Developing a Networkx Graph
The Count of Monte Cristo

Peter Trubey

May 15, 2014

Abstract

This paper will document the effort to create a working social network of character interaction from the book *The Count of Monte Cristo*, using basic python commands. I will discuss some of the problems encountered, interpretation, development of edge weight between nodes (characters), and more.

1 Introduction

The book 'The Count of Monte Cristo', written in 1844, is an extremely good read and offers a very interesting view of Aristocratic France, during after the reign of Napoleon Bonaparte. In the book, the titular character, Edmond Dantes, is wrongly imprisoned by the actions of three conspirators, and the inaction of a fourth. In prison he befriends another prisoner; a man of the world who trains him in matters aristocratic. He learns art, economics, mathematics, manner of speech and languages around the world, and the existence of a vast fortune buried by an extremely wealthy member of the clergy. Dantes breaks out of prison, seeks out the fortune, and sets about exacting a plan of revenge against the conspirators who wronged him. The text itself was retrieved from Project Gutenberg's website [1].

Throughout the book Dantes maintains a set of aliases that he uses for interaction to advance his goals. After spending more than 10 years in prison, the fortunes of those conspirators he seeks have changed. Fernand Mondego, the man who accused Dantes of treason so he could steal Dantes' betrothed, is now the Count de Morcerf, elevated by his actions fighting in the army of Spain. He succeeded in courting Dantes' betrothed, Mercedes, and fathered a son, Albert. Danglars, Dantes' junior officer whose jealousy had him propose the conspiracy, is now a Baron by marriage, and a wealthy banker. Gerard de Villefort, the investigator who sent Dantes to prison to protect his own career, is now a judge, and has a reputation as a hanging judge. It is to this scene that we see Dantes, now under the alias of the enigmatic Count of Monte Cristo, return. Calculating and charismatic, Dantes worms his way into their lives to destroy them, as they destroyed his.

In this paper I will describe the algorithm by which I created the network graph, discuss its bonuses and drawbacks, and compare the results of the graphs when I keep the various aliases of Dantes separate, and when I merge them.

2 Methodology

2.1 The Idea

So to establish a social network, we need to establish character interaction, or a relationship existing between characters. There are automated methods we can use for this, but I wanted to build something from scratch. So what I came up with is establishing character interaction by *coincidence*. In this sense, I am defining incidence

as a character existing in a particular scene. Therefore, we can regard coincidence as two characters existing in the same scene. This has some benefits, in that it is relatively easy to search for character names in a given chunk of text using regular expressions. However, there are some drawbacks to this method.

2.1.1 The Problems

The first problem with establishing character coincidence is breaking the text into scenes. Earlier we established coincidence as two characters existing in a scene. The problem with that as applied to this book, is that the book is not separated into scenes. If we were working with a play, the scenes would be defined for us. However, working with a novel, we are forced to create arbitrary scene breaks. Breaking on chapters only would be too coarse, as there is only 117 chapters in the book. Breaking on paragraphs would be too fine, as many paragraphs are single line retorts, where the identity of the speaker is implied by context. I work through this by defining a scene as having a minimum character length, and breaking on the first paragraph break after that character length. From that, we establish *scenes* comprising one large paragraph, or of several smaller paragraphs. An advantage of this is if we manage to break a long interaction into several scenes, then the strength of association between those characters increases proportional to the length of interaction. The other problem with this idea of incidence is that a character may not be present when they are mentioned in text. Indeed, the titular character is often discussed when he is not present. I rationalize this by thinking that though they may not be present, if they are discussed then they had some effect on the scene and it is sufficient to think of them as present for this purpose.

The second problem encountered comes from defining character names. One person may go by several names, or titles, or even occupations. For example, Maximilian Morrel, the son of Dantes' employer Pierre Morrel, at times goes by Maximillian, Morrel, or Captain of Spahis. However, Pierre Morrel also goes by the name Morrel. Solving this requires breaking the book into two parts at around 35%, before which all references to Morrel are to the father, and past which all references to Morrel are to the son. Another problem of defining names arises when one character's name is a substring of anothers. For example, Gerard de Villefort is one of the primary antagonists of the book. He is frequently referred to as *Villefort*. His wife, Heloise, is also referred to as Madame de Villefort. His daughter, Valentine, is also referred to as Mademoiselle de Villefort. Solving these problem required the use of lookahead and lookbehind assertions in the regular expressions, so that mention of only *Villefort* without Mademoiselle or Madame would correspond to Gerard de Villefort. A similar problem was encountered for the heads of the other households.

2.2 The Algorithm

With the above problems and solutions in mind, we can define the algorithm created and used to generate the graph.

After splitting the book into what we are defining as scenes, we then perform regular expression searches for all the characters we defined. These results are assembled as row vectors into the matrix X, the matrix of character incidence. The results of those regular expression searches are Boolean, so in python we need to multiply by 1.0 in order to turn them into floats. When we then take the matrix product of X and its transpose to get the matrix of character coincidence. For each cell $[i, j]$, we have the number of times character i appeared in the same scene as character j, and on the diagonal we have the number of scenes each character appeared in. The edge weight formula described in the algorithm gives the geometric mean of the percentage of each character's *screen time* spent with the other character. Thus, we use this as a measure of strength of character ties. The advantage of this measure is it is not heavily skewed against minor characters, and reduces spurious association between major (though unrelated) characters.

Algorithm 1: Developing the Network

1. Read in book as string
2. Split book on chapters
3. While iteration $i <$ length(book):
   - temp = book at index i
   - temp = split temp at first paragraph break after minimum scene length
   - Replace index i in book with temp, inserting remainder into $i + 1$. Increases length of book by 1.
4. Chars = (char names, regular expressions, labels)
5. For all chars:
   - Incidence = regular expression search of all scenes
6. X = matrix of incidence, [number of chars $\times$ number of scenes]
7. $Y = XX^T$ = Matrix of Coincidence, [number of chars $\times$ number of chars]
8. For i in number of characters:
   - For $j < i$:
     - Edge weight between characters $i, j = \sqrt{\frac{Y[i,j]^2}{Y[i,i]*Y[j,j]}}$
     - Only include edge weights on graph $>$ minimum edge weight.

3 Results

On the graphs, I have colored the various aliases of Dantes as green, and his allies as teal. The Morrels are blue, the Villeforts are purple, the Danglars are brown, and the Morcerf's are orange. The specific enemies of Dantes are colored in red.

The resulting networkx Graphs represent the associations of the book fairly well. In the graph holding the aliases of Dantes separate (figure 1), the Count of Monte Cristo persona dominates the graph, holding associations with almost everyone. But it is interesting to see that Caderrouse's wife only has association with Caderrouse and the Abbe Busconi persona. Networkx's layout engine did a good job of keeping the various families somewhat together, and seeing how strong and central Maximilian Morrel's character is is rather interesting.

When we compare against the collapsed graph (figure 2), we see just how much character interaction was held through the various aliases of Dantes. The collapsed graph is much more sparse, as while main characters had interaction with multiple Dantes aliases, on the collapsed graph they only have 1 edge.

One adaptation of The Count of Monte Cristo that I have seen held Albert de Morcerf as a protagonist. From this graph, given his central location and association with the Dantes' main enemies, it certainly seems reasonable to tell the story from his perspective.

As I intentionally created an nondirectional graph, one of the things we can very easily do from here is turn it into a directional graph, using shared screen time as a percentage as the directional weight. We could also attempt using some of the many available text mining tools that utilize more sophisticated ways of dealing with character name collisions, or ambiguous references.

References

[1] Alexandre Dumas. *The Count of Monte Cristo*, volume 1184. Project Gutenberg, 1844.

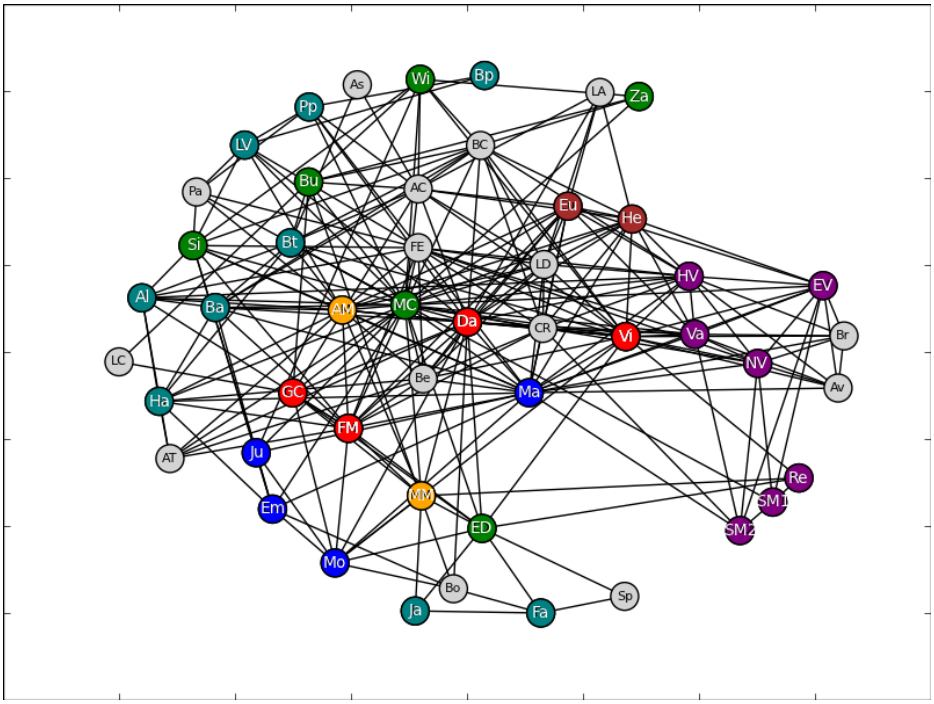


Figure 1: Network Graph with Separate Aliases

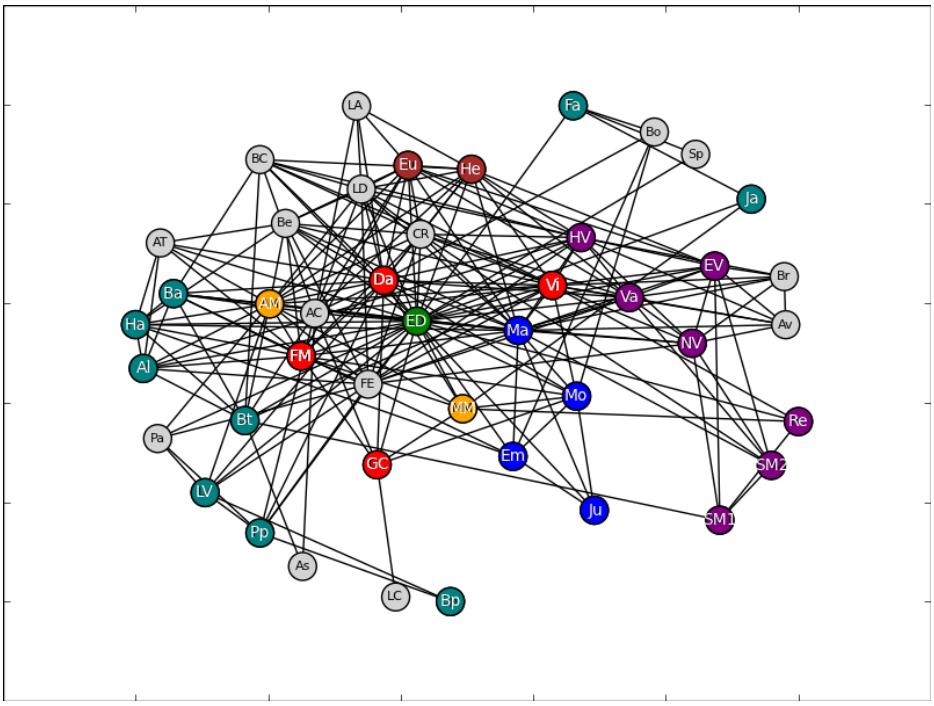


Figure 2: Network Graph collapsing Dantes' Aliases into Edmond Dantes

A Code Used

A.1 Character Regular Expressions

```
chars1a = []
import copy

# First Part of book, Keeping Aliases of Edmond Dantes Separate
# Edmond
chars1a.append(('Edmond Dantes','Edmond|Dantes|Number\s34','ED'));
chars1a.append(('Count of Monte Cristo','Count\sof\sMonte\sCristo|The\sCount|the
    ↪ \scount|The\scount','MC'));
chars1a.append(('Lord Wilmore','Wilmore','Wi'));
chars1a.append(('Sinbad the Sailor','Sinbad','Si'));
chars1a.append(('Abbe Busoni','Busoni','Bu'));
chars1a.append(('Zaccone','Zaccone','Za'));

# Edmond's Allies
chars1a.append(('Abbe Faria','Faria','Fa'));
chars1a.append(('Bertuccio','Bertuccio','Bt'));
chars1a.append(('Luigi Vampa','Luigi|Vampa','LV'));
chars1a.append(('Peppino','Peppino','Pp'));
chars1a.append(('Beppo','Beppo','Bp'));
chars1a.append(('Haidee','Haidee','Ha'));
chars1a.append(('Ali','Ali(?!\sPasha)(?!\sTepelini)','Al'));
chars1a.append(('Baptistin','Baptistin','Ba'));
chars1a.append(('Jacopo','Jacopo','Ja'));

# Morrel Family
chars1a.append(('Old Man Morrel','(?<!Captain\s)(?<!Maximilien\s)(?<!
    ↪ Mademoiselle\s)Morrel(?!\sand\Son)|Pierre','Mo'));
chars1a.append(('Maximilian Morrel','Maximilian|Captain\sof\sSpahis|captain\sof\
    ↪ sSpahis|Captain\sMorrel','Ma'));
chars1a.append(('Julie Herbault','Julie|Mademoiselle\sMorrel|Madam\sHerbault','
    ↪ Ju'));
chars1a.append(('Emmanuel Herbault','Emmanuel|Herbault','Em'));

# Mondego Family
chars1a.append(('Mercedes Mondego','Mercedes|Madame\sde\sMorcerf\sM[.]\sand\
    ↪ sMadame\sde\sMorcerf','MM'));
chars1a.append(('Fernand Mondego','Fernand|Mondego|Count\sde\sMorcerf|M[.]\sde\
    ↪ sMorcerf|M[.]\sand\sMadame\sde\sMorcerf','FM'));
chars1a.append(('Albert de Morcerf','Albert|Viscount\sde\sMorcerf\sViscount\sof\
    ↪ sMorcerf','AM'));

# Danglers Family
chars1a.append(('Danglars','(?<!Eugenie\s)(?<!Madame\s)Danglars','Da'));
chars1a.append(('Hermine Danglars','Baroness\sDanglars|Hermine|Madame\sDanglars'
```

```
    ↪ ,'He'));
chars1a.append(('Eugenie Danglars','Eugenie|Mademoiselle\sDanglars','Eu'));

# Villefort Family
chars1a.append(('Villefort','Judge|Gerard|M[.]\sde\sVillefort|M[.]\sand\sMadame\
    ↪ sde\sVillefort','Vi'));
chars1a.append(('Renee de Villefort','Renee','Re'));
chars1a.append(('Marquis de Saint-Meran','Marquis\sde\sSaint-Meran|Marquis\sand\
    ↪ sMarquise\sde\sSaint-Meran','SM1'));
chars1a.append(('Marquise de Saint-Meran','Marquise de Saint-Meran','SM2'));
chars1a.append(('Valentine de Villefort','Valentine|Mademoiselle\sde\sVillefort'
    ↪ ,'Va'));
chars1a.append(('Noirtier de Villefort','Noirtier','NV'));
chars1a.append(('Heloise de Villefort','Heloise|Madame\sde\sVillefort|M[.]\sand\
    ↪ sMadame\sde\sVillefort','HV'));
chars1a.append(('Edouard de Villefort','Edouard|Edward','EV'));

# Incidental Characters
chars1a.append(('Assunta','Assunta','As'));
chars1a.append(('Pastrini','Pastrini','Pa'));
chars1a.append(('Benedetto','Benedetto|Andrea|Viscount\sof\sCavalcanti|Viscount\
    ↪ sde\sCavalcanti','AC'));
chars1a.append(('Caderousse','Gascard|Caderousse','GC'));
chars1a.append(('La Carconte','Carconte','LC'));
chars1a.append(('Franz d\'Epinay','Franz|d\'Epinay','FE'));
chars1a.append(('Lucien Debray','Lucien|Debray','LD'));
chars1a.append(('Beauchamp','Beauchamp|reporter','Be'));
chars1a.append(('Raoul de Chateau-Renaud','Raoul|Chateau-Renaud','CR'));
chars1a.append(('Louise d\'Armilly','Louise|d\'Armilly','LA'));
chars1a.append(('Boville','Boville','Bo'));
chars1a.append(('Barrois','Barrois','Br'));
chars1a.append(('Avrigny','Avrigny','Av'));
chars1a.append(('Bartolomeo','Bartolomeo|Major\sCavalcanti','BC'));
chars1a.append(('Ali Tepelini','Ali\sTepelini|Yanina','AT'));
chars1a.append(('Cardinal Spada','Spada','Sp'));

# Second Part of Book, Keeping Aliases of Edmond Dantes Separate

# Morrel Family
chars1b = copy.deepcopy(chars1a);
chars1b[15] = ('Morrel','Pierre')
chars1b[16] = ('Maximilian','Maximilian|Captain\sof\sSpahis|captain\sof\sSpahis|
    ↪ Morrel')

#Collapsing all of Dantes' Personas into 1

chars2a = copy.deepcopy(chars1a);
```

```
chars2a[0] = ('Edmond␣Dantes','Edmond|Dantes|Number\s34|Count\sof\sMonte\sCristo
    ↪ |The\sCount|the\scount|The\scount|Thomson\sand\sFrench|Wilmore|Sinbad|
    ↪ Busconi|Zaccone','ED')
del chars2a[1:6]

chars2b = copy.deepcopy(chars2a);
chars2b[10] = ('Morrel','Pierre')
chars2b[11] = ('Maximilian','Maximilian|Captain\sof\sSpahis|captain\sof\sSpahis|
    ↪ Morrel')
```

A.2 Building the Graph GML file

```
import numpy
import pandas
import re

execfile('chars.py')                    # fill the characters regex
txt = open('monte_cristo.txt').read() # open the file

text = txt.split('Chapter')             # split into Chapters
text = text[1:]                         # Remove Gutenberg header


sl = 1000                               # min scene length (chars)
rd = 5                                  # rounding digits, for graph
min_weight = 0.05                       # Minimum Edge Weight for Inclusion

# Building the Scene List

i = 0

while i < len(text):
        if (bool(re.search('\n\n',text[i])) == True) & (len(text[i]) > sl):
                sp = text[i].find('\n\n',sl);
                text = text[:i] + [text[i][:sp],text[i][sp:]] + text[i+1:];
        i += 1;

text = [x.replace('\n\n','␣') for x in text]
text = [x.replace('\n','␣') for x in text]

# Morrel changes meaning from father to son somewhere
# around 35% of the way into book, So I will split the
# book into two parts at about 35% of the way through

sp = int(numpy.ceil(.35*len(text)))

part1 = text[:sp]
part2 = text[sp:]
```

```
#  Building the list of Character Incidence
#  - Defined as a Character appearing in a scene

incidence1a = [[bool(re.search(y[1],x)) for x in part1] for y in chars1a]
incidence1b = [[bool(re.search(y[1],x)) for x in part2] for y in chars1b]

incidence2a = [[bool(re.search(y[1],x)) for x in part1] for y in chars2a]
incidence2b = [[bool(re.search(y[1],x)) for x in part2] for y in chars2b]

incidence1 = []
incidence2 = []

for i in range(0,len(incidence1a)):
        incidence1.append(incidence1a[i] + incidence1b[i])

for i in range(0,len(incidence2a)):
        incidence2.append(incidence2a[i] + incidence2b[i])

incmat1 = 1.0*numpy.matrix(incidence1)
incmat2 = 1.0*numpy.matrix(incidence2)

#  Getting list of Character Coincidence
#  - Count of Two Characters appearing in same scene

coinmat1 = incmat1*incmat1.getT()
coinmat2 = incmat2*incmat2.getT()

coinlist1 = []
coinlist2 = []

for i in range(1,len(coinmat1)):
        for j in range(0,i):
                temp = numpy.sqrt(numpy.power(coinmat1[i,j],2)*1.0/(coinmat1[i,i
                    ↪ ]*coinmat1[j,j]));
                if temp > min_weight:
                        coinlist1.append((i,j,temp));

for i in range(1,len(coinmat2)):
        for j in range(0,i):
                temp = numpy.sqrt(numpy.power(coinmat2[i,j],2)*1.0/(coinmat2[i,i
                    ↪ ]*coinmat2[j,j]));
                if temp > min_weight:
                        coinlist2.append((i,j,temp));

#  Setting Up The Graph
#  - Character lists to Populate Node Information
```

```
charlist1 = []
charlist2 = []

for i in range(0,len(chars1a)):
        charlist1.append((i,chars1a[i][0],chars1a[i][2]))

for i in range(0,len(chars2a)):
        charlist2.append((i,chars2a[i][0],chars2a[i][2]))

#  Writing the Graph GML for both Graphs.

outstring1 = "Creator␣\"Peter␣Trubey\"\ngraph\n[\n"

for i in range(0,len(charlist1)):
        outstring1 += ('␣␣␣node\n␣␣␣[\n␣␣␣␣␣␣␣id␣'+str(charlist1[i][0])+'\n␣␣␣␣␣␣
            ↪ desc␣\"'+charlist1[i][1]+'\"\n␣␣␣␣␣␣label␣'+charlist1[i][2]+'\n␣␣␣
            ↪ ]\n')

for i in range(0,len(coinlist1)):
        outstring1 += ('␣␣␣edge\n␣␣␣[\n␣␣␣␣␣␣␣source␣' + str(coinlist1[i][0]) + '
            ↪ \n␣␣␣␣␣␣␣target␣'+str(coinlist1[i][1]) + '\n␣␣␣␣␣␣␣value␣' + str(
            ↪ numpy.round(coinlist1[i][2],rd)) + '\n␣␣␣]\n')

outstring1 += ']'

outstring2 = "Creator␣\"Peter␣Trubey\"\ngraph\n[\n"

for i in range(0,len(charlist2)):
        outstring2 += ('␣␣␣node\n␣␣␣[\n␣␣␣␣␣␣␣id␣'+str(charlist2[i][0])+'\n␣␣␣␣␣␣
            ↪ desc␣\"'+charlist2[i][1]+'\"\n␣␣␣␣␣␣label␣'+charlist2[i][2]+'\n␣␣␣
            ↪ ]\n')

for i in range(0,len(coinlist2)):
        outstring2 += ('␣␣␣edge\n␣␣␣[\n␣␣␣␣␣␣␣source␣' + str(coinlist2[i][0]) + '
            ↪ \n␣␣␣␣␣␣␣target␣'+str(coinlist2[i][1]) + '\n␣␣␣␣␣␣␣value␣' + str(
            ↪ numpy.round(coinlist2[i][2],rd)) + '\n␣␣␣]\n')

outstring2 += ']'

f = open('MC1.gml','w')
f.write(outstring1)
f.close()

f = open('MC2.gml','w')
f.write(outstring2)
f.close()
```

A.3 Making the Graph Visuals

```
import networkx as nx
import math

# Importing Graphs
mc1 = nx.read_gml('mc1.gml',relabel=True)
mc2 = nx.read_gml('mc2.gml',relabel=True)

# Creating Character Groups
char_Dantes1 = dict((x,x) for x in [u'ED',u'MC',u'Wi',u'Si',u'Bu',u'Za'])
char_Dantes2 = dict((x,x) for x in [u'ED'])
char_Allies = dict((x,x) for x in [u'Fa',u'Bt',u'LV',u'Pp',u'Bp',u'Ha',u'Al',u'
    ↪ Ba',u'Ja'])
char_Morrel = dict((x,x) for x in [u'Mo',u'Ma',u'Ju',u'Em'])
char_Mondego = dict((x,x) for x in [u'MM',u'FM',u'AM'])
char_Danglars = dict((x,x) for x in [u'Da',u'He',u'Eu'])
char_Villefort = dict((x,x) for x in [u'Vi',u'Re',u'SM1',u'SM2',u'Va',u'NV',u'HV
    ↪ ',u'EV'])
char_Enemies = dict((x,x) for x in [u'Vi',u'FM',u'Da',u'GC'])

# Calculating Layouts
mc1_dist = 1.0/(math.sqrt(len(mc1.nodes())))
mc2_dist = 1.0/(math.sqrt(len(mc2.nodes())))

mc1_pos_wgt = nx.spring_layout(mc1,scale=1.25,k = .8*mc1_dist, weight='value')
mc2_pos_wgt = nx.spring_layout(mc2,scale=1.0,k = .8*mc2_dist,weight='value')

# Creating Graphs (separated Aliases)
nx.draw_networkx(mc1,mc1_pos_wgt,font_size=8,node_size=300,node_color='lightgray
    ↪ ')
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Dantes1, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Dantes1, node_color='green',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Allies, font_size=10,font_color='
    ↪ white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Allies, node_color='teal',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Morrel, font_size=10,font_color='
    ↪ white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Morrel, node_color='blue',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Villefort, font_size=10,
    ↪ font_color='white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Villefort, node_color='purple',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Danglars, font_size=10,font_color
```

```
    ↪ ='white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Danglars, node_color='brown',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Mondego, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Mondego, node_color='orange',
    ↪ node_size=300)
nx.draw_networkx_labels(mc1, mc1_pos_wgt, char_Enemies, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc1, mc1_pos_wgt, char_Enemies, node_color='red',
    ↪ node_size=300)

# Creating Graph (Collapsed Aliases)
nx.draw_networkx(mc2,mc2_pos_wgt,font_size=8,node_size=300,node_color='lightgray
    ↪ ')
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Dantes2, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Dantes2, node_color='green',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Allies, font_size=10,font_color='
    ↪ white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Allies, node_color='teal',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Morrel, font_size=10,font_color='
    ↪ white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Morrel, node_color='blue',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Villefort, font_size=10,
    ↪ font_color='white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Villefort, node_color='purple',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Danglars, font_size=10,font_color
    ↪ ='white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Danglars, node_color='brown',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Mondego, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Mondego, node_color='orange',
    ↪ node_size=300)
nx.draw_networkx_labels(mc2, mc2_pos_wgt, char_Enemies, font_size=10,font_color=
    ↪ 'white')
nx.draw_networkx_nodes(mc2, mc2_pos_wgt, char_Enemies, node_color='red',
    ↪ node_size=300)
```