Visualizing Hamlet through Linear Algebra

Ben Bailey

December 11, 2016

1 Introduction

This paper was inspired by Franco Moretti's work at the Stanford Literary Lab, specifically *Network Theory, Plot Analysis* [1]. One important difference from Moretti's networks is that while Moretti created his networks by hand, mine were created by a Python program I wrote. The advantage is that using an automated process to construct these networks allows one to analyze multiple different works much more quickly. However, I did not manage to get my network to detect whether the characters where on stage at the same time, and rather focused on whether they were in the same scene at all. This will lead to some differences between Moretti's network and mine.

There was also unfortunately one error in the automatic method in that Guildernstern is in the network both as "Guildenstern" and "Guildenstern:". This is a data scraping issue that I attempted but ultimately failed to remedy. To lessen this error's impact, when importing the graph into MATLAB to do calculations, I used the `'OmitSelfLoops'` option which should fix any calculation errors it could have caused. The character "All" is also mistakenly included in the network (though it is interesting to see which characters do not appear in a scene in which everyone on stage speaks).

2 Analysis

There are many different paths of analysis you can take once you've constructed a network. One of the more interesting ones in *Hamlet*'s case is examining the "centrality" of different nodes in the network, which gives the analyst a better idea of how important different characters are to the overall structure of the play. While there are many different algorithms that can be used to determine centrality, I was especially intrigued by the basic PageRank algorithm that Google originally used to order its search results (first described in [2]). Since this algorithm has obviously proved good at ordering web pages, I thought it would be interesting to see how it would interpret something radically different than its typical object.

We can use linear algebra to calculate the centrality of different nodes in the network using a rough version of Google's PageRank algorithm. The first step in this process is to construct an *adjacency matrix* (A) that looks something like this (abbreviated here for brevity—just keep in mind that in reality it is a 35x35 matrix):

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 4 & \dots & 0 \\ 0 & 0 & 0 & 1 & 5 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & 0 & 1 & \dots & 0 \end{bmatrix}$$

This matrix shows which nodes each node in the graph is connected to, as well as the weight of that connection (determined by the amount of scenes the character is in with that character). It's important to note that unlike many more standard adjacency matrices, the entries in A are weighted so that the eventual PageRank calculation takes the number of times each character is in a scene with another into account. In

addition, the diagonals of A are all 0, since the characters are never supposed to be linked to themselves. To calculate a node's PageRank, we must first find a way to transform this adjacency matrix into a *transition matrix*, a matrix of the probabilities that one moves from one node starting from another.

Graph theory tends to use the term "stochastic matrix" in place of "transition matrix." In a double-stochastic matrix, the rows and columns each sum to one. However, we will be working with a left-stochastic matrix, meaning that only each column sums to one.

We can normalize the adjacency matrix so that the columns all sum to one with the MATLAB command `T = bsxfun(@rdivide,A,sum(A))`. The result is a 35x35 matrix presented here in abbreviated form:

$$T_l = \begin{bmatrix} 0 & 0.1111 & 0 & 0 & 0.0147 & \dots & 0 \\ 0.1111 & 0 & 0 & 0 & 0.0147 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0.1250 \\ 0 & 0 & 0 & 0 & 0.0147 & \dots & 0 \\ 0.1111 & 0.1111 & 0.1250 & 0.0769 & 0 & \dots & 0.1250 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0.1250 & 0 & 0.0147 & \dots & 0 \end{bmatrix}$$

There are two ways to calculate the PageRank of the nodes from this transition matrix. The first, less algebraic way is to multiply a starting vector of equal probabilities by the transition matrix (and then multiplying the result of that multiplication by the transition matrix) until the multiplication does not alter the vector at all. In this case, the vector of initial probabilities is a column vector of length 35 with each entry equal to 1/35 (0.0286). The resultant vector of the multiplication process is a column vector, again with a length of 35.

The more algebraic method of calculating PageRank is to solve the eigenvector problem for the transition matrix T. This means we must solve the equation $T\boldsymbol{v} = \lambda\boldsymbol{v}$. The eigenvector of T_l that has an eigenvalue (λ) of 1 should be equivalent to the steady state of the matrix that we found using the previous method.

Regardless which method is used, the resultant PageRank vector ($\boldsymbol{r}$) should be a column vector with 35 entries that all sum to one like this (abbreviated here for brevity):

$$\boldsymbol{r} = \begin{bmatrix} 0.0129 \\ 0.0129 \\ 0.0114 \\ 0.0186 \\ 0.0973 \\ 0.1030 \\ 0.0043 \\ \vdots \\ 0.0114 \end{bmatrix}$$

This vector is the probability that someone starting at one random character in the network of the play will end up at the character specified by the index of the vector. For instance, Hamlet's index is 6, so he is the sixth element in the PageRank vector, meaning that the probability that someone will end on him in their journey through the network is 10.3%. Claudius' index is 5, so someone has a slightly smaller chance of ending on him when "browsing" the network, about 9.73%. The lowest rank of 1.14% with the index of 3 belongs to the First Ambassador, a relatively minor character.

Using this algorithm to determine network centrality gives different results than most which tend give the honor of highest centrality to Horatio (sorting the nodes by degree does the same). Since most centrality algorithms give Horatio the highest centrality, that's what I'll use for analysis, but it's interesting to see how different algorithms work and what happens if you treat a play the same way Google treats the Internet.

The first figure (organized with the Frutcherman-Reingold algorithm) gives us the best idea of the individual connections between characters in the play. This places Horatio in the center due to his degree

(which we will cover shortly), but here we can better focus on the connections between outer characters of the play (Figure 1). This method of organization gives the viewer a good view of the overall network of connections between characters, the "social network" of the play. However, the layout makes it hard to draw concrete conclusions about the overall structure of the play itself. For a better idea of the play's structure, we can turn to the other, more circular figures.

For each of the circular figures (2, 3, 4, and 5) I used a circular graph layout algorithm in the program Gephi to organize each figure by different attributes. This algorithm has a unique possibility of putting a select number of nodes in the center of the larger ring in an inner ring. Which nodes to place in the inner circle is decided by ranking them according to any arbitrary factor (all the figures in this paper are arranged according to some measurement of the importance of individual nodes).

Perhaps the most interesting feature of the figures ordered by traditional centrality and degree (with three characters in the center) (figures 2 and 3) is that Horatio is part of the inner circle of three nodes, but we can observe a more direct triangle of connections between Hamlet, Gertrude, and Claudius. While more characters are connected to Horatio, the three more traditionally important characters have stronger connections between each other than between themselves and Horatio. If we widen the inner circle to include the node with the fourth-highest degree, Hamlet is brought in, and a strong diamond of connections between the members of the inner circle emerges.

The last figure, Figure 5, is similar to the previous figures with one key difference: it is ordered by PageRank. Here, Hamlet takes Horatio's throne as the most central node, while Horatio moves to the position formerly occupied by Hamlet. This demonstrates the difference choosing a centrality algorithm makes. In this figure, the strong triangle of relations between the royal family does not need to circumvent Horatio but instead can keep to itself in the center of the larger circle. Horatio's importance is lessened in this interpretation, hiding an interesting facet of the play's structure that was revealed by the previous circular figures.

3 Conclusion

While PageRank may not be the best method of ranking centrality for *Hamlet*, I was impressed by how well the algorithm can be adapted to a different type of structure than it was designed for. I feel that this network analysis of Hamlet provides a unique way of interacting with a text that has been analyzed so heavily in traditional ways. In this automated analysis, I confirmed one of Moretti's most interesting findings, that Horatio is a much more important character (and much more integral to the structure of the play) than most readers realize. I believe that the automated method of constructing networks from texts is also a valuable tool that has some promise for providing quick insights about structure that humans do not ordinarily detect.

4 Code

```
In [2]: from bs4 import BeautifulSoup
        import csv
        import pprint

In [3]: f = open('hamlet.html', 'r')
        f_text = f.read()

        soup = BeautifulSoup(f_text, "lxml")

In [4]: def is_speech(tag):
            return tag.has_attr('ref') and "speech" in tag["ref"]

In [5]: acts_scenes = []
        with open('hamlet.csv', 'w') as file:
            w = csv.writer(file)
            w.writerow(["act", "scene", "character", "text"]) # add header row
```

```
            for a in soup.find_all(is_speech):
                for child in a.nextSibling.nextSibling.children:
                    if child.name == "a":
                        ref = child["ref"].split(".")
                        acts_scenes.append(ref[0] + "." + ref[1])
                        w.writerow([ref[0], ref[1], a.b.text.title(), child.text.strip()])
```

```
In [7]: with open('hamlet.csv', 'r') as file:
            r = csv.reader(file, delimiter=',')
            next(r, None) # skip header row
            m = {el:[] for el in acts_scenes}
            for row in r:
                m[row[0] + "." + row[1]].append(row[2])

        chars = []
        for k, v in m.items():
            m[k] = list(set(v))
            chars += v

        chars = list(set(chars)) # a list of characters in the play
        # m is a map of acts/scenes to the characters that appear in them
```

```
In [14]: tuples = []
         for scene, c_in_s in m.items():
             for c in c_in_s:
                 y = list(filter(lambda x: x != c, c_in_s))
                 for e in y:
                     z = (c, e, scene)
                     tuples.append(z)
```

```
In [10]: def add_tuple(t):
             a, b, c = t
             return (a, b, c)
```

```
In [11]: tuples_reduced = []
         for t in tuples:
             a, b, c = t
             if a == "Guildenstern:": # correct for a typographical error
                 a = "Guildenstern"
             tuples_reduced.append((a, b, 1))
```

```
In [23]: # create network edges
         from collections import Counter
         import networkx as nx
         G = nx.Graph()
         net_dict = {el:[] for el in chars}
         count = Counter((elem[0], elem[1]) for elem in tuples_reduced)

         for pair, weight in count.most_common(1000):
             a, b = pair
             G.add_edge(a, b, weight=weight)
```

```
In [21]: nx.write_gml(G, "hamlet.gml") # write the graph to a format Gephi understands
```

References

[1] Franco Moretti. Network theory, plot analysis. New Left Review, 2011.

[2] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

5 Figures

List of Figures

1 A Frutcherman-Reingold network colored by degree 6
2 A circular network colored by centrality and sorted by degree 7
3 A circular network colored and sorted by degree 8
4 A circular network colored and sorted by degree with four characters in the center 9
5 A circular network colored and sorted by PageRank 10

Figure 1: A Frutcherman-Reingold network colored by degree

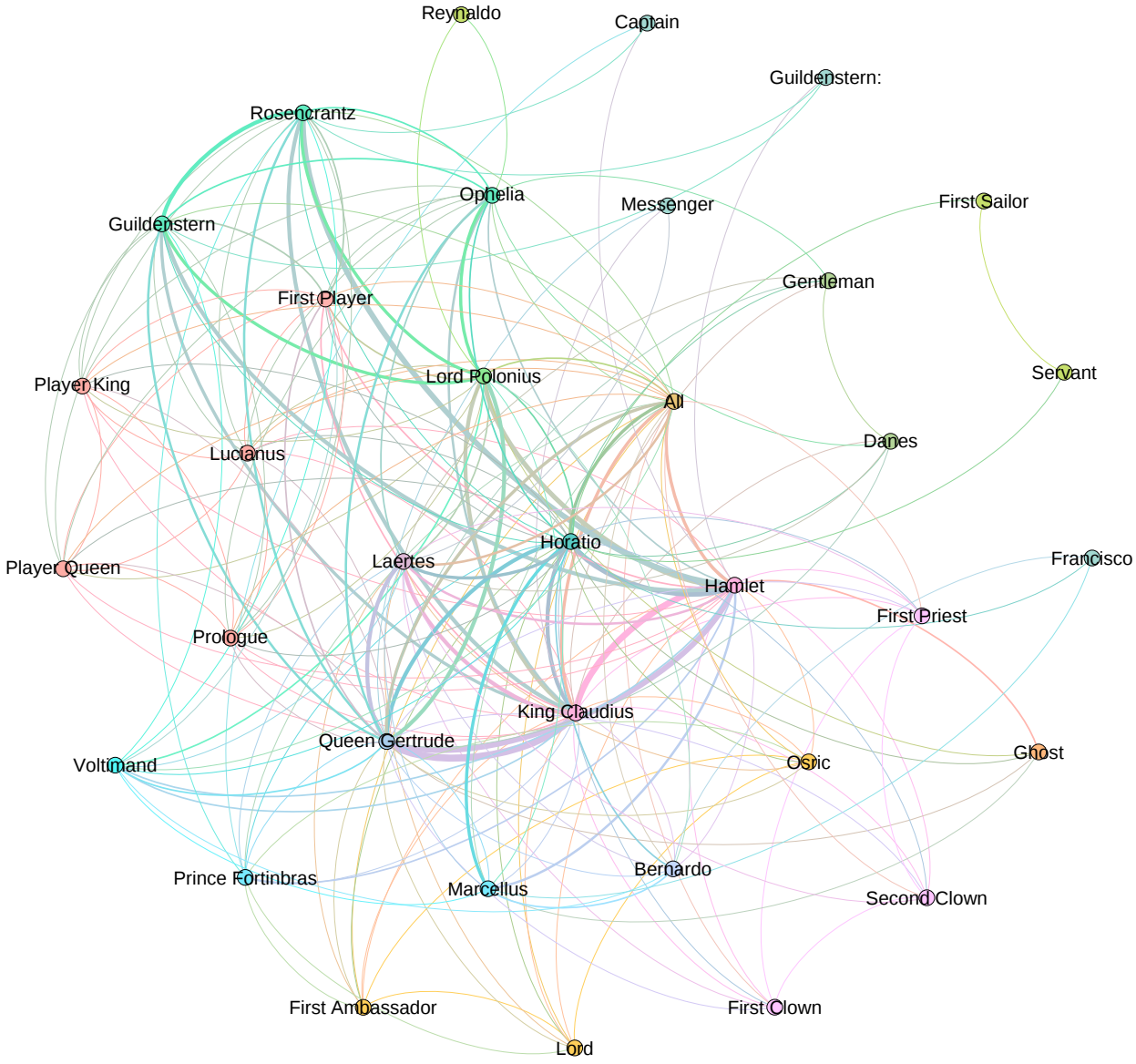

Figure 2: A circular network colored by centrality and sorted by degree

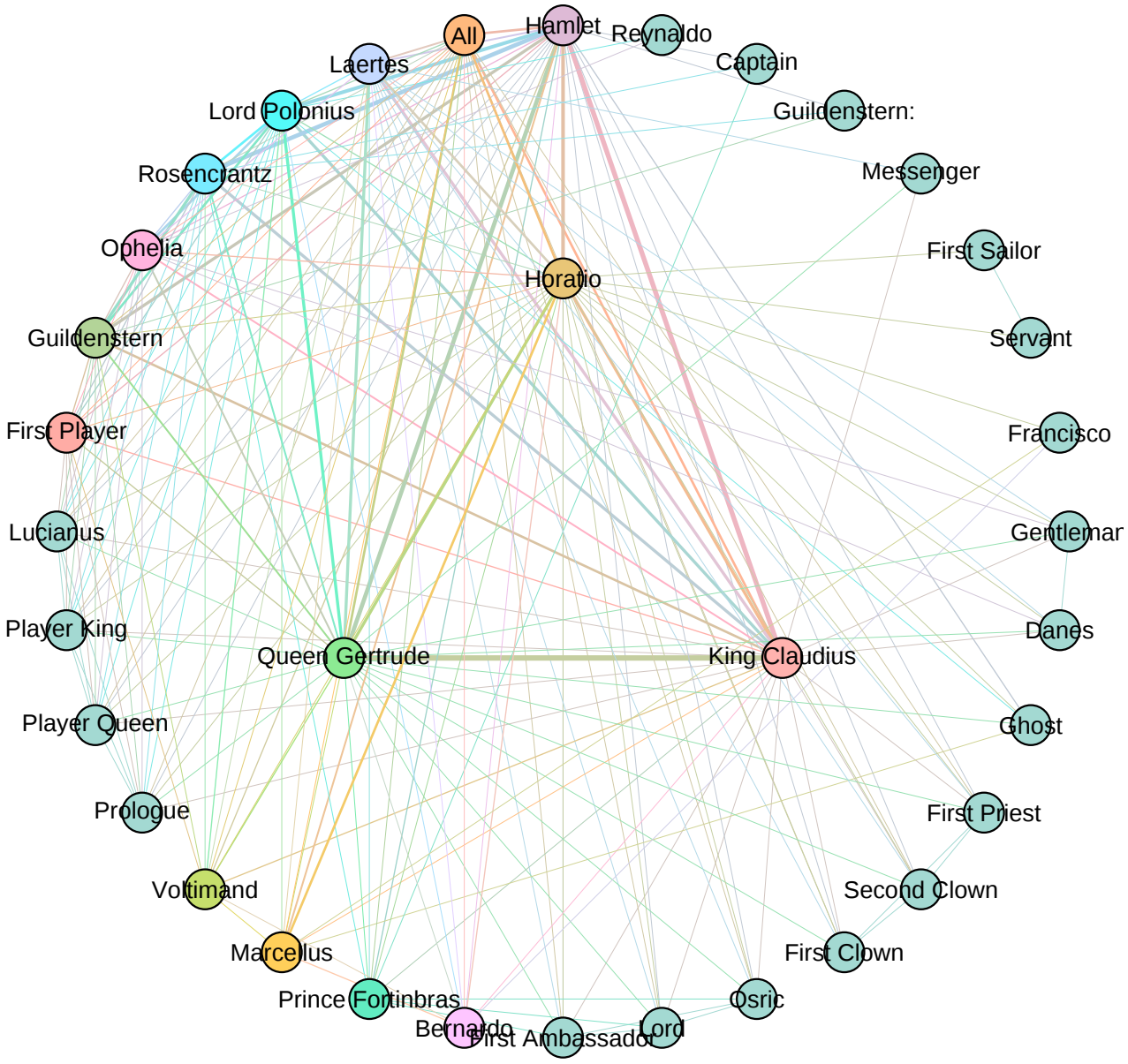

Figure 3: A circular network colored and sorted by degree

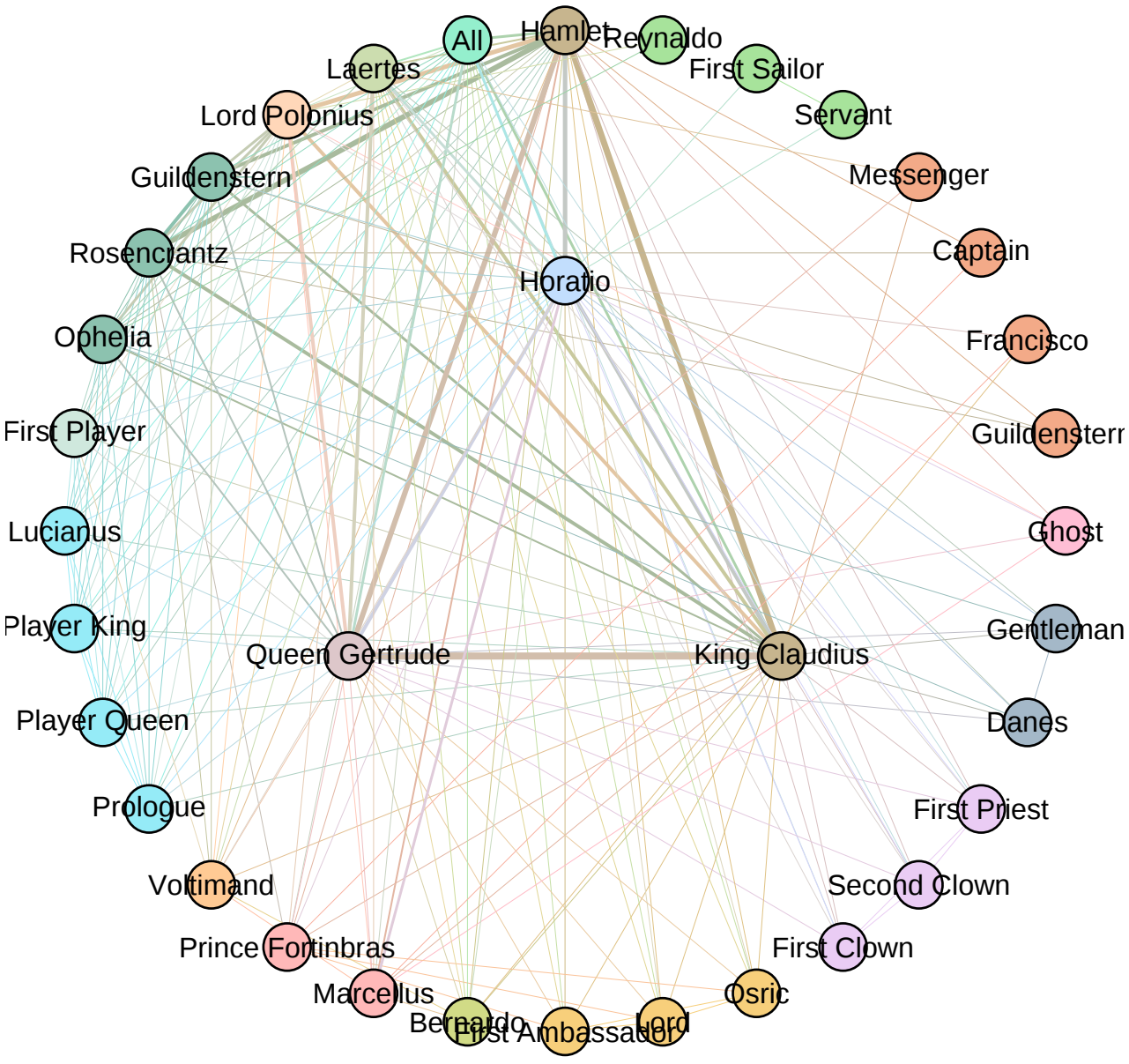

Figure 4: A circular network colored and sorted by degree with four characters in the center

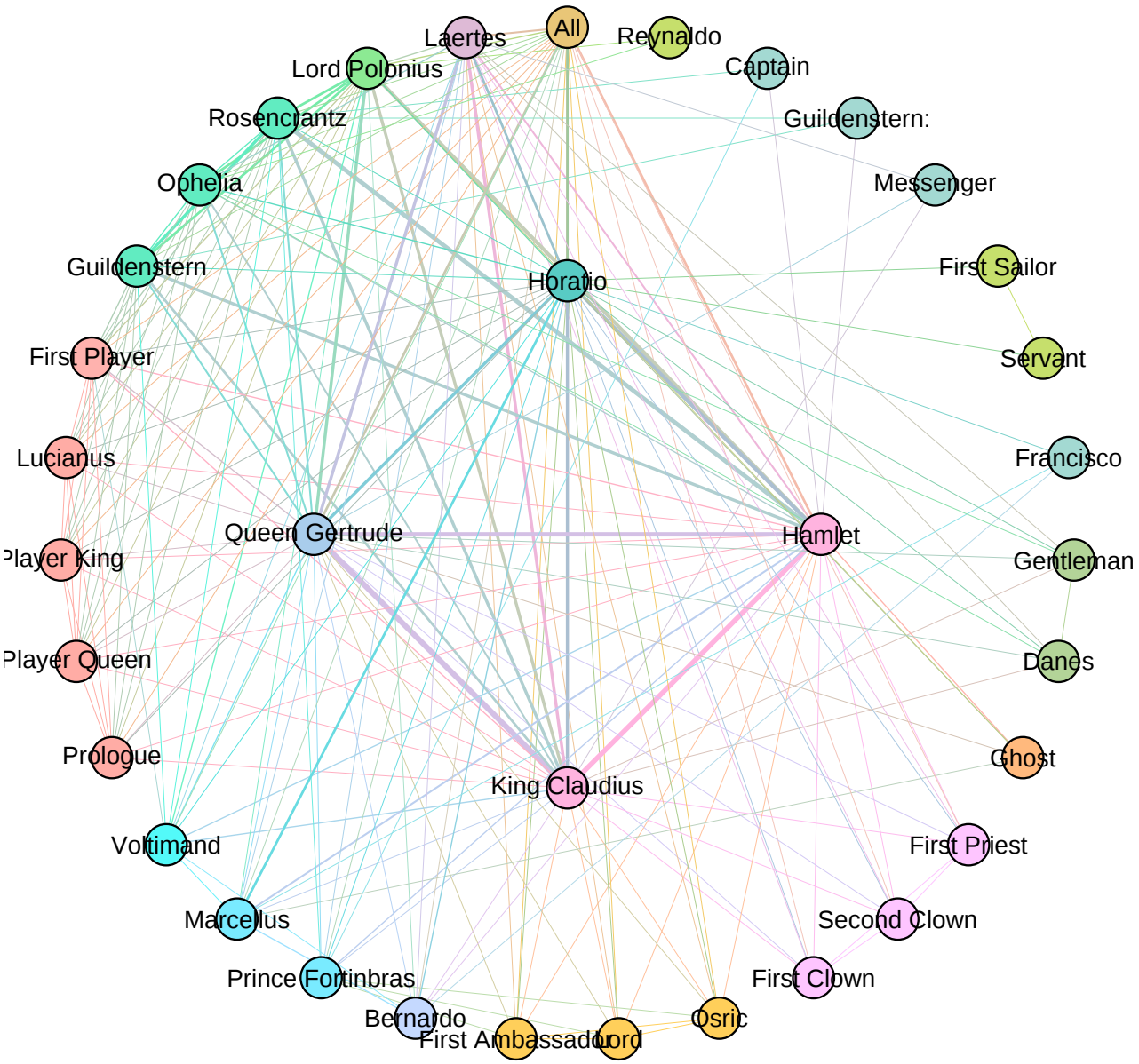

Figure 5: A circular network colored and sorted by PageRank