Structural Balance of Harry Potter's character network

1 Introduction

What is structural balance? The principles underlying structural balance could be traced all the way back to 1940s, a theory from Heider in his work on social psychology dating (Heider, 1946). Basically, the author proposed that if one individual, say A, has a positive relationship with another individual B, and B has a positive tie to an impersonal entity C (could be a person or an object), then the overall relationship between A, B and C is said to be balanced if A also likes C; whereas it is imbalanced if A dislikes C. It is also claimed that balance in relationship is an equilibrium state which individual strives to achieve and maintain. A decade later, this theory was then generalized and extended to graphs by Cartwright and Harary (1956), in which triangles in network are balanced only when one or all edge(s) are positive signed.

Patterns of social interactions within a network can be explained by different principals at different levels, ranging from node, dyadic, triadic, subgroup to graph levels (Dinh, Rezapour, Jiang, & Diesner, 2020). A triad is any group of three nodes with directed links, and it could be classified into 16 classes (Holland & Leinhardt, 1978) which is commonly known as "triad census" (see Figure 1-1). In this study, structural balance of Harry Potter's (HP's) character network was analysed at a triadic level, as suggested by Dinh et al. (2020), that any other levels of analysis fail to explain the patterns of tie formations between three nodes and their associated links. Apart from being a relatively "fresh" topic for the network itself, HP's character network was chosen because character networks may possess certain properties observed in real-world networks (Labatut & Bost, 2019).

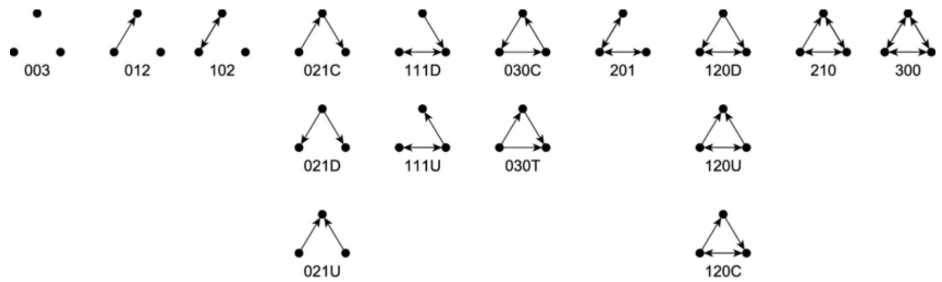


Figure 1-1 Triadic census and their names, image adapted from Cugmas, Ferligoj, and Žiberna (2017).

2 Motivation and Problem Statement

It is generally known that determining if a network is balanced or not is easy, for example to divide the graph into two nodes subsets, it is balanced only if intra-group edges are all positive and inter-group edges are all negative. Another way would be using generalisation to check if a signed graph contains cycle with an odd number of negative edges. If yes, the graph is unbalanced and vice versa. However, measuring how close a graph is to be balanced is not easy. Using one of the existing methods defined by network scholars, this study approached the problem "How balanced is HP's character network?" and a further question "How will it be different to also consider direction of ties in computing the balance score?".

3 Methodology

3.1 Experiments

Two experiments were carried out in this study. First, both the direction of ties and sign consistency of triads are considered in calculating the "balancedness" of the network, and the second experiment was about only sign consistency is considered in computation.

In order words, for the latter, the original network was transformed into an undirected network. The method of flattening used was the one suggested by Diesner and Evans (2015). When there is at least one direction of edge found between two nodes, there will be an undirected tie link between them, as long as the edge(s) are in the same sign, if else, the nodes will not be connected (see Figure 3-1).

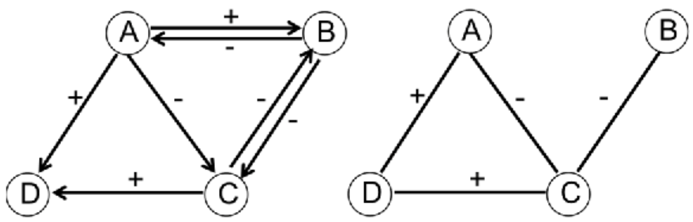


Figure 3-1 shows how the signed and directed network is transformed into an undirected network. (Diesner & Evans, 2015)

3.2 Algorithms

In the first experiment, an algorithm proposed by Dinh et al. (2020) to calculate balance by considering triads with only transitive semi-cycles was used to calculate the balance score of the directed graph. A semi-cycle is a set of three directed edges that starts from a vertex, follows the direction of edges, and might (cyclic) or might not (non-cyclic) return to the same vertex. Based on Heider (1946)'s theory, transitivity is a necessary condition for stability and balance. Since cyclic semi-cycle is proven to be "intransitive and contained limited information on the process of influence among relationships", Dinh et al. (2020) considered only four classes of triad with a total of 11 permutations of semi-cycle in the calculation (see Figure 3-2 left).

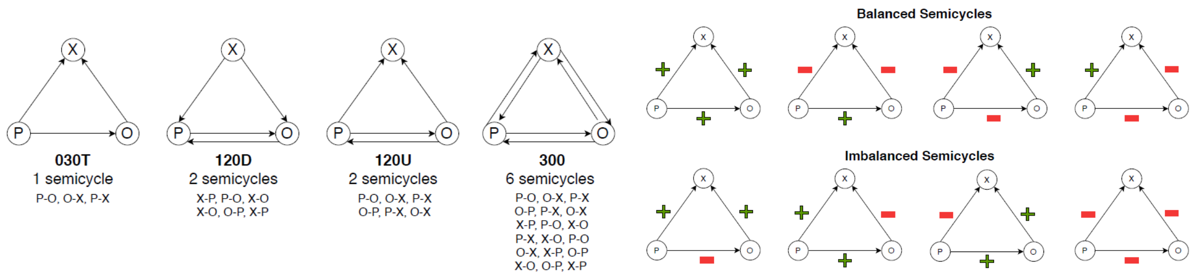


Figure 3-2 (Left) Transitive and balanced triads, with their semi-cycles. (Right) Balanced and imbalanced semi-cycles. Image adapted from Dinh et al. (2020).

A step-by-step calculation of balance score for the network is explained in Figure 3-3. A balanced semi-cycle is one with even number of negative signs (see Figure 3-2 right). A triad is *completely balanced* if all its semi-cycles are balanced, *partially balanced* if at least one of them is balanced, and *completely imbalanced* if all of them are imbalanced. After calculating balance ratio for each triad of a type, the balance ratio for the set of all transitive triads of same type is obtained. Finally, the overall balance ratio, $B_{Avg(G)}$ is calculated by averaging the balance ratio of all triad classes across a network. A signed digraph is claimed to be balanced if all its triads are balanced.

For each triad class T_i, i = 1,2,3 & 4:

For each triad set, T_j:

For each semi-cycle:

Sign of semi-cycle = product of sign of three edges

Let S_j^+ = number of positive semi-cycle

S_j = total number of semi-cycles

Balance ratio of T_j, $B_{Tj} = S_j^+/S_j$

Let N_{Ti} = total number of T_j with $B_{Tj} \neq 0$

$T^{(i)}$ = total number of T_j

Balance ratio of T_i, $B_{Ti} = N_{Ti}/T^{(i)}$

Average balance ratio of the graph, $B_{Avg(G)} = \sum B_{Ti}/4$

Figure 3-3 shows step-by-step in algorithm used to compute overall balance ratio of a network.

For second experiment, the same steps of computation were applied, except that no triads but triangles and that the "balancedness" is evaluated on triangles instead of semi-cycles.

3.3 Dataset

The dataset consists of 65 characters appeared in all HP book series, directed and with sign (indicating the type of relationship, '+' means like and '-' means dislike) as the attribute of the ties between two characters. It is sourced from website data.world (Garg, 2017).

4 Results

4.1 Descriptive measures of original network

The dataset was analysed using python and network libraries such as Networkx (see Appendix). Some descriptive measures were analysed and tabulated in Table 4-1. It is found that the 65 nodes and 456 edges network have an average degree of 14, which means that in average each character will link to other 14 characters, whether they are incoming or outcoming links. The global clustering or the transitivity of network is 0.38, which gives a hint on the high proportional of closed triads vs open triads found in the network. High clustering coefficient at a local level shows the high probability of a character's neighbours forming connected pairs. The network density is at 0.11, showing low ratio of observed to possible edge. According to Labatut and Bost (2019), the diameter of character network reflects the compactness of story. In this case, it is believed that a small average shortest distance is observed as the character network is constructed from all series.

The network was visualised using Gephi, and the result is shown in Figure 4-1.

Table 4-1 Summary of descriptive measures of HP's network

Descriptive network measures	No. of nodes	No. of edges	Average degree	Transitivity	Local clustering	Density	Average shortest path length
HP network	65	456	14.0308	0.3825	0.4151	0.1096	2.375

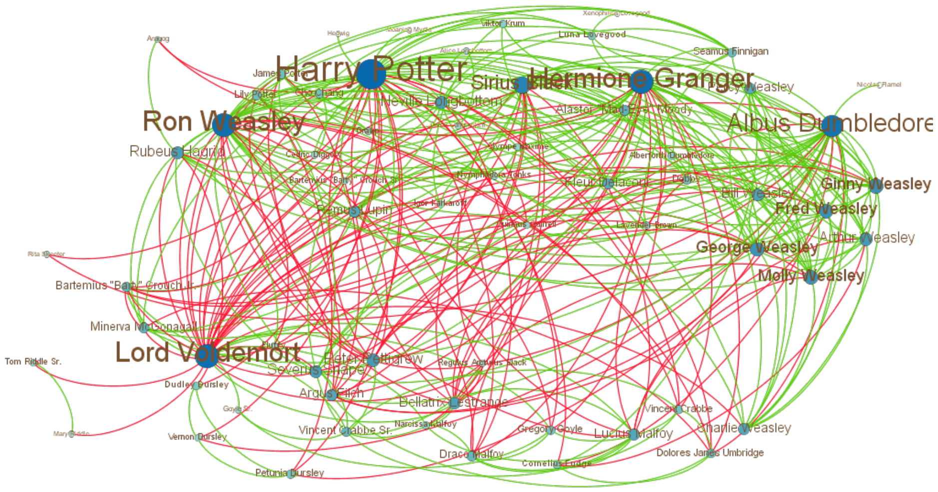


Figure 4-1 shows HP's interaction network, where colour of edge between two nodes represents sign of relationship one character has unto another: green indicates positive relationship, for example to like or to value while red indicates the contrast.

4.2 Considering ties direction and sign consistency

Table 4-2 Summary of balance ratios of network in first experiment

Transitive triads type	Count	Completely Balanced	Partially Balanced	Completely Imbalanced	Balance Ratio, B_T(%)
030T	211	190	0	21	90.050
120D	131	115	0	16	87.790
120U	167	158	2	7	95.810
300	84	82	1	1	98.810
Total	593	545	3	45	B_Avg(G)= 93.11

The result of each iteration explained in Figure 3-3 is summarized in Table 4-2. It is observed that when considering semi-cycles of triads in computing balance score, HP's character network is 93.11% balanced. This could be explained by the fact that most transitive triads in the network are completely balanced, small part completely imbalanced and very few (3 out of 593) partially imbalanced. Taking a detailed look to the semi-cycles level (see Table 4-3), the prevalence of two balanced semi-cycles is seen. This again evidences the closeness of the network to complete balance.

Table 4-3 Counts of semi-cycle by type in the first experiment

Semicycle type	Counts	Ratio-Total
+++	1117	0.850
++-	17	0.010
+--	116	0.090
---	61	0.050
Total	1311	1

4.3 Considering only sign consistency

Table 4-4 and Table 4-5 show the results of balance ratio computation and counts of triangle for undirected network of HP's character. Since direction of ties is neglected, the calculation based only on triangles. There are total of 1037 triangles in the network, and 859 of them are completed balanced – 499 are with all positive-signed edges and 360 contains one positive edge. With this, the graph is 82.84% balanced.

Table 4-4 Balance ratio of network in second experiment

	Count	Completely Balanced	Partially Balanced	Completely Imbalanced	Balance Ratio, B_avg(G)(%)
flattened undirected network	1037	859	0	178	82.840

Table 4-5 Counts of triangle by type in second experiment.

Triangle type	Counts	Ratio-Total
+++	499	0.481
++-	140	0.135
+--	360	0.347
---	38	0.037
Total	1037	1

5 Discussions

One of the ways to compare two result values is to calculate the percentage deviation between them as shown in Equation 1.

$$\% \, deviation = \frac{B_{avg(dgraph)} - B_{avg(ugraph)}}{B_{avg(dgraph)}} \, x \, 100 \qquad (1)$$

where $dgraph$ denotes a directed network used in first experiment and $ugraph$ denotes an undirected network in second experiment.

$$\% \, deviation = \frac{93.11 - 82.84}{93.11} \, x \, 100 = 11.03\%$$

Hence, it is observed that the balance ratio of without considering link direction deviates 11% from the one considering, which is more than 10% margin if the first experiment value is taken as a null value.

There are a few points that could be made out from this. First, although considering direction seems to add more restrictions (for example only 4 types of triad are investigated) it actually considers and includes every combination of node relationship (1311 semi-cycles) in calculation. Moreover, the deletion of contrast link among two characters for undirected network decreases the clustering effect in the network, aka less set of triangles are found. A good instance is the relationship between Ron, Hagrid and Aragog. In the first experiment, this triad set is categorized as partially imbalanced since Ron dislikes Aragog but Aragog likes Ron. However, this relationship will not be in consideration if the network is flattened to undirected one (as their ties' signs are opposite). It also means that this is no longer a triangle that is concerned, since no edge formed between Ron and Aragog in the second experiment. Thus, it is said that the balance ratio of network when direction of edges was taken into account, along with sign consistency, reflects more accurately than the one without.

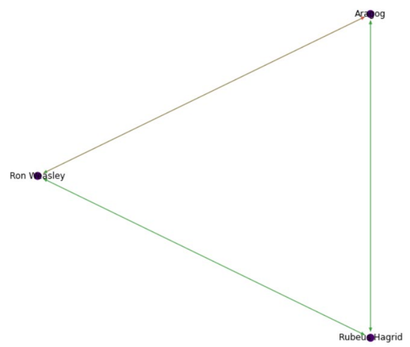


Figure 5-1 One example of triads (class 300) with contrast links between two nodes.

6 Conclusion

HP's character network is closer to balance (93%) when both direction of edges and sign consistency are considered in computation than only sign consistency is included (82%). This shows low degree of violation of transitivity and structural balance in the network.

One of the limitations of this study is lack of description of the dataset used, for example, why only 65 characters included since there are over 700 of characters in the series. This might raise an error of insufficient sample size for statistical measurement and a biased result. A temporal structural balance analysis across 7 books of HP is suggested for future study.

References

Cartwright, D., & Harary, F. (1956). Structural balance: a generalization of Heider's theory. *Psychological review, 63 5*, 277-293.

Cugmas, M., Ferligoj, A., & Žiberna, A. (2017). Generating global network structures by triad types. *PLoS ONE, 13*. doi:10.1371/journal.pone.0197514

Diesner, J., & Evans, C. S. (2015). Little bad concerns: Using sentiment analysis to assess structural balance in communication networks. *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 342-348.

Dinh, L., Rezapour, R., Jiang, L., & Diesner, J. (2020). Structural balance in signed digraphs: considering transitivity to measure balance in graphs constructed by using different link signing methods. *ArXiv, abs/2006.02565*.

Garg, H. K. (2017). Harry Potter Universe. Retrieved from https://data.world/harishkgarg/harry-potter-universe

Heider, F. (1946). Attitudes and Cognitive Organization. *The Journal of Psychology, 21*(1), 107-112. doi:10.1080/00223980.1946.9917275

Holland, P. W., & Leinhardt, S. (1978). An Omnibus Test for Social Structure Using Triads. *Sociological Methods & Research, 7*(2), 227-256. doi:10.1177/004912417800700207

Labatut, V., & Bost, X. (2019). Extraction and Analysis of Fictional Character Networks: A Survey. *ACM Comput. Surv., 52*(5), Article 89. doi:10.1145/3344548

Appendix

COMP5313project21

May 27, 2021

1 Import Libraries

[1]:
```
%matplotlib inline
import sys
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import math as m
import numpy as np
import pandas as pd
import os
import networkx as nx
import collections
from statistics import mean, stdev
import itertools

from collections import defaultdict
import operator

## For Hierarchical Clustering
from scipy.cluster import hierarchy
from scipy.spatial import distance

## For Community Detection (Louvain Method)
import community
```

2 Load data and create network

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[3]:
```
characters_path = '/content/drive/MyDrive/1USyd/3rd_sem/COMP5313/assessments/
↪Assignment2/harishkgarg-harry-potter-universe/characters.csv'
```

```python
df_characters = pd.read_csv(characters_path)
print(df_characters.columns)
print(df_characters.shape)
```

```
Index(['id', 'name', 'bio'], dtype='object')
(65, 3)
```

[4]:
```python
df_characters.head()
```

[4]:
```
   id  ...                                                bio
0   0  ...  Brother of Sirius. Used to be a Death Eater bu...
1   1  ...  Best friend of James Potter and godfather of H...
2   2  ...  Killed by a werewolf. She was a gryffindor stu...
3   3  ...  Ravenclaw student who dated Cedric Diggory and...
4   4  ...  Father of Crabbe and death-eater who escaped A...

[5 rows x 3 columns]
```

[5]:
```python
name_mapping ={}
for row in df_characters.itertuples():
  index = row[1]
  name = row[2]
  name_mapping[index] = name
```

[6]:
```python
name_mapping
```

[6]:
```
{0: 'Regulus Arcturus Black',
 1: 'Sirius Black',
 2: 'Lavender Brown',
 3: 'Cho Chang',
 4: 'Vincent Crabbe Sr.',
 5: 'Vincent Crabbe',
 6: 'Bartemius "Barty" Crouch Sr.',
 7: 'Bartemius "Barty" Crouch Jr.',
 8: 'Fleur Delacour',
 9: 'Cedric Diggory',
 10: 'Alberforth Dumbledore',
 11: 'Albus Dumbledore',
 12: 'Dudley Dursley',
 13: 'Petunia Dursley',
 14: 'Vernon Dursley',
 15: 'Argus Filch',
 16: 'Seamus Finnigan',
 17: 'Nicolas Flamel',
 18: 'Cornelius Fudge',
 19: 'Goyle Sr.',
 20: 'Gregory Goyle',
 21: 'Hermione Granger',
 22: 'Rubeus Hagrid',
```

```
23: 'Igor Karkaroff',
24: 'Viktor Krum',
25: 'Bellatrix Lestrange',
26: 'Alice Longbottom',
27: 'Frank Longbottom',
28: 'Neville Longbottom',
29: 'Luna Lovegood',
30: 'Xenophilius Lovegood',
31: 'Remus Lupin',
32: 'Draco Malfoy',
33: 'Lucius Malfoy',
34: 'Narcissa Malfoy',
35: 'Olympe Maxime',
36: 'Minerva McGonagall',
37: 'Alastor "Mad-Eye" Moody',
38: 'Peter Pettigrew',
39: 'Harry Potter',
40: 'James Potter',
41: 'Lily Potter',
42: 'Quirinus Quirrell',
43: 'Tom Riddle Sr.',
44: 'Mary Riddle',
45: 'Lord Voldemort',
46: 'Rita Skeeter',
47: 'Severus Snape',
48: 'Nymphadora Tonks',
49: 'Dolores Janes Umbridge',
50: 'Arthur Weasley',
51: 'Bill Weasley',
52: 'Charlie Weasley',
53: 'Fred Weasley',
54: 'George Weasley',
55: 'Ginny Weasley',
56: 'Molly Weasley',
57: 'Percy Weasley',
58: 'Ron Weasley',
59: 'Dobby',
60: 'Fluffy',
61: 'Hedwig',
62: 'Moaning Myrtle',
63: 'Aragog',
64: 'Grawp'}
```

[7]:
```
relations_path = '/content/drive/MyDrive/1USyd/3rd_sem/COMP5313/assessments/
 ↪Assignment2/harishkgarg-harry-potter-universe/relations.csv'
df_relations = pd.read_csv(relations_path)
print(df_relations.columns)
```

```python
print(df_relations.shape)
df_relations.head()
```

```
Index(['source', 'target', 'type'], dtype='object')
(513, 3)
```

[7]:

	source	target	type
0	0	1	-
1	0	25	-
2	0	45	-
3	1	0	-
4	1	11	+

[8]:
```python
df_relations['sign'] = df_relations['type'].apply(lambda x: 1 if x == '+' else␣
↪-1)
```

[9]:
```python
df_relations.head()
```

[9]:

	source	target	type	sign
0	0	1	-	-1
1	0	25	-	-1
2	0	45	-	-1
3	1	0	-	-1
4	1	11	+	1

[10]:
```python
G0 = nx.DiGraph()
for row in df_relations.itertuples():
  source = row[1]
  target = row[2]
  sign_ = row[4]
  G0.add_edge(source, target, sign = sign_)
```

2.1 Descriptive network measures

[11]:
```python
nx.is_strongly_connected(G0)
```

[11]: False

[12]:
```python
nx.is_weakly_connected(G0)
```

[12]: True

[13]:
```python
largest = max(nx.strongly_connected_components(G0), key=len)
```

[14]:
```python
len(largest)
```

[14]: 59

[15]:
```python
weak_lcc = max(nx.weakly_connected_components(G0), key=len)
```

[16]:
```python
len(weak_lcc)
```

[16]: 65

[17]:
```python
list(G0.edges())[:3]
```

[17]:
```
[(0, 1), (0, 25), (0, 45)]
```

[18]:
```python
G0[0][1]['sign']
```

[18]:
```
-1
```

[19]:
```python
G0.number_of_nodes()
```

[19]:
```
65
```

[20]:
```python
G0.number_of_edges()
```

[20]:
```
456
```

[21]:
```python
nx.transitivity(G0)
```

[21]:
```
0.3825301204819277
```

[22]:
```python
def z_measures(G):
    '''Function to calc <z> and sigma_z'''
    N = G.number_of_nodes()
    zij= dict(nx.degree(G))
    zi = np.array([zij[k] for k in zij])
    average_z = np.sum(zi)/N
    av_zsquared = np.sum(np.square(zi))/N
    sd_z = np.sqrt(av_zsquared - np.square(average_z))

    return average_z, sd_z
```

[23]:
```python
z_measures(G0)
```

[23]:
```
(14.03076923076923, 12.608872615704122)
```

[24]:
```python
nx.density(G0)
```

[24]:
```
0.10961538461538461
```

[25]:
```python
nx.shortest_paths.average_shortest_path_length(G0)
```

[25]:
```
2.375
```

[26]:
```python
nx.average_clustering(G0)
```

[26]:
```
0.41509401336828167
```

[27]:
```python
cij = dict(nx.clustering(G0))
ci = [cij[k] for k in cij]
average_ci = (np.sum(ci)/G0.number_of_nodes())
```

[28]:
```python
average_ci
```

[28]:
```
0.4150940133682817
```

[29]:
```python
nx.number_strongly_connected_components(G0)
```

[29]:
```
6
```

[ ]:
```
list(nx.strongly_connected_components(G0))
```

[31]:
```
nx.number_weakly_connected_components(G0)
```

[31]:
```
1
```

[ ]:
```
list(nx.weakly_connected_components(G0))
```

2.2 draw function

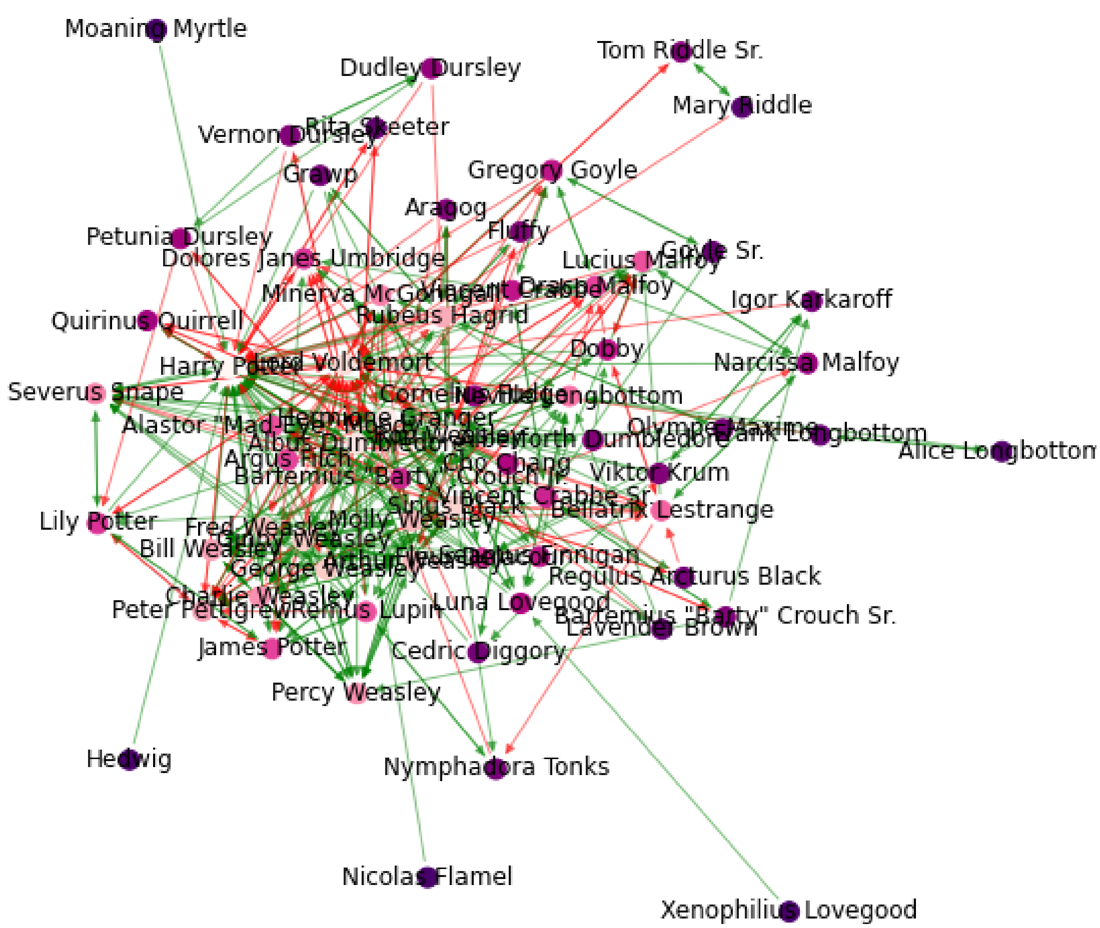
[33]:
```
edges = list(G0.edges())
colors = []
for u,v in edges:
  if G0[u][v]['sign'] == 1:
    colors.append('g')
  else:
    colors.append('r')
```

[34]:
```
from itertools import count
def draw_network(G, with_labels = True, graph_layout = 'shell'):
    '''Draw the graph with node's color varies with its degree'''
    # get unique groups
    groups = set([G.degree()[node] for node in list(G.nodes())])
    mapping = dict(zip(sorted(groups),count()))
    nodecolors = [mapping[G.degree()[node]] for node in list(G.nodes())]
    nodedegree = dict(G.degree)

    # drawing nodes and edges separately so we can capture collection for␣
 ↪colobar
    plt.figure(figsize=(10, 10))

    if graph_layout == 'spring':
      pos = nx.spring_layout(G)
    elif graph_layout == 'spectral':
      pos = nx.spectral_layout(G)
    elif graph_layout == 'random':
      pos = nx.random_layout(G)
    else:
      pos = nx.shell_layout(G)

    edges = list(G.edges())
    colors = []
    for u,v in edges:
      if G[u][v]['sign'] == 1:
        colors.append('g')
      else:
        colors.append('r')
```

```
    node_size = 100
    # node_size = [v * 100 for v in nodedegree.values()]
    ec = nx.draw_networkx_edges(G, pos, edge_color = colors, alpha=0.5)
    nc = nx.draw_networkx_nodes(G, pos, nodelist = G.nodes(), node_color = nodecolors,\
        node_size =node_size, cmap=plt.cm.RdPu_r)

    if with_labels == True:
        labels = nx.draw_networkx_labels(G,pos)

    # plt.colorbar(nc)
    plt.axis('off')
    plt.show()
```

[35]:
```
G_hp = nx.relabel_nodes(G0, name_mapping)
```

[36]:
```
draw_network(G_hp, graph_layout='spring')
```



3 Structural balance

3.1 Digraph

[37]:
```
G0.number_of_edges()
```

[37]:
```
456
```

3.1.1 triads

[38]:
```
nx.triadic_census(G0)
```

[38]:
```
{'003': 28021,
 '012': 6749,
 '021C': 288,
 '021D': 237,
 '021U': 1107,
 '030C': 5,
 '030T': 211,
 '102': 4599,
 '111D': 942,
 '111U': 584,
 '120C': 36,
 '120D': 131,
 '120U': 167,
 '201': 333,
 '210': 186,
 '300': 84}
```

[39]:
```
# https://stackoverflow.com/questions/55339231/
↪get-the-list-of-triad-nodes-who-fall-under-the-category-of-individual-triadic/
↪55375317#55375317
def _tricode(G, v, u, w):
    """Returns the integer code of the given triad.

    This is some fancy magic that comes from Batagelj and Mrvar's paper. It
    treats each edge joining a pair of `v`, `u`, and `w` as a bit in
    the binary representation of an integer.

    """
    combos = ((v, u, 1), (u, v, 2), (v, w, 4), (w, v, 8), (u, w, 16),
              (w, u, 32))
    return sum(x for u, v, x in combos if v in G[u])
```

[40]:
```python
def triads_dict(G):
    TRICODES = (1, 2, 2, 3, 2, 4, 6, 8, 2, 6, 5, 7, 3, 8, 7, 11, 2, 6, 4, 8, 5, 9,
                9, 13, 6, 10, 9, 14, 7, 14, 12, 15, 2, 5, 6, 7, 6, 9, 10, 14, 4,␣
↪9,
                9, 12, 8, 13, 14, 15, 3, 7, 8, 11, 7, 12, 14, 15, 8, 14, 13, 15,
                11, 15, 15, 16)

    TRIAD_NAMES = ('003', '012', '102', '021D', '021U', '021C', '111D', '111U',
                   '030T', '030C', '201', '120D', '120U', '120C', '210', '300')

    TRICODE_TO_NAME = {i: TRIAD_NAMES[code - 1] for i, code in␣
↪enumerate(TRICODES)}

    triad_nodes = {name: set([]) for name in TRIAD_NAMES}
    m = {v: i for i, v in enumerate(G)}
    for v in G:
        vnbrs = set(G.pred[v]) | set(G.succ[v])
        for u in vnbrs:
            if m[u] > m[v]:
                unbrs = set(G.pred[u]) | set(G.succ[u])
                neighbors = (vnbrs | unbrs) - {u, v}
                not_neighbors = set(G.nodes()) - neighbors - {u, v}
                # Find dyadic triads
                for w in not_neighbors:
                    if v in G[u] and u in G[v]:
                        triad_nodes['102'].add(tuple(sorted([u, v, w])))
                    else:
                        triad_nodes['012'].add(tuple(sorted([u, v, w])))
                for w in neighbors:
                    if m[u] < m[w] or (m[v] < m[w] < m[u] and
                                       v not in G.pred[w] and
                                       v not in G.succ[w]):
                        code = _tricode(G, v, u, w)
                        triad_nodes[TRICODE_TO_NAME[code]].add(
                            tuple(sorted([u, v, w])))

    # find null triads
    all_tuples = set()
    for s in triad_nodes.values():
        all_tuples = all_tuples.union(s)
    triad_nodes['003'] = set(itertools.combinations(G.nodes(), 3)).
↪difference(all_tuples)

    return triad_nodes
```

[41]:
```python
triads_hp = triads_dict(G0)
```

[42]:
```
four_triads = {}
interested_triads = ['030T','120D','120U','300']
for (triad,ls) in triads_hp.items():
  if triad in interested_triads:
    four_triads[triad] = ls
```

[43]:
```
len(four_triads['300'])
```

[43]:
```
84
```

[44]:
```
list(four_triads['300'])[0]
```

[44]:
```
(53, 54, 57)
```

[45]:
```
list(list(four_triads['300'])[0])
```

[45]:
```
[53, 54, 57]
```

[46]:
```
list(list(four_triads['120D'])[0])
```

[46]:
```
[11, 53, 57]
```

[47]:
```
H = G0.subgraph(list(list(four_triads['120D'])[0]))
sorted(list(H.edges()))
```

[47]:
```
[(11, 53), (11, 57), (53, 57), (57, 53)]
```

[48]:
```
u,v = sorted(list(H.edges()))[0]
```

3.1.2 030T

[49]:
```
def semicycles_statistic(semicycle_signs):
  # '+ + +'
  all_positive = 0
  # '+ + -'
  one_negative = 0
  # '+ - -'
  one_positive = 0
  # '- - -'
  all_negative = 0

  neg_count = semicycle_signs.count(-1)
  if (neg_count%2)==0:
    # Even number of -ve means positive
    if neg_count == 0:
      all_positive += 1
    else:
      one_positive += 1
  else:
    # Odd number of -ve means negative
    if neg_count == 1:
      one_negative += 1
```

```
    else:
      all_negative += 1

  return all_positive, one_negative, one_positive, all_negative
```

[ ]:
```
four_triads['030T']
```

[51]:
```
def triad_balance_count1(semicycles_sign):
  '''For triad type 030T and undirected closed triads only'''
  complete_balance = semicycles_sign.count(1)
  partial_balance = 0
  imbalanced = semicycles_sign.count(-1)
  balance_ratio = (complete_balance / len(semicycles_sign))*100
  return complete_balance, partial_balance, imbalanced, round(balance_ratio,2)
```

[52]:
```
def draw_triads(subgraph):
  interested_nodes = [11,21,39,45,58]
  triad_nodes = list(subgraph.nodes())
  for i in triad_nodes:
    if i in interested_nodes:
      subgraph = nx.relabel_nodes(subgraph, name_mapping)
      d = draw_network(subgraph)
      return d
```

[ ]:
```
all_positive1 = 0
one_negative1 = 0
one_positive1 = 0
all_negative1 = 0

semicycles_sign1 = []

for triad in list(four_triads['030T']):
  nodes = list(triad)
  H = G0.subgraph(nodes)

  # draw_triads(H)

  sign_list1=[]
  for u,v,data in H.edges(data=True):
    sign_list1.append(data['sign'])

  neg_count1 = sign_list1.count(-1)
  if (neg_count1%2)==0:
    # Even number of -ve means positive
    # draw_triads(H)
    semicycles_sign1.append(1)
  else:
    # Odd number of -ve means negative
    draw_triads(H)
```

```
    semicycles_sign1.append(-1)

  all_pos1, one_neg1, one_pos1, all_neg1 = semicycles_statistic(sign_list1)
  all_positive1 += all_pos1
  one_negative1 += one_neg1
  one_positive1 += one_pos1
  all_negative1 += all_neg1


complete_balance1, partial_balance1, imbalanced1, balance_ratio1 =␣
↪triad_balance_count1(semicycles_sign1)
```

[54]:
```
0%2
```

[54]: 0

[55]:
```
(complete_balance1 + partial_balance1 + imbalanced1) ==␣
↪len(list(four_triads['030T']))
```

[55]: True

3.1.3 120D

[56]:
```
def triad_balance_count(semicycles_sign):
  '''For triad types 120D 120U 300'''
  complete_balance = 0
  partial_balance = 0
  imbalanced = 0

  positive_semicycles = semicycles_sign.count(1)
  negative_semicycles = semicycles_sign.count(-1)

  if positive_semicycles == len(semicycles_sign):
    complete_balance += 1
  elif negative_semicycles == len(semicycles_sign):
    imbalanced += 1
  else:
    partial_balance += 1

  # balanced ratio
  b_triad = positive_semicycles / len(semicycles_sign)


  return complete_balance, partial_balance, imbalanced, b_triad
```

[57]:
```
def balance_ratio(b_triad_all):
  total_triads = len(b_triad_all)
  negative_triads = b_triad_all.count(0)
  positive_triads = total_triads - negative_triads
```

```
  balance_ratio = (positive_triads / total_triads)*100
  return round(balance_ratio, 2)
```

[ ]:
```
complete_balance2 = 0
partial_balance2 = 0
imbalanced2 = 0
B_triad2 = []

all_positive2 = 0
one_negative2 = 0
one_positive2 = 0
all_negative2 = 0

for triad in list(four_triads['120D']):
  nodes = list(triad)
  H = G0.subgraph(nodes)
  node = sorted(H.degree, key=lambda x: x[1], reverse=False)
  # let first node with degree 2 = X (as in paper)
  X,X_degree = node[0]
  # remaining nodes = P or O (as in paper)
  P,P_degree = node[1]
  O,O_degree = node[2]

  semicycle_list = [[(X,P),(P,O),(X,O)],\
                    [(X,O),(O,P),(X,P)]]

  semicycles_sign2 = []
  for semicycle in semicycle_list:
    sign_list2=[]
    for edge in semicycle:
      u, v = edge
      sign_list2.append(H[u][v]['sign'])

    neg_count2 = sign_list2.count(-1)
    if (neg_count2%2)==0:
      semicycles_sign2.append(1)
    else:
      draw_triads(H)
      semicycles_sign2.append(-1)

    all_pos2, one_neg2, one_pos2, all_neg2 = semicycles_statistic(sign_list2)
    all_positive2 += all_pos2
    one_negative2 += one_neg2
    one_positive2 += one_pos2
    all_negative2 += all_neg2
```

```
  complete_balance, partial_balance, imbalanced , b_triad =␣
↪triad_balance_count(semicycles_sign2)
  complete_balance2 += complete_balance
  partial_balance2 += partial_balance
  imbalanced2 += imbalanced
  B_triad2.append(b_triad)

balance_ratio2 = balance_ratio(B_triad2)
```

[59]:
```
(complete_balance2 + partial_balance2 + imbalanced2) ==␣
↪len(list(four_triads['120D']))
```

[59]: True

3.1.4 120U

[ ]:
```
complete_balance3 = 0
partial_balance3 = 0
imbalanced3 = 0
B_triad3 = []

all_positive3 = 0
one_negative3 = 0
one_positive3 = 0
all_negative3 = 0

for triad in list(four_triads['120U']):
  nodes = list(triad)
  H = GO.subgraph(nodes)
  node = sorted(H.degree, key=lambda x: x[1], reverse=False)
  # let first node with degree 2 = X (as in paper)
  X,X_degree = node[0]
  # remaining nodes = P or O (as in paper)
  P,P_degree = node[1]
  O,O_degree = node[2]

  semicycle_list = [[(P,O),(O,X),(P,X)],\
                    [(O,P),(P,X),(O,X)]]

  semicycles_sign3 = []
  for semicycle in semicycle_list:
    sign_list3=[]
    for edge in semicycle:
      u, v = edge
      sign_list3.append(H[u][v]['sign'])

    neg_count3 = sign_list3.count(-1)
    if (neg_count3%2)==0:
```

```
      semicycles_sign3.append(1)
    else:
      semicycles_sign3.append(-1)
      draw_triads(H)

    all_pos3, one_neg3, one_pos3, all_neg3 = semicycles_statistic(sign_list3)
    all_positive3 += all_pos3
    one_negative3 += one_neg3
    one_positive3 += one_pos3
    all_negative3 += all_neg3

  complete_balance, partial_balance, imbalanced, b_triad =␣
 ↪triad_balance_count(semicycles_sign3)
  complete_balance3 += complete_balance
  partial_balance3 += partial_balance
  imbalanced3 += imbalanced
  B_triad3.append(b_triad)

balance_ratio3 = balance_ratio(B_triad3)
```

[61]:
```
(complete_balance3 + partial_balance3 + imbalanced3) ==␣
 ↪len(list(four_triads['120U']))
```

[61]: True

3.1.5 300

[62]:
```
H = G0.subgraph(list(list(four_triads['300'])[0]))
u,v = sorted(list(H.edges()))[0]
```

[63]:
```
H[u][v]['sign']
```

[63]: 1

[ ]:
```
complete_balance4 = 0
partial_balance4 = 0
imbalanced4 = 0
B_triad4 = []

all_positive4 = 0
one_negative4 = 0
one_positive4 = 0
all_negative4 = 0

for triad in list(four_triads['300']):
  nodes = list(triad)
  H = G0.subgraph(nodes)

  # since all nodes have same degree
```

```
  # it doesnt matter which node is X,P,O
  X = nodes[0]
  P = nodes[1]
  O = nodes[2]

  # semicycles as shown in paper
  semicycle_list = [[(X,P),(P,O),(X,O)],\
                    [(X,O),(O,P),(X,P)],\
                    [(P,O),(O,X),(P,X)],\
                    [(O,P),(P,X),(O,X)],\
                    [(P,X),(X,O),(P,O)],\
                    [(O,X),(X,P),(O,P)]
                    ]

  semicycles_sign4 = []
  for semicycle in semicycle_list:
    sign_list4=[]
    for edge in semicycle:
      u, v = edge
      sign_list4.append(H[u][v]['sign'])

    neg_count4 = sign_list4.count(-1)
    if (neg_count4%2)==0:
      semicycles_sign4.append(1)
    else:
      semicycles_sign4.append(-1)
      draw_triads(H)

    all_pos4, one_neg4, one_pos4, all_neg4 = semicycles_statistic(sign_list4)
    all_positive4 += all_pos4
    one_negative4 += one_neg4
    one_positive4 += one_pos4
    all_negative4 += all_neg4

  complete_balance, partial_balance, imbalanced, b_triad =␣
↪triad_balance_count(semicycles_sign4)
  complete_balance4 += complete_balance
  partial_balance4 += partial_balance
  imbalanced4 += imbalanced
  B_triad4.append(b_triad)

balance_ratio4 = balance_ratio(B_triad4)
```

[65]:
```
(complete_balance4 + partial_balance4 + imbalanced4) ==␣
↪len(list(four_triads['300']))
```

[65]:
```
True
```

3.1.6 Summary

[66]:
```
# For summary table use
count_triads = []
for i in interested_triads:
  triad_quantity = len(list(four_triads[i]))
  count_triads.append(triad_quantity)

count_sum = [sum(count_triads)]
count_result = count_triads + count_sum

compb = [complete_balance1, complete_balance2, complete_balance3,␣
 ↪complete_balance4]
cb_sum = [sum(compb)]
compb_result = compb + cb_sum

partb = [partial_balance1, partial_balance2, partial_balance3, partial_balance4]
pb_sum = [sum(partb)]
partb_result = partb + pb_sum

imb = [imbalanced1, imbalanced2, imbalanced3, imbalanced4]
imb_sum = [sum(imb)]
imb_result = imb + imb_sum

b_ratio = [balance_ratio1, balance_ratio2, balance_ratio3, balance_ratio4]
br_avg = [round(sum(b_ratio)/len(b_ratio),2)]
br_result = b_ratio + br_avg
```

[67]:
```
!pip install texttable
```

```
Collecting texttable
  Downloading https://files.pythonhosted.org/packages/06/f5/46201c428aebe0eecfa8
3df66bf3e6caa29659dbac5a56ddfd83cae0d4a4/texttable-1.6.3-py2.py3-none-any.whl
Installing collected packages: texttable
Successfully installed texttable-1.6.3
```

[68]:
```
from texttable import Texttable
t = Texttable()
row_header = ['030T','120D','120U','300','Total']
x = [[]]
# t.add_rows([['Transitive triads type', 'Count','Completely Balanced',\
#              'Partially Balanced','Completely Imbalanced', 'Balance Ratio,␣
 ↪B_T(%)']])
for i in range(len(row_header)):
  if i == range(len(row_header))[-1]:
    x.append([row_header[i], str(count_result[i]), str(compb_result[i]),\
              str(partb_result[i]), str(imb_result[i]), 'B_Avg(G)=␣
 ↪'+str(br_result[i])
```

```
  ])
 else:
  x.append([row_header[i], str(count_result[i]), str(compb_result[i]),\
            str(partb_result[i]), str(imb_result[i]), str(br_result[i])
  ])

t.add_rows(x)
t.set_cols_align(['c']+['r']* (len(x)-1))
t.header(['Transitive triads type', 'Count','Completely Balanced',\
             'Partially Balanced','Completely Imbalanced', \
            'Balance Ratio, B_T(%)'])
print(t.draw())
```

```
+--------------+-------+-------------+-------------+-------------+-------------+
|  Transitive  | Count | Completely  |  Partially  | Completely  |   Balance   |
| triads type  |       |  Balanced   |  Balanced   | Imbalanced  |   Ratio,    |
|              |       |             |             |             |   B_T(%)    |
+==============+=======+=============+=============+=============+=============+
|     030T     |   211 |         190 |           0 |          21 |      90.050 |
+--------------+-------+-------------+-------------+-------------+-------------+
|     120D     |   131 |         115 |           0 |          16 |      87.790 |
+--------------+-------+-------------+-------------+-------------+-------------+
|     120U     |   167 |         158 |           2 |           7 |      95.810 |
+--------------+-------+-------------+-------------+-------------+-------------+
|     300      |    84 |          82 |           1 |           1 |      98.810 |
+--------------+-------+-------------+-------------+-------------+-------------+
|    Total     |   593 |         545 |           3 |          45 |   B_Avg(G)= |
|              |       |             |             |             |       93.11 |
+--------------+-------+-------------+-------------+-------------+-------------+
```

[69]:
```
# to record all four types of triads semicycles' statistics (counts)
# '+ + +'
all_positive = [all_positive1, all_positive2, all_positive3, all_positive4]
# '+ + -'
one_negative = [one_negative1, one_negative2, one_negative3, one_negative4]
# '+ - -'
one_positive = [one_positive1, one_positive2, one_positive3, one_positive4]
# '- - -'
all_negative = [all_negative1, all_negative2, all_negative3, all_negative4]

#
semicycle_stat = np.array([all_positive, one_negative,\
                          one_positive, all_negative])
semicycle_sum = np.sum(semicycle_stat,1)
semicycle_ratio = np.round(semicycle_sum/np.sum(semicycle_sum),2)
```

[70]:
```
print(semicycle_stat)
```

```
[[ 74 136 182 447]
 [ 17  24  14   9]
 [116  94 136  48]
 [  4   8   2   0]]
```

[71]:
```
count_result2 = list(semicycle_sum)+[np.sum(semicycle_sum)]
ratio_total2 = list(semicycle_ratio)+[np.sum(semicycle_ratio)]
```

[72]:
```
count_result2
```

[72]:
```
[839, 64, 394, 14, 1311]
```

[73]:
```
ratio_total2
```

[73]:
```
[0.64, 0.05, 0.3, 0.01, 1.0]
```

[74]:
```
t2 = Texttable()
row_header2 = ['+++','++-','+--','---','Total']
x2 = [[]]

for i in range(len(row_header2)):
  x2.append([row_header2[i], str(count_result2[i]), str(ratio_total2[i])])

t2.add_rows(x2)
t2.set_cols_align(['c','r','r'])
t2.header(['Semicycle type', 'Counts','Ratio-Total'])
print(t2.draw())
```

```
+----------------+--------+-------------+
| Semicycle type | Counts | Ratio-Total |
+================+========+=============+
|      +++       |    839 |       0.640 |
+----------------+--------+-------------+
|      ++-       |     64 |       0.050 |
+----------------+--------+-------------+
|      +--       |    394 |       0.300 |
+----------------+--------+-------------+
|      ---       |     14 |       0.010 |
+----------------+--------+-------------+
|     Total      |   1311 |           1 |
+----------------+--------+-------------+
```

3.2 Undirected graph

[75]:
```
def have_bidirectional_relationship(G, node1, node2):
    return G.has_edge(node1, node2) and G.has_edge(node2, node1)
```

[76]:
```
def check_if_mutual(G):
  G0 = G.copy()
  edges = list(G0.edges())
```

```python
  result = {}
  for u,v in edges:
    if have_bidirectional_relationship(G0, u, v):
      if G0[u][v]['sign'] != G0[v][u]['sign']:
        # different sign
        key1 = 'False'
        if key1 in result:
          result[key1].append((u,v))
        else:
          result[key1] = [(u,v)]
      else:
        # same sign
        key2 = 'True'
        if key2 in result:
          result[key2].append((u,v))
        else:
          result[key2] = [(u,v)]
    else:
      # not reciprocal
      key3 = 'Single'
      if key3 in result:
        result[key3].append((u,v))
      else:
        result[key3] = [(u,v)]

  return result
```

[77]:
```python
check_mutual_dict = check_if_mutual(G0)
edge_to_remove = check_mutual_dict['False']
Gun = G0.copy()
Gun.remove_edges_from(edge_to_remove)
Gu = Gun.to_undirected(reciprocal= False) # keep those one-directional ties
```

[78]:
```python
Gu.number_of_nodes()
```

[78]: 65

[79]:
```python
Gu.number_of_edges()
```

[79]: 331

3.2.1 for all closed triads

[80]:
```python
from itertools import combinations
def closed_triads(G0):
  G = G0.copy()
  triad_list = []
  for nodes in combinations(G.nodes, 3):
    H = G.subgraph(nodes) # all triads
```

```
    n_edges = H.number_of_edges()
    if n_edges == 3: # only closed triads needed
      triad_list.append(nodes)
  return triad_list
```

[81]:
```
triad_list = closed_triads(Gu)
```

[82]:
```
len(triad_list)
```

[82]: 1037

[83]:
```
triad_list[:5]
```

[83]: [(0, 1, 25), (0, 1, 45), (0, 25, 45), (0, 25, 4), (0, 45, 4)]

[84]:
```
countX = 0
for triad in triad_list:
  H = Gu.subgraph(triad)
  if H.number_of_edges() != 3:
    countX +=1
```

[85]:
```
countX
```

[85]: 0

[ ]:
```
all_positive0 = 0
one_negative0 = 0
one_positive0 = 0
all_negative0 = 0

semicycles_sign0 = []

for triad in triad_list:
  H = Gu.subgraph(triad)

  # draw_triads(H)

  sign_list0=[]
  for u,v,data in H.edges(data=True):
    sign_list0.append(data['sign'])

  neg_count0 = sign_list0.count(-1)
  if (neg_count0 % 2)==0:
    # Even number of -ve means positive
    # draw_triads(H)
    semicycles_sign0.append(1)
  else:
    # Odd number of -ve means negative
    draw_triads(H)
    semicycles_sign0.append(-1)
```

```
  all_pos0, one_neg0, one_pos0, all_neg0 = semicycles_statistic(sign_list0)
  all_positive0 += all_pos0
  one_negative0 += one_neg0
  one_positive0 += one_pos0
  all_negative0 += all_neg0


complete_balance0, partial_balance0, imbalanced0, balance_ratio0 \
= triad_balance_count1(semicycles_sign0)
```

3.2.2 Summary

[91]:
```
t3 = Texttable()
row_header3 = 'flattened undirected network'
x3 = [[]]
x3.append([row_header3, str(len(triad_list)), str(complete_balance0),\
               str(partial_balance0), str(imbalanced0), str(balance_ratio0)
    ])

t3.add_rows(x3)
t3.set_cols_align(['c']*6)
t3.header([' ', 'Count','Completely Balanced',\
             'Partially Balanced','Completely Imbalanced', \
            'Balance Ratio, B_avg(G)(%)'])
print(t3.draw())
```

```
+--------------+-------+-------------+-------------+-------------+-------------+
|              | Count | Completely  |  Partially  | Completely  |   Balance   |
|              |       |  Balanced   |  Balanced   | Imbalanced  |   Ratio,    |
|              |       |             |             |             | B_avg(G)(%) |
+==============+=======+=============+=============+=============+=============+
|  flattened   | 1037  |     859     |      0      |     178     |   82.840    |
|  undirected  |       |             |             |             |             |
|   network    |       |             |             |             |             |
+--------------+-------+-------------+-------------+-------------+-------------+
```

[88]:
```
semicycle_stat1 = np.array([all_positive0, one_negative0,\
                           one_positive0, all_negative0])
semicycle_ratio1 = np.round(semicycle_stat1/np.sum(semicycle_stat1),3)
count_result4 = list(semicycle_stat1)+[np.sum(semicycle_stat1)]
ratio_total4 = list(semicycle_ratio1)+[np.sum(semicycle_ratio1)]
```

[90]:
```
t4 = Texttable()
row_header4 = ['+++','++-','+--','---','Total']
x4 = [[]]

for i in range(len(row_header4)):
```

```
    x4.append([row_header4[i], str(count_result4[i]), str(ratio_total4[i])])

t4.add_rows(x4)
t4.set_cols_align(['c','r','r'])
t4.header(['Triangle type', 'Counts','Ratio-Total'])
print(t4.draw())
```

```
+---------------+--------+-------------+
| Triangle type | Counts | Ratio-Total |
+===============+========+=============+
|      +++      |    499 |       0.481 |
+---------------+--------+-------------+
|      ++-      |    140 |       0.135 |
+---------------+--------+-------------+
|      +--      |    360 |       0.347 |
+---------------+--------+-------------+
|      ---      |     38 |       0.037 |
+---------------+--------+-------------+
|     Total     |   1037 |           1 |
+---------------+--------+-------------+
```

4 Visualise network

[ ]:
```
G0 = nx.relabel_nodes(G0, name_mapping)
```

[ ]:
```
G = G0.to_undirected()
```

[ ]:
```
partition = community.best_partition(G)
```

[ ]:
```
cmap = cm.get_cmap('viridis', max(partition.values()) + 1)
```

[ ]:
```
cmap.name
```

[ ]:
```
'viridis'
```

[ ]:
```
pos = nx.circular_layout(G)
pos.keys()
```

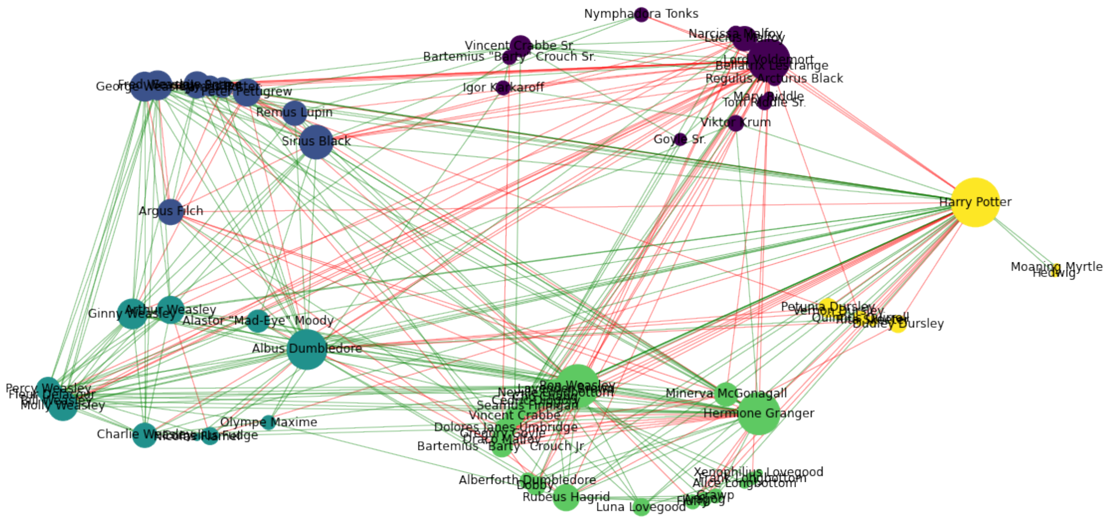
[ ]:
```
dict_keys(['Regulus Arcturus Black', 'Sirius Black', 'Bellatrix Lestrange',
'Lord Voldemort', 'Albus Dumbledore', 'Hermione Granger', 'Remus Lupin', 'Lucius
Malfoy', 'Narcissa Malfoy', 'Minerva McGonagall', 'Alastor "Mad-Eye" Moody',
'Peter Pettigrew', 'Harry Potter', 'James Potter', 'Lily Potter', 'Severus
Snape', 'Nymphadora Tonks', 'Arthur Weasley', 'Fred Weasley', 'George Weasley',
'Ginny Weasley', 'Ron Weasley', 'Lavender Brown', 'Neville Longbottom', 'Cho
Chang', 'Cedric Diggory', 'Seamus Finnigan', 'Vincent Crabbe Sr.', 'Vincent
Crabbe', 'Bartemius "Barty" Crouch Sr.', 'Dolores Janes Umbridge', 'Gregory
Goyle', 'Draco Malfoy', 'Bartemius "Barty" Crouch Jr.', 'Igor Karkaroff', 'Percy
Weasley', 'Fleur Delacour', 'Bill Weasley', 'Molly Weasley', 'Alberforth
Dumbledore', 'Dobby', 'Petunia Dursley', 'Vernon Dursley', 'Rubeus Hagrid',
'Quirinus Quirrell', 'Rita Skeeter', 'Charlie Weasley', 'Dudley Dursley', 'Argus
```

```
Filch', 'Luna Lovegood', 'Nicolas Flamel', 'Cornelius Fudge', 'Goyle Sr.',
'Fluffy', 'Aragog', 'Grawp', 'Olympe Maxime', 'Viktor Krum', 'Alice Longbottom',
'Frank Longbottom', 'Xenophilius Lovegood', 'Tom Riddle Sr.', 'Mary Riddle',
'Hedwig', 'Moaning Myrtle'])
```

[ ]:
```
def visualise_louvain(G, with_labels = True, nodesize = 'fixed'):
  if G.is_directed():
    G = G.to_undirected()

  partition = community.best_partition(G)

  nodes_by_group = {}
  for i, v in partition.items():
    nodes_by_group[v] = [i] if v not in nodes_by_group.keys() else␣
↪nodes_by_group[v] + [i]

  plt.figure(figsize=(20, 10))

  # color the nodes according to their partition
  cmap = cm.get_cmap('viridis', max(partition.values()) + 1)
  # change the node size according to their degree
  nodedegree = dict(G.degree)

  ## layout (group nodes by partition)
  # modified from: https://stackoverflow.com/questions/55750436/
↪group-nodes-together-in-networkx
  pos = nx.circular_layout(G)
  # prep center points (along circle perimeter) for the clusters
  angs = np.linspace(0, 2*np.pi, 1+len(nodes_by_group.keys()))
  repos = []
  rad = 3.0     # radius of circle
  for ea in angs:
    if ea > 0:
        #print(rad*np.cos(ea), rad*np.sin(ea))  # location of each cluster
        repos.append(np.array([rad*np.cos(ea), rad*np.sin(ea)]))
  for ea in pos.keys():
    posx = 0
    if ea in nodes_by_group[0]:
        posx = 0
    elif ea in nodes_by_group[1]:
        posx = 1
    elif ea in nodes_by_group[2]:
        posx = 2
    elif ea in nodes_by_group[3]:
        posx = 3
    elif ea in nodes_by_group[4]:
        posx = 4
```

```
    elif ea in nodes_by_group[5]:
        posx = 5
    else:
        pass
    #print(ea, pos[ea], pos[ea]+repos[posx], color, posx)
    pos[ea] += repos[posx]

  edges = list(G.edges())
  colors = []
  for u,v in edges:
    if G[u][v]['sign'] == 1:
      colors.append('g')
    else:
      colors.append('r')

  if nodesize == 'fixed':
    n_size = 100
  else:
    n_size = [v * 50 for v in nodedegree.values()]

  ec = nx.draw_networkx_edges(G, pos, edge_color = colors, alpha=0.5)
  nc = nx.draw_networkx_nodes(G, pos,\
                              nodelist = G.nodes(),\
                              node_color = list(partition.values()),\
                              node_size = n_size,\
                              cmap=cmap)

  if with_labels == True:
      labels = nx.draw_networkx_labels(G,pos)

  # plt.colorbar(nc)
  plt.axis('off')
  plt.show()
```

[ ]:
```
visualise_louvain(G0, with_labels=True, nodesize = "")
```




```
[ ]: !wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
     from colab_pdf import colab_pdf
     colab_pdf('COMP5313project21.ipynb')
```

```
--2021-05-27 00:18:56--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1865 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-05-27 00:18:56 (36.0 MB/s) - colab_pdf.py saved [1865/1865]


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```