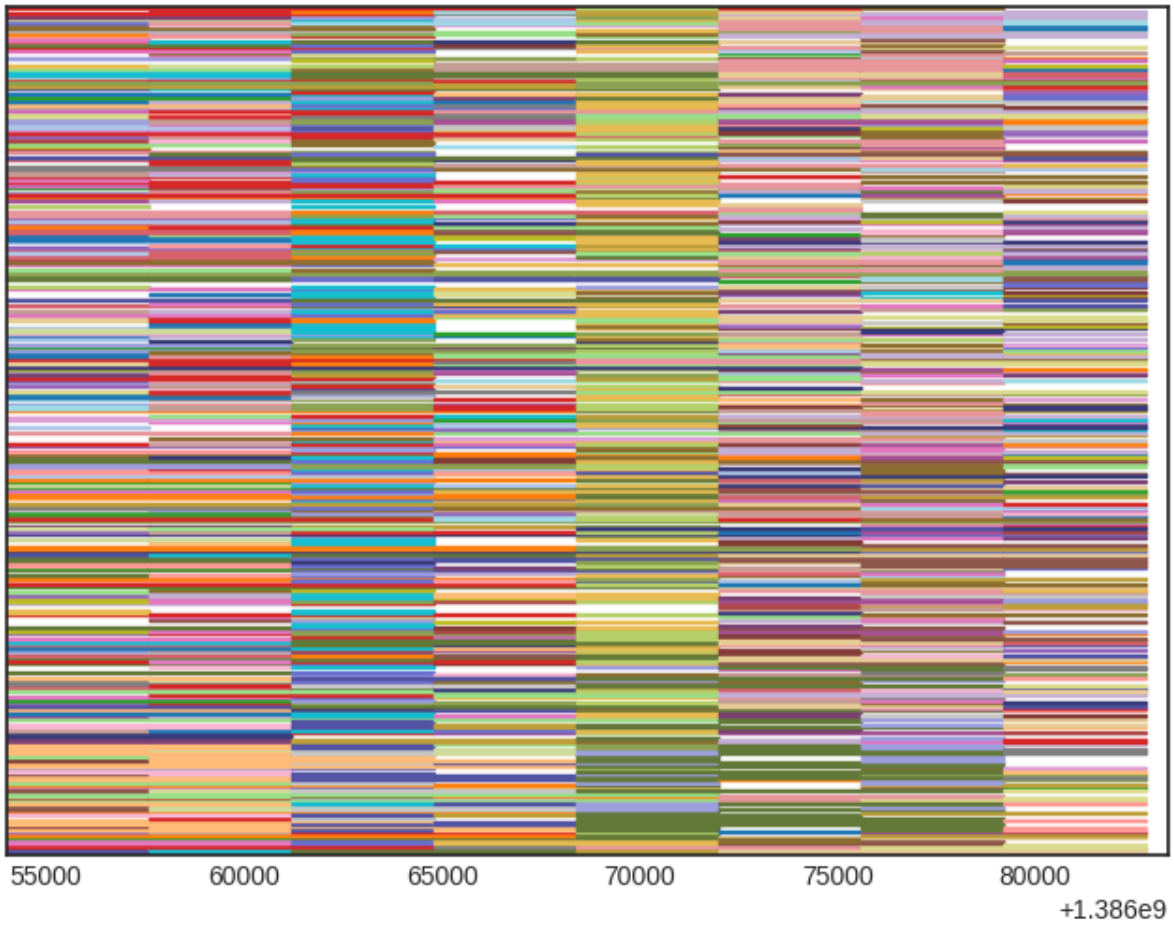
Here the zero corresponds to t = 19.5h. We see that there is a first peak just after. We find the main peak of each day around t = 2.3h. Then we observe another peak for t = 4.3h. We can see here these peaks where a maximum of nodes are active and we find the periodicity of 2h. Also this type of representation allows us to highlight several types of student behavior. Some are active almost continuously, we observe long horizontal bands in the graph. Others are very active but intermittently, we observe horizontal dotted lines.

If the peaks in interaction correspond to breaks between classes, it is surprising to observe so much contact during class time. Hundreds of them occur every 5 minutes. A reasonable hypothesis is that not all classes run at the same time. While most are studying, some students who do not have classes may continue to communicate. To try to see this it would be interesting to look at the dynamics of the communities on this graph to see how they evolve and understand the reasons for the drop in activity between peaks. For example, can you see a class disappearing because it is in class? The graph below represents the community dynamics during the day aggregated by one-hour periods.

## 3.2 Dynamic community detection

In [283]:
```
g = tn.read_interactions(file=r'High-School_data_2013.csv', frequen
t_start = g.change_times()[0]
gslice = g.slice(t_start + 20000, t_start + 100000)
g2 =gslice.aggregate_time_period("hour")

com = tn.iterative_match(g2)
tn.plot_longitudinal(g2, com);
```

```
graph will be loaded as:  <class 'tnetwork.dyn_graph.dyn_graph_sn.
DynGraphSN'>

N/A% (0 of 8) |                          | Elapsed Time: 0:00:00 E
TA:  --:--:--

starting no_smoothing

 75% (6 of 8) |###################       | Elapsed Time: 0:00:00 E
TA:   0:00:00/home/clement-hallopeau/anaconda3/lib/python3.8/site-
packages/numpy/core/numeric.py:2378: FutureWarning: elementwise co
mparison failed; returning scalar instead, but in the future will
perform elementwise comparison
  return bool(asarray(a1 == a2).all())
```



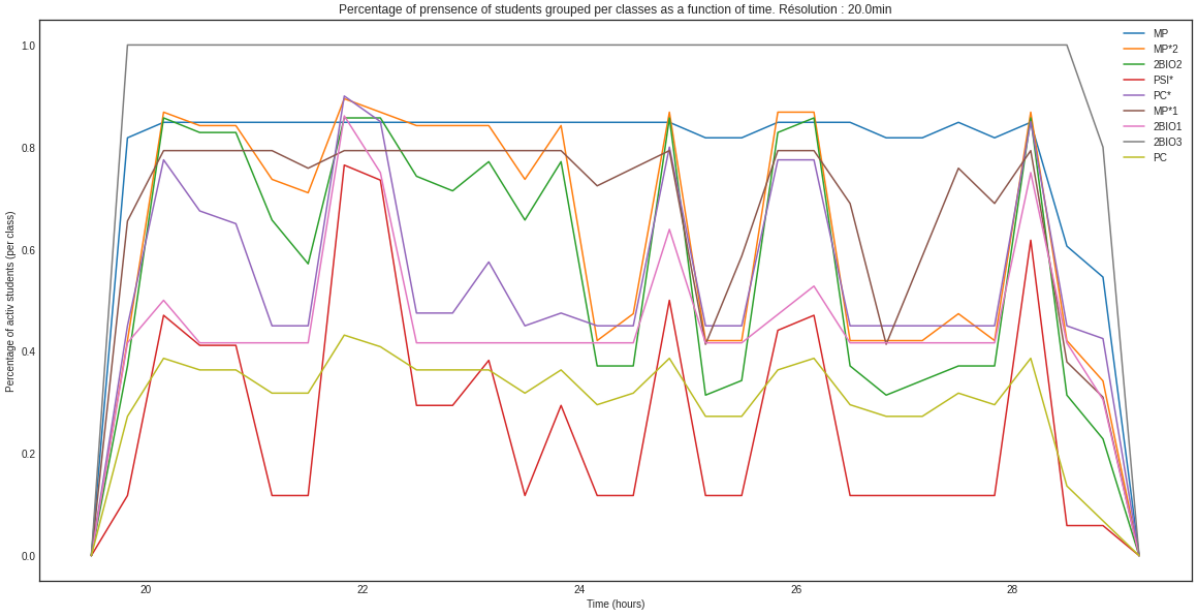
Hard to say much... Detecting communities on graphs is a difficult problem. It is obviously even more difficult on dynamic graphs that are constantly evolving over time, with here a large number of different classes that are not clearly separated and merge into each other.

Another approach may be to look at the percentage of attendance of students in each class over time. In this way it is possible to know which class has class and which does not.

In [284]:
```
def Cumulated(t_start, t_stop, df=ContactDynamics):
    DynamicContact = tn.DynGraphIG()

    masque = (df["t"] >= t_start) & (df["t"] < t_stop)
    filtered_df = df[masque]

    for c in range(len(filtered_df)):
        DynamicContact.add_interaction(ContactDynamics.i[c], Contac


    return DynamicContact.cumulated_graph()

def Presence(t_start, step, Nb_step, df=ContactDynamics, Meta=Meta)

    MP = []
    MP2 = []
    BIO2 = []
    PSI = []
    PCstar = []
    MP1 = []
    BIO1 = []
    BIO3 = []
    PC = []
    t = np.arange(t_start, t_start + step*Nb_step, step)/3600

    classes = list(set(list(Meta["section"]))) #Liste des différent


    for h in range(Nb_step):


        g = Cumulated(h*step + t_start, (h+1)*step + t_start)
        L = list(g.nodes())     #Liste des noeuds présents dans le

        EffectifsTotaux = dict() #Création d'un dictionnaire avec l

        for x in classes :
            EffectifsTotaux[x] = len(Meta[Meta["section"] == x])

        newdf = Meta[Meta.i.isin(L)] #liste des élèves présents dan

        effectifs = collections.Counter(list(newdf["section"]))

        Pourcentage = dict()

        for x in classes :
```

```
            Pourcentage[x] = effectifs[x]/EffectifsTotaux[x]

        MP.append(Pourcentage['MP'])
        MP2.append(Pourcentage['MP*2'])
        BIO2.append(Pourcentage['2BIO2'])
        PSI.append(Pourcentage['PSI*'])
        PCstar.append(Pourcentage['PC*'])
        MP1.append(Pourcentage['MP*1'])
        BIO1.append(Pourcentage['2BIO1'])
        BIO3.append(Pourcentage['2BIO3'])
        PC.append(Pourcentage['PC'])

    plt.figure(figsize=(20,10))
    plt.plot(t, MP, label='MP')
    plt.plot(t, MP2, label='MP*2')
    plt.plot(t, BIO2, label='2BIO2')
    plt.plot(t, PSI, label='PSI*')
    plt.plot(t, PCstar, label='PC*')
    plt.plot(t, MP1, label='MP*1')
    plt.plot(t, BIO1, label='2BIO1')
    plt.plot(t, BIO3, label='2BIO3')
    plt.plot(t, PC, label='PC')

    plt.xlabel("Time (hours)")
    plt.ylabel("Percentage of activ students (per class)")
    plt.title("Percentage of prensence of students grouped per clas
    plt.legend()
```

In [285]:
```
Presence(19.5*3600, 1200, 30)
```



We notice that the MP and MP*1 classes have a constant activity during the day which is 80% and 100% respectively. This means for the MP*1s that all students had at least one contact every 20 minutes.

The other classes behaved differently with periods of less activity and more activity corresponding to breaks between classes. PSI* is a typical example where there is a 30 to 60 percent difference between rest and activity periods. However, we do not observe the total absence of certain classes that could explain the drop in activity during classes. They all modulate their activity in the same way. The participation rate remains high during the classes, which was not expected. This can possibly be explained by the fact that the students are constantly wearing the radio boxes around their necks, which means that they can interact in class when the students are sitting close to each other when they are not really communicating.

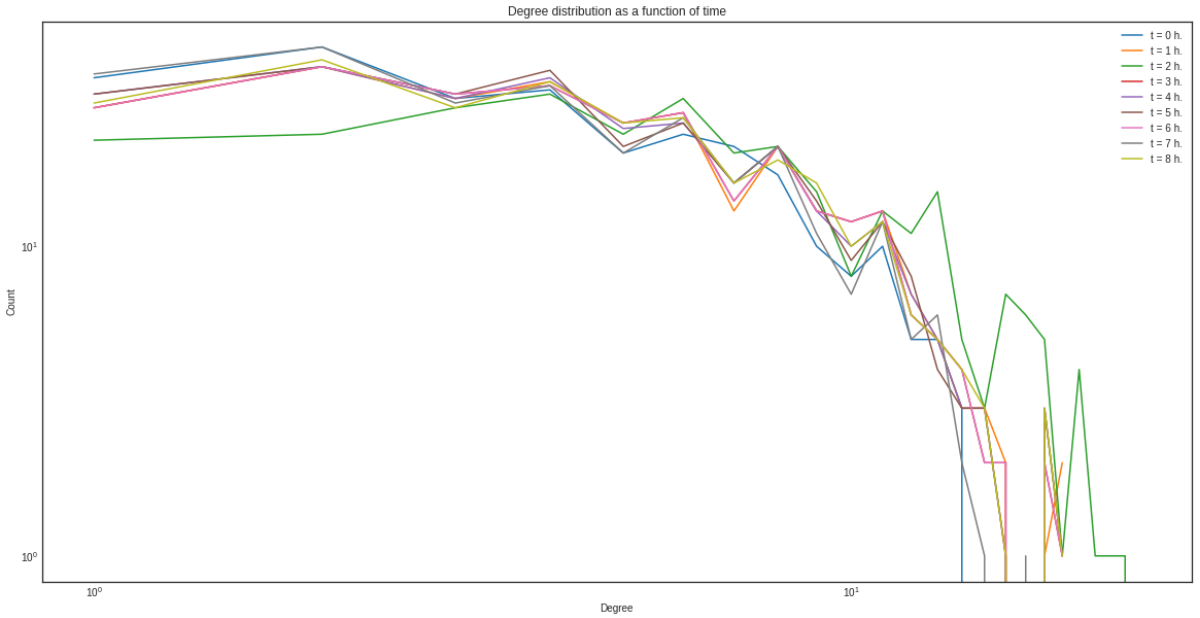
## 3.3 Dynamic contact Network properties analysis

Let us now study the elementary properties of this network. As it evolves very rapidly over time (every 20 seconds), we will average the interactions over one hour to obtain graphs that evolve slowly. We calculate the distribution of degrees, clustering coefficient and diameter on the 9 graphs obtained by aggregating the interactions of the day hour by hour.

In [286]:
```
def NodesSetAttributes(g):
    for k in list(g.nodes()):
        g.nodes[k]['section']= str(list(Meta.section[Meta.i==int(k)
        g.nodes[k]['sex']= str(list(Meta.sex[Meta.i==int(k)])[0])

def Drawing(g):
    nx.draw(g,pos=nx.spring_layout(g, k=0.3), node_color = [plt.cm.


def TimeAnalysis(t_start, step, Nb_step, df=ContactDynamics, plot=T

    for h in range(Nb_step):


        g = Cumulated(h*step + t_start, (h+1)*step + t_start)
        NodesSetAttributes(g)

        if plot :
            plt.figure()
            plt.title("t=" + str((h*step)/60 + t_start/60) + "min")

            Drawing(g)

            for u in list(set(list(nx.get_node_attributes(g,'sectio
                plt.scatter([],[], color=[plt.cm.tab10(u) for u in

            plt.legend()
            plt.show()
```

```
def DegreeDistribution(t_start, step, Nb_step, df=ContactDynamics):
    plt.figure(figsize=(20,10))
    plot=[]
    for h in range(Nb_step) :


        g = Cumulated(h*step + t_start, (h+1)*step + t_start)

        deglist=[]
        for k in list(nx.degree(g)):
            deglist.append(k[1])

        C = collections.Counter(deglist)
        x, y = [], []


        for k in range(len(C)):
            x.append(k+1)
            y.append(C[k+1])

        plot.append(y)

        plt.loglog(x, y, label="t = " + str(h) + " h.")

        plt.legend()
    plt.xlabel("Degree")
    plt.ylabel("Count")
    plt.title("Degree distribution as a function of time")
    plt.show()
```

In [287]:
```
DegreeDistribution(19.5*3600, 3600, 9)
```

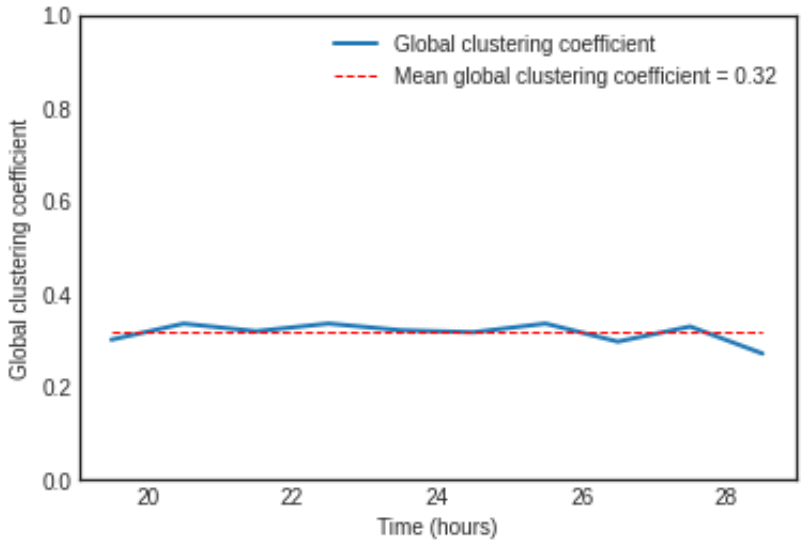


We notice that the MP and MP*1 classes have a constant activity during the day which is 80% and 100% respectively. This means for the MP*1 that all students had at least one contact every 20 minutes.

The other classes behaved differently with periods of less activity and more activity corresponding to breaks between classes. PSI* is a typical example where there is a 30 to 60 percent difference between rest and activity periods. However, we do not observe the total absence of certain classes that could explain the drop in activity during classes. They all modulate their activity in the same way. The participation rate remains high during the classes, which was not expected. This can possibly be explained by the fact that the students are constantly wearing the radio boxes around their necks, which means that they can interact in class when the students are sitting close to each other when they are not really communicating.

In [288]:
```
def GlobalCC(t_start, step, Nb_step, df=ContactDynamics):
    t = np.arange(t_start, t_start + step*Nb_step, step)/3600
    GlobalCC = []

    for h in range(Nb_step):
        g = Cumulated(h*step + t_start, (h+1)*step + t_start)

        A = (g.subgraph(c) for c in nx.connected_components(g))
        giant = list(A)[0]

        C = nx.average_clustering(giant)

        GlobalCC.append(C)

    plt.figure()
    plt.plot(t,GlobalCC, label = "Global clustering coefficient", l
    plt.plot([t[0], t[-1]], [np.mean(GlobalCC), np.mean(GlobalCC)],
    plt.ylim(0,1)
    plt.xlabel("Time (hours)")
    plt.ylabel("Global clustering coefficient")
    plt.legend()
    plt.show()
```

In [289]:
```
GlobalCC(19.5*3600, 3600, 10)
```



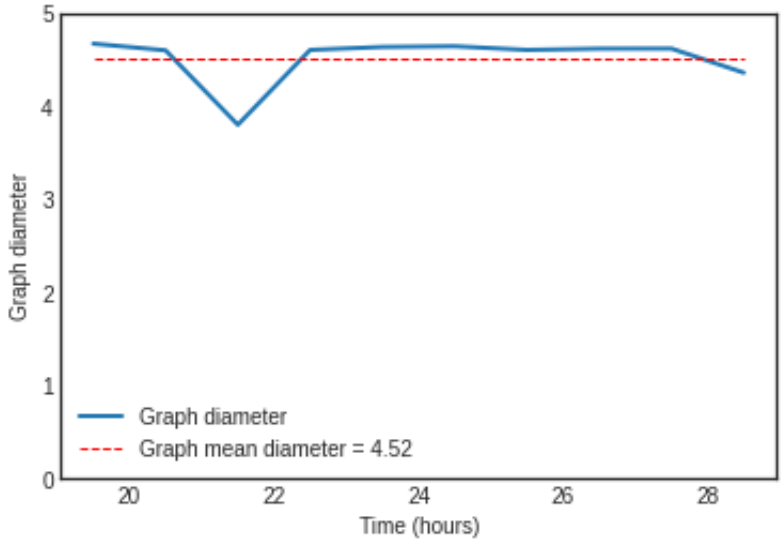

Here again the graph is stable over time since the clustering coefficient is relatively constant over time. Moreover its average value over the day is 0.32 is high, especially compared to scale-free networks or random graphs of the Edos-Renyi type as shown below.

In [290]:
```
h0 = Cumulated(19.5*3600, 19.5*3600 + 3600);   #Comme le réseau est
n = len(h0.nodes())     #Nombres de noeuds dans le graphe cumulé su
m = len(h0.edges())     #Nombres d'arrêtes dans le graphe cumulé su

g = nx.gnm_random_graph(n, m)
print("Clustering coefficient pour un graphe aléatoire: ", nx.avera
```

```
Clustering coefficient pour un graphe aléatoire:  0.01933324686133
6743
```

In [291]:
```
def Diameter(t_start, step, Nb_step, df=ContactDynamics):
    t = np.arange(t_start, t_start + step*Nb_step, step)/3600
    d = []

    for h in range(Nb_step):
        g = Cumulated(h*step + t_start, (h+1)*step + t_start)

        A = (g.subgraph(c) for c in nx.connected_components(g))
        giant = list(A)[0]

        d.append(nx.average_shortest_path_length(giant))

    plt.figure()
    plt.plot(t,d, label = "Graph diameter", linewidth=2)
    plt.plot([t[0], t[-1]], [np.mean(d), np.mean(d)], color='red',
    plt.xlabel("Time (hours)")
    plt.ylabel("Graph diameter")
    plt.ylim(0,5)
    plt.legend()
    plt.show()

    return d
```

In [292]:
```
Diameter(19.5*3600, 3600, 10);
```




Again, the graph is stable over time since the clustering coefficient is relatively constant over time. In addition, its average value over the day of 0.32 is high, especially when compared to networks without a scale or Edos-Renyi type randomized graphs, as shown in the graph below.

To gain even more accuracy, the following section study the networks at the single person level while integrating time resolution.

# IV. Dynamic Graphs at the Single Person Level

We will try here to understand how these interactions, this network dynamic is observed at the level of a person, how to characterize his role within the network but also his place within the various interactions of the high school. We will therefore study different parameters in order to understand how high school students interact and communicate.

## 4.1 Preliminary steps

In [294]:
```
#Observation of time
g = tn.read_interactions(file=r'High-School_data_2013.csv', frequen
L_t=g.snapshots_timesteps()
print(L_t)
print(len(L_t))
```

```
graph will be loaded as:  <class 'tnetwork.dyn_graph.dyn_graph_s
n.DynGraphSN'>
[1385982020, 1385982040, 1385982060, 1385982080, 1385982100, 138
5982120, 1385982140, 1385982160, 1385982180, 1385982200, 1385982
220, 1385982240, 1385982260, 1385982280, 1385982300, 1385982320,
1385982340, 1385982360, 1385982380, 1385982400, 1385982420, 1385
982440, 1385982460, 1385982480, 1385982500, 1385982520, 13859825
40, 1385982560, 1385982580, 1385982600, 1385982620, 1385982640,
1385982660, 1385982680, 1385982700, 1385982720, 1385982740, 1385
982760, 1385982780, 1385982800, 1385982820, 1385982840, 13859828
60, 1385982880, 1385982900, 1385982920, 1385982940, 1385982960,
1385982980, 1385983000, 1385983020, 1385983040, 1385983060, 1385
983080, 1385983100, 1385983120, 1385983140, 1385983160, 13859831
80, 1385983200, 1385983220, 1385983240, 1385983260, 1385983280,
1385983300, 1385983320, 1385983340, 1385983360, 1385983380, 1385
983400, 1385983420, 1385983440, 1385983460, 1385983480, 13859835
00, 1385983520, 1385983540, 1385983560, 1385983580, 1385983600,
1385983620, 1385983640, 1385983660, 1385983680, 1385983700, 1385
983720, 1385983740, 1385983760, 1385983780, 1385983800, 13859838
```

In [ ]:
```
#Get a list of Time Indexed Graphs

L_G=g.snapshots()
```

In [ ]:
```
#Observe one of this graph

G=L_G[ 1385982020+99480]
nx.draw(G,with_labels = True)

print(G.nodes())
print(G.number_of_nodes())
print(G.degree(9))

neighbor=[k for k in G.neighbors(9)]
print(neighbor)
```

We notice that each point of the graph has one or two edges so we can observe communications between 2 or 3 people. Why in a group of 3 not everyone speaks to each other? This may be due to the way the data is retrieved using transmitters and receivers that may have some acquisition defects.

## 4.2 General study of the dynamic graph and interactions over time

In [ ]:
```
L_Nodes=[]
L_temps = []
t_init=L_t[0]
Edges=[]
edges_tot=0
Nbre_Nodes=[]
Density=[]



for i in L_t:
    G_i = L_G[i]
    L_temps.append(i-t_init)
    Nodes = list(G_i.nodes()) #All persons communicating at time t
    Nbre_Edges = G_i.number_of_edges() #Number of interactions at t
    Edges.append(Nbre_Edges)
    edges_tot = edges_tot +Nbre_Edges #Total number of communicatio

    Nbre_nod = G_i.number_of_nodes() #Number of people interacting
    Nbre_Nodes.append(Nbre_nod) #Ntotal number of people who have i
    dens = nx.classes.function.density(G_i) #Density at time t
    Density.append(dens)


    #We are now going to study each of the nodes over time.

    for k in Nodes:

        Deg_k=G_i.degree(k)
```

```
        if k not in L_Nodes:
            L_Nodes.append(k)



            if k in list(FRIENDSHIP.nodes()):
                FRIENDSHIP.nodes[k]['time'] = 20 #An interaction ti
                FRIENDSHIP.nodes[k]['Deg_temp'] = Deg_k # We add th

            if k in list(FACEBOOK.nodes()):
                FACEBOOK.nodes[k]['time'] = 20
                FACEBOOK.nodes[k]['Deg_temp'] = Deg_k

        else :

            if k in list(FACEBOOK.nodes()):
                FACEBOOK.nodes[k]['time'] += 20  #An interaction ti
                FACEBOOK.nodes[k]['Deg_temp'] += Deg_k # We add the

            if k in list(FRIENDSHIP.nodes()):
                FRIENDSHIP.nodes[k]['time'] += 20
                FRIENDSHIP.nodes[k]['Deg_temp'] += Deg_k

between_friendship= nx.betweenness_centrality(FRIENDSHIP)
between_facebook= nx.betweenness_centrality(FACEBOOK)

nx.set_node_attributes(FRIENDSHIP,between_friendship,name='betweene
nx.set_node_attributes(FACEBOOK,between_facebook,name='betweeness')
nx.set_node_attributes(FRIENDSHIP,dict(FRIENDSHIP.degree()),name='d
nx.set_node_attributes(FACEBOOK,dict(FACEBOOK.degree()),name='degre
nx.set_node_attributes(FRIENDSHIP,nx.clustering(FRIENDSHIP),name='c
nx.set_node_attributes(FACEBOOK,nx.clustering(FACEBOOK),name='clust
```

The aim here is to have a dynamic evolution of the different parameters such as edges nodes etc but also to take note of the speaking time and the number of interactions that each person has during these 5 days (knowing that a person can interact with 2 persons during 20 seconds). We will try to see if there is a correlation between the role that a person has in this network ("popular", "With a lot of friends" (degree), "Bridges between different classes/communities" (betweeness) is what he is in a rather 'tight' group (clustering-coefficient) etc. ...) and the time they spend interacting or the number of interactions they have over time.

## 4.3 Total talking time of each student as a function of its centrality measures : Friendship graph

In [ ]:
```
figsize = (16,3)
plt.figure(figsize=(16,0))
plt.title('Time talking = f(FRIENDSHIP)')
plt.axis('off')
```

```
plt.axis('off')
plt.figure(figsize = figsize) #time talking FRIENDSHIP
plt.subplot(131)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'betweeness').value
plt.xlabel('betweeness')
plt.ylabel('time talking (seconds)')
plt.xscale("log")
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])
plt.xlim([0.00001,0])


plt.subplot(132)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'degree').values())
plt.xlabel('degree')
plt.ylabel('time talking (seconds)')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])

plt.subplot(133)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'clustering').value
plt.xlabel('clustering')
plt.ylabel('time talking (seconds)')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])

plt.figure(figsize=(16,0))
plt.title('Time talking = f(FACEBOOK)')
plt.axis('off')

plt.figure(figsize = figsize) #time talking FACEBOOK
plt.subplot(131)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'betweeness').values(
plt.xlabel('betweeness')
plt.ylabel('time talking (seconds)')
plt.xscale('log')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])
plt.xlim([0.00001,0])



plt.subplot(132)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'degree').values()),l
plt.xlabel('degree')
plt.ylabel('time talking (seconds)')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])


plt.subplot(133)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'clustering').values(
plt.xlabel('clustering')
plt.ylabel('time talking (seconds)')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.ylim([0,70000])

plt.figure(figsize=(16,0))
```

```
plt.title('Number of interactions = f(FRIENDSHIP)')
plt.axis('off')

plt.figure(figsize = figsize) #number interactions FRIENDSHIP
plt.subplot(131)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'betweeness').value
plt.xlabel('betweeness')
plt.ylabel('Nbre of people you talk to')
plt.yscale('log')
plt.xscale('log')
plt.ylim([10,10000])


plt.subplot(132)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'degree').values())
plt.xlabel('degree')
plt.ylabel('Nbre of people you talk to')
plt.yscale('log')
plt.ylim([10,10000])

plt.subplot(133)
plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'clustering').value
plt.xlabel('clustering')
plt.ylabel('Nbre of people you talk to')
plt.yscale('log')
plt.ylim([10,10000])


plt.figure(figsize=(16,0))
plt.title('Number of interactions = f(FACEBOOK)')
plt.axis('off')

plt.figure(figsize = figsize) #number interactions FACEBOOK
plt.subplot(131)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'betweeness').values(
plt.xlabel('betweeness')
plt.ylabel('Nbre of people you talk to')
plt.xscale('log')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))


plt.subplot(132)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'degree').values()),l
plt.xlabel('degree')
plt.ylabel('Nbre of people you talk to')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))


plt.subplot(133)
plt.plot(list(nx.get_node_attributes(FACEBOOK,'clustering').values(
plt.xlabel('clustering')
plt.ylabel('Nbre of people you talk to')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
```

We observe a priori no or few correlations between the role of a person in this network and the number of interactions or the time of interaction that this one has during the day (/week) but this result may seem rather surprising, indeed someone with more friends IRL / or on FACEBOOK should communicate much more than a 'solitary' person or with few friends so how to explain this somewhat surprising result? Let's try to take a little more interest in the individual and his relationships with his friends. To do this we will look at the rate of interaction that a student has with each of his contacts. This rate for a contact is defined by the relationship: Rate= Nbre_of_interaction_with_this_contact/Total_of_student_interaction

We will therefore try to see if this student has a special relationship with another student or if he communicates equally with all his friends.

## 4.4 Study of privileged relationships

In [ ]:

```
L_temps = []
t_init=L_t[0]
Edges=[]
edges_tot=0
Nbre_Nodes=[]
Density=[]

p =  np.random.randint(0,len(L_Nodes)) #we choose a random student
Node_interest = L_Nodes[p]
print(Node_interest)
Deg_p=[]
deg_tot=0
L_contact=[]
Taux=[]
Taux2=[]

classe_p= str(list(Meta.section[Meta.i==Node_interest])[0]) #We not
#sexe_p= str(list(Meta.sex[Meta.i==Node_interest])[0])   #We note t

for i in L_t:
    G_i = L_G[i]
    L_temps.append(i-t_init)
    Nodes = list(G_i.nodes())


    if Node_interest in Nodes: #The student communicates at time t

        Deg_p.append(G_i.degree(Node_interest)) #We note how many i
        deg_tot=deg_tot+G_i.degree(Node_interest) #The total number


        neighbors=[neighbor for neighbor in G_i.neighbors(Node_inte
        #Each person contacted by the student is identified.


        for neighbor in neighbors:
            if neighbor not in L_contact :
```

```
                l=len(L_contact) #Students are added to the "contac

                classe_neighbor= str(list(Meta.section[Meta.i==neig
                sexe_neighbor= str(list(Meta.sex[Meta.i==neighbor])

                L_contact.append(neighbor)
                Taux.append(1) #he has communicated once with the s

                if classe_neighbor == classe_p:
                    Taux2.append((1,1)) #they are from the same cla
                elif classe_neighbor != classe_p:
                    Taux2.append((1,0))   # they are not from the s

            else: #The two students have already communicated toget
                indice=L_contact.index(neighbor)

                Taux[indice]+=1 #We add a communication
                Taux2[indice]= (Taux2[indice][0]+1,Taux2[indice][1]




    else :
        Deg_p.append(0)#No communication there



Taux = np.array(Taux)/deg_tot #Communication rate with a person= Nb
Taux2 = np.array(Taux2)/deg_tot
Taux_sorted2= sorted(Taux2, key=lambda Taux2: Taux2[0] ,reverse=Tru
#We put the person with whom the student p communicates the most fi
```

In [ ]:
```
L=[i for i in range (0,len(Taux))]
print(deg_tot)

plt.figure(figsize = (16,5))
plt.subplot(121)
plt.plot(L_temps,Deg_p)#Communication evolution as a fnction of tim
#plt.xlim([99400,99600])

plt.xlabel('time (s)')
plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))

plt.ylabel("node's degree(t)")

#print(L_contact)
#print(Taux)

Taux_sorted=sorted(Taux ,reverse=True) #Pourcentage de temps de par
#dans l'ordre décroissant du pourcentage

plt.subplot(122)
plt.plot(L,Taux_sorted,'o')
plt.ylabel('Communication rate')
plt.xlabel('Contact')
plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
```

In [ ]:
```
L=[i for i in range (0,len(Taux))]
Taux_bis=[]
L_color=[]
Taux_sorted2= sorted(Taux2, key=lambda Taux2: Taux2[0] ,reverse=Tru
for i in range(0,len(Taux2)):
    Taux_bis.append(Taux_sorted2[i][0])
    #For each person, the percentage of interaction that the studen
    L_color.append(Taux_sorted2[i][1])
    #The dot is green if they are of the same red class otherwise


plt.scatter(L, Taux_bis, c=L_color, s=25, cmap='RdYlGn')
plt.xlabel('contact')
plt.ylabel('communication rate')

plt.show()
```

We generally notice that people have a privileged relationship with a person of the same class who occupies 30 to 40% of their interactions against 20% for the second one. Of course this result is general and some specific cases contradict this, but let's look at it in a general way.

In [ ]:
```
Nbre_student= len(L_Nodes)
Taux_moy=[0]*10
Pourcentage_sex=0
Pourcentage_classe=0
solo=0
STD=[]

for i in range(0,10):
```

```
for i in range(0,10):
    STD.append([])


for student in L_Nodes:#We will do the same study as before with ea
    classe_student= str(list(Meta.section[Meta.i==student])[0])
    sexe_student= str(list(Meta.sex[Meta.i==student])[0])
    Deg_student=[]
    deg_tot=0
    L_contact=[]
    Taux=[]
    Taux2=[]
    for i in L_t:
        G_i = L_G[i]
        Nodes = list(G_i.nodes())


        if student in Nodes:


            deg_tot=deg_tot+G_i.degree(student)
            neighbors=[neighbor for neighbor in G_i.neighbors(stude

            for neighbor in neighbors:
                if neighbor not in L_contact :
                    l=len(L_contact)
                    classe_neighbor= str(list(Meta.section[Meta.i==
                    sexe_neighbor= str(list(Meta.sex[Meta.i==neighb
                    L_contact.append(neighbor)
                    Taux.append(1)
                    if classe_neighbor == classe_student:
                        if sexe_neighbor == sexe_student: #They ide
                            Taux2.append((1,1,1))

                        else :
                            Taux2.append((1,1,0)) #Same class but d

                    elif classe_neighbor != classe_student:
                        if sexe_neighbor == sexe_student:
                            Taux2.append((1,0,1))

                        else :
                            Taux2.append((1,0,0))


                else:
                    indice=L_contact.index(neighbor)

                    Taux[indice]+=1
                    Taux2[indice]= (Taux2[indice][0]+1,Taux2[indice



    Taux = np.array(Taux)/deg_tot

    Taux2 = np.array(Taux2)
    Taux_sorted2= sorted(Taux2, key=lambda Taux2: Taux2[0] ,reverse
    Pourcentage_sex= Pourcentage_sex +Taux_sorted2[0][2] #If the st
```

```
    #with a person of the same gender : +1 otherwise +0
    Pourcentage_classe= Pourcentage_classe + Taux_sorted2[0][1] #If
    #with a person of the same class : +1 otherwise +0

    Taux_sorted= sorted(Taux,reverse=True)

    Taux_sorted2=np.array(Taux_sorted2)/deg_tot


    if student in list(FRIENDSHIP.nodes()): #We want to find out wh
        # give more or less time to their 'privileged relationship'
        FRIENDSHIP.nodes[student]['Bestfriend'] = Taux_sorted[0]


    if student in list(FACEBOOK.nodes()):
        FACEBOOK.nodes[student]['Bestfriend'] = Taux_sorted[0]


    if len(Taux_sorted)>10: #At this time, only those with more tha
        #(Large majority)
        for i in range (0,10):
            Taux_moy[i] =Taux_moy[i]+Taux_sorted[i] #We want to obs
            STD[i].append(Taux_sorted[i])
    else:
        solo=solo+1

L=[i for i in range (1,11)]
Taux_moy=np.array(Taux_moy)/(Nbre_student-solo)
Pourcentage_sex= Pourcentage_sex/Nbre_student #We observe the Perce
#the same gender

Pourcentage_classe= Pourcentage_classe/Nbre_student #We observe the
#the same class
```

In [ ]:

```
print('there are '+str(solo)+ ' of the students with less than 10 c
#Number of the students with less than 10 contacts
print('There are ' +str(Nbre_student)+' students') #Number of stude

print(str(int(Pourcentage_sex*100)) +'% privileged relations are wi
#Percentage of privileged relations with a person of the same gende
print(str(int(Pourcentage_classe*100)) +'% privileged relations are
#Percentage of privileged relations with a person of the same class
```

On remarque donc qu'à priori le genre a un léger impact mais qu'au cours de la journée la grande majorité des personnes intéragissent en priorité avec une personne de leur classe ce qui semble logique étant donnée qu'en une journée de cours on aura plutôt tendance à communiquer avec des personnes de notre classe lors des cours des pauses ou pour parler des cours qui prennent une grande partie de la journée surtout en prépa. De plus la prépa fait en sorte que l'on ne connais préférentiellement que des gens de notre classes: On arrive tous inconnus et plutôt peu d'interactions en dehors de plus classes similaires d'une année à l'autre.

In [ ]:
```
L=[i for i in range (0,10)]
std_moy=[]


for i in range (0,10):
    std_moy.append(np.std(STD[i]))


plt.plot(L,Taux_moy,'o',label='Average communication rate')
#Taux d'intérêt moyen accordé aux dix premiers contacts par la moye
plt.errorbar(range(len(L)), Taux_moy, yerr = std_moy,
                fmt = 'none',ecolor='black' ,capsize = 10, elinewid

fitopti=scipy.optimize.curve_fit(lambda t,a,b,c: a*np.exp(b*t)+c,
print(fitopti)

#fitexp=np.polyfit(L, np.log(Taux_moy), 1)
#print(fitexp)
#plt.plot(np.exp(fitexp[0])*np.exp(fitexp[1]*np.array(L)))

plt.plot(fitopti[0][2]+fitopti[0][0]*np.exp(fitopti[0][1]*np.array(
plt.xlabel('Contact')
plt.ylabel('Average communication rate for every contact')
plt.legend()
plt.show()
```

It is therefore noticeable that the behavior consisting in granting almost twice as much time to the privileged contact compared to the 2nd contact seems to be quite common. Moreover, this decrease is accelerating quite quickly: 5% for the 5th contact, i.e. 8 times less than the 1st contact. This decrease is modeled 'anecdotally' well with an exponential decrease. It can be interpreted in the following way: an affinity is created between two people from the same class who will be together in class or in pairs of labs, which favours a significant shift in interactions between the different contacts.

We can also begin to understand the fact that a person will have similar communication time to others regardless of his role because he will generally have a privileged relationship that will occupy a good part of his time and will compensate for his 'lack of popularity'.

Let's see if the rate of communication with the privileged relationship is favored/disadvantaged or neutral depending on the student's role in the network.

We can also see that the time given to the 2 first contacts is quite variable but seems to remain much higher than the others (only the second contact eventually remains in competition with the first in a few cases).

## 4.5 Interest rate of the privileged relationship

In [ ]:

```
plt.figure(figsize = (16,0))
plt.title('FACEBOOK : Interest rate')
plt.axis('off')

plt.figure(figsize = (16,3))
plt.subplot(131)
plt.title('Interest rates as a function of the degree')

plt.plot(list(nx.get_node_attributes(FACEBOOK,'degree').values()),l
plt.xlabel('degree')
plt.ylabel('Interest rate')

plt.subplot(132)
plt.title('Interest rates as a function of the betweeness')
plt.plot(np.log(list(nx.get_node_attributes(FACEBOOK,'betweeness').
plt.xlabel('log(betweeness)')
plt.ylabel('Interest rate')


plt.subplot(133)
plt.title('Interest rates as a function of the clustering coefficie
plt.plot(list(nx.get_node_attributes(FACEBOOK,'clustering').values(
plt.xlabel('clustering')
plt.ylabel('Interest rate ')


plt.figure(figsize = (16,0))
plt.title('FRIENDSHIP : Interest rate')
plt.axis('off')

plt.figure(figsize=(16,3))
plt.subplot(131)
plt.title('Interest rate  as a function of the degree')

plt.plot(np.log(list(nx.get_node_attributes(FRIENDSHIP,'degree').va
plt.xlabel('log(degree)')
plt.ylabel('log(Interest rate) ')


plt.subplot(132)
plt.title('Interest rates as a function of the betweeness')

plt.plot(np.log(list(nx.get_node_attributes(FRIENDSHIP,'betweeness'
plt.xlabel('log(betweeness)')
plt.ylabel('Interest rate')


plt.subplot(133)
plt.title('Interest rates as a function of the clustering coefficie

plt.plot(list(nx.get_node_attributes(FRIENDSHIP,'clustering').value
plt.xlabel('clustering')
plt.ylabel('Interest rate ')
```

No apparent correlation except for the degree graph (in log-log there is a fairly good correlation). This can be explained by the fact that students with few or some friends will have a privileged relationship while a 'popular' person with a higher degree will have less time to spend on their privileged relationship due to their higher number of contacts to whom they necessarily devote a minimum amount of time (although this difference in the degree graph is not so important as it should be visualized in log-log scale). We also notice that this correlation does not apply to the FACEBOOK network because it is quite simple to have a lot of friends without having to spend time with them on a social network.

# V. Conclusions

To conclude, this work brought different informations characterizing the social network at work in this high school.

First, from the study of the static graphs emerges that some nodes are more important than other nodes, by looking at the degree distribution and the betweenness centrality. These simple measures already highlight how different is the graph from Facebook. According to the degree distribution, people tend to connect easily than in the real life. But this graphs is also interesting in terms of communities. The two graphs from real-life interactions tend form clusters based on section more than sex, shown by assortativity measures. Here again, the graph from Facebook is different since it structure is not understandable through these measures. It is only by merging the nodes attributes according to the scientific field they belong, that the FACEBOOK structure can be understood. When repeating centrality and assortativity measures on each class, no clear trend emerge, probably because the scale of study is too narrow to apply such tools.

Then, the general analysis of the dynamic contact dataset showed periodic patterns. We can see spikes emerging in the contact count graph with a periodicity of two hours corresponding to the breaks between classes. We can see it as well when ploting the interval graph. Then we highlighted the difficulty to perform dynamic community detection since the edges are changing over time and studied the percentage of presence of students in the graph grouped by classes. We couldn't identify much differences between the different classes during the day and the decrease of activity between two peaks is not due to the absence of interactions within classes because of lectures. Every class lower its interaction during class time and increases it during break time. Finally we showed that the network has all the caracteristics of a small world network.

When looking at the single person level, the study showed that it is quite complex to define a person's friendships according to their 'role' within a network. Indeed, in order to better understand the place that a person takes within the social dynamic, it is necessary to deepen the study and to go beyond his or her initial biases. This has led to some very interesting points: First of all, a person with a key role in their school is not necessarily more or less solitary, and does not necessarily communicate much more than a person with a limited circle of friends.

Furthermore, it was particularly interesting to note that, in general, the same dynamics and hierarchy generally apply to the time allocated to each of his contacts by a pupil, which clearly tends to favour dialogue with one or two people rather than with a group of people where he would communicate in an equitable manner. As this favouritism is greatly influenced by the classes, we have tried to interpret it through our knowledge of the general dynamics of the preparatory classes (lab pairs, Oral interrogations trinom etc...) which tends to reinforce a relationship (or at least communication) between two people.This also raises the question of the influence of the context in which two individuals evolve on their relationship: is the context more important than affinity?

Finally, it was found that despite the central role of certain people in general, there was only a slight difference in the time each person devotes to his or her privileged relationship, except in the case of people with a lot of 'friends' who seem to devote slightly less time to this privileged relationship. However, this difference is not significant and does not allow us to describe a general trend, especially as these cases of high degree students are quite rare.

# Annexes

## Dynamique d'évolution des communautées

In [ ]:
```
import tnetwork.DCD as DCD
```

In [ ]:
```
last_time = g.end()
L_t=g.snapshots_timesteps()
lenL_t=len(L_t)
print(last_time)
t_start= L_t[0]
times_to_plot = [t_start,L_t[int(lenL_t/3)],L_t[int(lenL_t/3*2)],L_
plot = tn.plot_as_graph(g,ts=times_to_plot,width=200,height=200)
```

In [ ]:
```
com_iterative = DCD.iterative_match(g)
```

In [ ]:
```
custom_match_function = lambda x,y: len(x&y)/max(len(x),len(y))
com_custom = DCD.iterative_match(g,match_function=custom_match_func
```

In [ ]:
```
plot = tn.plot_as_graph(g,com_iterative,ts=times_to_plot,auto_show=
```

```
Sur 2 secondes que des groupes de 2 ou 3 donc on ne va pas pouvoir
déterminer de vrai communauté. On détecte juste les binômes et
trinomes de gens qui parlent entre eux. On va donc cheercher à
déterminer les communautées sur le diagramme dynamic aggrégé sur
une heure afin que des groupes de discussions plus amples ne se
forment.
```

In [ ]:
```
g2 =g.aggregate_time_period("hour")
```

In [ ]:
```
last_time = g2.end()
L_t2=g2.snapshots_timesteps()
lenL_t2=len(L_t2)
print(last_time)
t_start= L_t2[0]
times_to_plot = [t_start,L_t2[int(lenL_t2/3)],L_t2[int(lenL_t2/3*2)
plot = tn.plot_as_graph(g2,ts=times_to_plot,width=200,height=200)
```

In [ ]:
```
com_iterative = DCD.iterative_match(g2)
```

In [ ]:
```
custom_match_function = lambda x,y: len(x&y)/min(len(x),len(y))
com_custom = DCD.iterative_match(g2,match_function=custom_match_fun
```

In [ ]:
```
plot = tn.plot_as_graph(g2,com_custom,ts=times_to_plot,auto_show=Tr
```

In [ ]:
```
to_plot = tn.plot_longitudinal(g2,com_custom,height=200)
```

In [ ]:
```
com_survival = DCD.label_smoothing(g2)
plot = tn.plot_longitudinal(g2,com_survival,height=200)
```

In [ ]:
```
com_smoothed = DCD.smoothed_louvain(g2)
```

In [ ]:
```
plot = tn.plot_longitudinal(g2,com_smoothed,height=200)
```

In [ ]:
```
custom_match_function2 = lambda x,y: len(x&y)/max(len(x),len(y))
com_custom_2 = DCD.iterative_match(g2,match_function=custom_match_f
```

In [ ]:
```
plot = tn.plot_longitudinal(g2,com_custom_2,height=200)
```

Complexe et pas assez de temps pour trouver bonne méthode mais quelques commu détectés pourrait être approfondi

In [ ]:
```
```