Who is the Centre of the Movie Universe?
Using Python and NetworkX to Analyse the Social Network of Movie Stars

Rhyd Lewis
School of Mathematics,
Cardiff University, Cardiff, Wales.
LewisR9@cf.ac.uk, http://www.rhydlewis.eu

February 27, 2020

Abstract
This paper provides the technical details of an article originally published in The Conversation in February 2020 [11]. The purpose is to use centrality measures to analyse the social network of movie stars and thereby identify the most “important” actors in the movie business. The analysis is presented in a step-by-step, tutorial-like fashion and makes use of the Python programming language together with the NetworkX library. It reveals that the most central actors in the network are those with lengthy acting careers, such as Christopher Lee, Nassar, Sukumari, Michael Caine, Om Puri, Jackie Chan, and Robert De Niro. We also present similar results for the movie releases of each decade. These indicate that the most central actors since the turn of the millennium include people like Angelina Jolie, Brahmanandam, Samuel L. Jackson, Nassar, and Ben Kingsley.

1 Introduction
Social network analysis is a branch of data science that allows the investigation of social structures using networks and graph theory. It can help to reveal patterns in voting preferences, aid the understanding of how ideas spread, and even help to model the spread of diseases [7, 12, 14].

A social network is made up of a set of nodes (usually people) that have links, or edges between them that describe their relationships. In this article we analyse the social network formed by movie actors. Each actor in this network is represented as a node. Pairs of actors are then joined by an edge if they are known to have appeared in a movie together. This information is taken from the Internet Movie Database IMDb [3]. Our analysis is carried out using the Python programming language and, in particular, the tools available in the NetworkX library [4].

2 A Small Example
Figure[1] shows a small social network formed by the actors appearing in Christopher Nolan’s three Batman movies, The Dark Knight Trilogy. As mentioned, each node in this network corresponds to an individual actor. An edge between a pair of nodes then indicates that the two actors appeared in the same movie together.

A number of features are apparent in this network. We can see that the nodes seem to be clustered into four groups. The tight cluster in the centre contains Christian Bale, Michael Caine, Gary Oldman and Morgan Freeman, who starred in all movies of the trilogy. In contrast, the remaining clusters hold the actors who appeared in just
one of the movies. The cluster at the top-right shows the actors who appeared in *Batman Begins*, the cluster at the bottom contains the stars of *The Dark Knight*, and the cluster on the left shows the actors from *The Dark Knight Rises*. We also see, for example, that Tom Hardy was in the same movie as Joseph Gordon-Levitt (in *The Dark Knight Rises*), but did not appear alongside actors such as Liam Neeson (who was a star of *Batman Begins*), or Heath Ledger (who appeared in *The Dark Knight*).

3 A Dataset of All Movies

While the Batman example shown in Figure 1 is helpful for illustrative purposes, in this article we are interested in investigating the social network of all actors from all movies. As mentioned, for this study we use information taken from the Internet Movie Database [3]. Specifically, we use a dataset compiled by the administrators of the Oracle of Bacon website [5]. Complete and up-to-date versions of this dataset can be downloaded directly from [1].

Our version of this dataset was downloaded at the start of January 2020 and contains the details of 164,318 different movies. Each movie in this set is stored as a JSON object containing, among other things, the title of the movie, a list of the cast members, and the year of its release. The complete dataset is obviously too large to reproduce here, but to illustrate the basic format, the box below shows the three-movie example used to produce the small social network shown in Figure 1.

```json
{
  "title": "Batman Begins",
  "cast": ["Christian Bale", "Michael Caine", "Liam Neeson ", "Katie Holmes", "Gary Oldman", "Cillian Murphy", "Tom Wilkinson", "Rutger Hauer", "Ken Watanabe", "Morgan Freeman"],
  "year": 2005
}
{
  "title": "The Dark Knight",
  "cast": ["Christian Bale", "Michael Caine", "Heath Ledger", "Gary Oldman", "Aaron Eckhart", "Maggie Gyllenhaal", "Morgan Freeman "],
  "year": 2008
}
{
  "title": "The Dark Knight Rises",
  "cast": ["Christian Bale", "Michael Caine", "Gary Oldman", "Anne Hathaway", "Tom Hardy", "Marion Cotillard", "Joseph Gordon-Levitt ", "Morgan Freeman"],
  "year": 2012
}
```

Before proceeding with our analysis, note that is was first necessary to remove a few “dud” movies from this dataset. In our case, we decided to remove the 44,075 movies that had no cast specified. We also deleted a further 5,416 movies that did not include a year of release. This leaves a final “clean” database of 114,827 movies with...
which to work. In the following Python code we call this file data.json.

4 Input and Preliminary Analysis

In this section we show how the dataset can be read into our program using standard Python commands. We then carry out a preliminary analysis of the data, produce some simple visualisations, and use these to help identify some inconsistencies in the dataset.

4.1 Reading the Dataset

To read the dataset, we begin by first importing the relevant Python libraries into our program. Next, we transfer the contents of the entire dataset into a Python list called Movies. Each element of this list contains the information about a single movie. The command json.loads(line) is used to convert each line of raw text (in JSON format) into an appropriate Python data structure. This is then appended to the Movies list.

```python
import json
import networkx as nx
import matplotlib.pyplot as plt
import collections
import statistics
import time
import random
Movies = []
with open("data.json", "r", encoding="utf-8") as f:
    for line in f.readlines():
        J = json.loads(line)
        Movies.append(J)
```

Having parsed the dataset in this way, we are now able to access any of its elements using standard Python commands. For example, the statement Movies[0] will return the full record of the first movie in the list; the statement Movies[0]["cast"][0] will return the name of the first cast member listed for the first movie; and so on.

4.2 Two Simple Bar Charts

Having read the dataset into the list Movies, we can now carry out some basic analysis. Here we will look at the number of movies produced per year, and the sizes of the casts that were used. The code below uses the collections.Counter() method to count the number of movies released per year. This information is written to the variable C, which is then used to produce a bar chart via the plt.bar() command.

```python
C = collections.Counter([d["year"] for d in Movies])
plt.xlabel("Year")
plt.ylabel("Frequency")
plt.title("Number of Movies per Year")
plt.bar(list(C.keys()), list(C.values()))
plt.show()
```

The resultant bar chart is shown below. As we might expect, we see that nearly all movies in this dataset were released between the early 1900’s and 2020, with a general upwards trend in the number of releases per year. However, the fact that the horizontal axis of our chart goes all the way back to 1800 hints at the existence of outliers and errors in the dataset. In fact, a few errors do exist. For example, the movie Cop starring James Woods is stated as being released in the year 1812, which is clearly ridiculous (James Woods wasn’t born until 1947, and Cop was actually released in 1988). On the other hand, a movie called Avatar 5 is given a “release date” of 2025
in the dataset, which is also incorrect (at present, only one Avatar movie has been made). Nevertheless, we will accept such oddities and continue with our investigation.

We now take a look at the sizes of casts used in movies. The following code produces a bar chart in the same way as the previous example.

```python
C = collections.Counter([len(d['cast']) for d in Movies])
plt.xlabel("Cast Size")
plt.ylabel("Frequency")
plt.title("Number of Movies per Cast Size")
plt.bar(list(C.keys()), list(C.values()))
plt.show()
```

This leads to the following bar chart:

This indicates that nearly all movies have casts of between one and fifty actors. However, there are again some outliers with much larger casts. To get the names of these movies, the following code reorders the list Movies into descending order of cast size. The first five movies on this list are then written to the screen.

```python
Movies = sorted(Movies, key=lambda i: len(i['cast']), reverse=True)
for i in range(5):
    print(Movies[i]['title'], "=" , len(Movies[i]['cast']))
```
This produces the following output, indicating the five movies with the largest cast sizes.

| Movie                                      | Cast Size |
|--------------------------------------------|-----------|
| Cirque du Soleil: Worlds Away              | 268       |
| Hollywood Without Make-Up                 | 132       |
| The Longest Day (film)                    | 117       |
| The Founding of a Party                   | 116       |
| The Founding of a Republic                | 106       |

As a final point, we can also see in the above bar chart that there is a preponderance of movies with a cast size of one. In some cases this is correct, such as with the 2018 stand-up comedy movie *Russell Brand: Re:Birth*. On the other hand, this also reveals some further problems in the dataset. For example, the movie *Lady with a Sword* (1971) is also recorded as having a cast size of one despite the fact that many actors actually appeared in it, such as Lily Ho, James Nam and Hsieh Wang.

## 5 Forming the Social Network

In this section we now construct the complete social network of actors using our dataset together with tools available in the Python library NetworkX. As mentioned earlier, our network is made up of nodes (actors in this case), with edges connecting actors that have appeared in a movie together. Probably the most appropriate type of network to use here is a *multigraph*. Multigraphs allow us to define multiple edges between the same pair of nodes, which makes sense here because actors will often appear in multiple movies together. Note also that the edges in this network are not directed. This means that if actor A has appeared with actor B, then B has also appeared with A.

The following code constructs our network $G$ using the *Movies* list from the previous section. As shown, the code considers each movie in turn. It then goes through each pair of actors that appeared in this movie and adds the appropriate edge to the network. Each edge is also labelled with the corresponding movie title. Upon construction of the network, the methods $G.number\_of\_nodes()$ and $G.number\_of\_edges()$ are then used to output some information to the screen.

```python
G = nx.MultiGraph()
for movie in Movies:
    for i in range(0, len(movie["cast"]) - 1):
        for j in range(i + 1, len(movie["cast"])):
            G.add_edge(movie["cast"][i], movie["cast"][j], title=movie["title"])  

print("Number of nodes in this multigraph =", G.number_of_nodes())
print("Number of edges in this multigraph =", G.number_of_edges())
```

As shown in the following output, the resultant network is very large, with a total of 395,414 different nodes (actors) and 9,968,607 different edges.

| Property                  | Value     |
|---------------------------|-----------|
| Number of nodes in this multigraph | 395414    |
| Number of edges in this multigraph | 9968607  |

## 6 Analysing Connections in the Network

Having formed our social network of actors, we can now analyse some of its interesting features. In this section we start by calculating the total number of movies that each actor has appeared in. We then determine the most prolific acting partnerships in the movie business by calculating the number of movies that each pair of actors has starred in.
6.1 Movies Per Actor

The following piece of code calculates the total number of movies per actor and lists the top five. For each node in our network, this is achieved by going through its incident edges and forming a set $S$ of all the different labels (movie titles) appearing on these edges. The final results are stored in the dictionary $D$. For output purposes, the contents of $D$ are then put into a sorted list $L$, and the first five entries in this list are written to the screen.

$$D = {}$$

for $v$ in G.nodes():
    $E = \text{list}(G.edges(v, \text{data=True}))$
    $S = \text{set()}$
    for $e$ in $E$:
        $S.add(e[2]["title"])$
    $D[v] = S$

$L = \text{sorted}(D.items(), \text{key=lambda item: len(item[1]), reverse=True})$

for $i$ in $\text{range}(5)$:
    $\text{print}(L[i][0], ":", \text{len}(L[i][1]))$

The output below shows the results. We see that the top positions are occupied by actors from Indian cinema, with the great Sukumari (1940–2013) winning the competition with 703 recorded movie appearances. The top one-hundred actors from this list are shown in Appendix A at the end of this document.

Sukumari : 703
Jagathy Sreekumar : 695
Adoor Bhasi : 579
Brahmanandam : 576
Manorama : 558

6.2 Acting Partnerships

We now consider the number of collaborations between different pairs of actors—that is, the number of movies that each pair of actors has appeared in together.

The following code calculates these figures. It goes through every pair of actors that are known to have appeared in at least one movie together, and then counts the total number of edges between the corresponding nodes. This information is collected in the dictionary $D$, which is again copied into an ordered list $L$. Again, the top five collaborations are then reported.

$$D = {}$$

for $e$ in G.edges():
    $D[e[0] + \text{" and "} + e[1]] = G.number_of_edges(e[0], e[1])$

$L = \text{sorted}(D.items(), \text{key=lambda kv: kv[1], reverse=True})$

for $i$ in $\text{range}(5)$:
    $\text{print}(L[i][0], ":", L[i][1])$

The output from this code is below. We see that the most prolific acting partnership in this network is due to the late Indian actors Adoor Bhasi (1927–1990) and Prem Nazir (1926–1991), who appeared in an impressive 292 movies together. Next on the list are Larry Fine and Moe Howard (two of the Three Stooges) who co-starred in 216 movies. By comparison, the comedy partnership of Oliver Hardy and Stan Laurel resulted in a paltry 105 movies, putting them at position 46 in the list overall. The top one-hundred acting partnerships are also listed in Appendix A.

Adoor Bhasi and Prem Nazir : 292
Larry Fine and Moe Howard : 216
Adoor Bhasi and Sankaradi : 207
Adoor Bhasi and Bahadoor : 198
7 Calculating Shortest Paths

As we have seen, when two actors have not appeared in a movie together there will be no edge between the corresponding nodes in the social network. However, we can still look for connections between actors by using paths of intermediate actors. This is similar to the so-called “Six Degrees of Separation”—the idea that all people are six or fewer social connections away from each other [6].

Connecting actors using chains of intermediate actors is an idea popularised by the Oracle of Bacon website [5], who provide a simple tool for finding shortest paths between any pair of actors. As mentioned earlier, the Oracle of Bacon is also the source of the dataset used in this work.

As an example, according to our dataset we find that the actors Anthony Hopkins and Samuel L. Jackson have never appeared in a movie together. In our network, this means that the corresponding two nodes have no edge between them. However, these nodes can still be regarded as fairly “close” to one another because, in this case, they are both linked to the node representing Scarlett Johansson. (Specifically, Anthony Hopkins acted with Scarlett Johansson in Hitchcock, and Samuel L. Jackson appeared with Johansson in Captain America: The Winter Soldier.) The shortest path from Anthony Hopkins to Samuel L. Jackson therefore has a length of two, since we need to travel along two edges in the network to get from one actor to the other. In reality, there may be many paths between Anthony Hopkins and Samuel L. Jackson in our network. However, determining the shortest path tells us that there are no paths with fewer edges.

Before looking at the techniques used in identifying shortest paths, we will first simplify our network slightly by converting it into a “simple graph”. Simple graphs allow a maximum of one edge between a pair of nodes; hence, when we have multiple edges between a pair of nodes in our multigraph (because the two actors have appeared in multiple movies together), these will now be represented as a single edge. Note that this conversion maintains the number of nodes in the network but it reduces the number of edges. It will therefore make some of our calculations a little quicker. The following code constructs our simple graph. The final line then checks whether this new network is connected. (A network is connected when it is possible to form a path between every pair of nodes.)

```python
G = nx.Graph()
for movie in Movies:
    for i in range(0, len(movie["cast"]) - 1):
        for j in range(i + 1, len(movie["cast"])):
            G.add_edge(movie["cast"][i], movie["cast"][j], title=movie["title"],
            G.add_node(movie["cast"]
            G.add_node(movie["cast"]
            G.add_node(movie["cast"]

print("Number of nodes in simple graph =", G.number_of_nodes())
print("Number of edges in simple graph =", G.number_of_edges())
print("Graph Connected? =", nx.is_connected(G))
```

This produces the following output. As can be seen, the network G still has 395,414 nodes, but it now contains 8,676,962 edges instead of the original 9,968,607—a 13% reduction. We also see that the network is not connected; that is, it is composed of more than one distinct connected component.

| Number of nodes in simple graph | 395414 |
|--------------------------------|--------|
| Number of edges in simple graph | 8676962 |
| Graph Connected?               | False  |

Shortest paths can now be found in our social network using the NetworkX command `nx.shortest_path()`. If the edges of the graph are unweighted (as is the case here) then this invokes a breadth first search; otherwise the slightly more expensive Dijkstra’s algorithm is used. Both of these methods are reviewed by Rosen [13]. In either case, the output from this command is a list of nodes \( P \) that specifies the shortest path between the two specified nodes. For example, the code:
P = nx.shortest_path(G, source="Anthony Hopkins", target="Samuel L. Jackson")
print(P)
gives the following output.

[Anthony Hopkins, Scarlett Johansson, Samuel L. Jackson]

While this code does indeed tell us the shortest path between Anthony Hopkins and Samuel L. Jackson, it does not give the names of the movies involved in this path. In addition, if no path exists between the actors, or if we type in a name that is not present in the network, then the program will halt with an exception error. A better alternative is to therefore put the `nx.shortest_path` statement into a bespoke function, and then add some code that (a) checks for errors, and (b) writes the output in a more helpful way. The following code does this.

```python
def writePath(G, u, v):
    print("Here is the shortest path from ", u, " to ", v, ":")
    if not u in G or not v in G:
        print("Error: ", u, " and/or ", v, " are not in the network")
        return
    try:
        P = nx.shortest_path(G, source=u, target=v)
        for i in range(len(P) - 1):
            t = G.edges[P[i], P[i + 1]]['title']
            print(" ", P[i], " was in ", t, " with ", P[i + 1])
    except nx.NetworkXNoPath:
        print("No path exists between ", u, " and ", v)
writePath(G, "Catherine Zeta-Jones", "Jonathan Pryce")
writePath(G, "Homer Simpson", "Neil Armstrong")
```

The bottom two lines of the above code make two calls to the `writePath()` function, resulting in the following output.

```
Here is the shortest path from Catherine Zeta-Jones to Jonathan Pryce
  Catherine Zeta-Jones was in Ocean's Twelve with Albert Finney
  Albert Finney was in Loophole with Jonathan Pryce
Here is the shortest path from Homer Simpson to Neil Armstrong
  Error: Homer Simpson and/or Neil Armstrong are not in the network
```

8 Connectivity and Centrality Analysis

In this section we now use three techniques from the field of centrality analysis to help us identify who the most “central” and “important” actors are in our social network.

Recall from the last section that our network is currently not connected. This means that the graph is made up more than one connected component, and that paths between actors in different components do not exist. To investigate these connected components, we can use the command `nx.connected_components(G)` to construct a list holding the number of nodes in each. Details of these can then be written to the screen:

```python
C = [len(c) for c in sorted(nx.connected_components(G), key=len, reverse=True)]
print("Number of components = ", len(C))
print("Component sizes = ", C)
```

The output of these statements is shown below. We see that our network of actors is actually made up of 2,533 different components; however, the vast majority of the nodes (96%) all occur within the same single connected
component, indicating the existence of paths between all of these 379,859 different actors. We also see that the remaining components are very small (in most cases they are composed of actors who all appeared in the same single together movie, but no others).

**Number of components = 2533**  
**Component sizes = [379859, 72, 46, 45, 42, 40, ..., 2, 2]**

For the remainder of our analysis we will now focus on the single connected component of 379,859 actors. To do this we first need to isolate this component. We can then use the `subgraph()` command to form the network represented by the component.

```python
C = max(nx.connected_components(G), key=len)  
G = G.subgraph(C)
```

The output of the above code confirms that this new network is indeed connected as we would expect.

**Number of nodes in simple graph = 379859**  
**Number of edges in simple graph = 8612493**  
**Graph Connected? = True**

The following three subsections will now investigate the “centrality” of the nodes appearing in this connected network. As mentioned, three different measures will be considered: degree centrality, betweenness centrality, and closeness centrality.

### 8.1 Degree Centrality

In networks, the **degree** of a node is simply the number of edges that are touching it. For the network of actors that we have now formed, the degree therefore represents the number of different people that this actor has worked with. To access the degree of a node, the simplest option is to use the command `G.degree(v)`.

```python
print(G.degree("Henry Fonda"))
```

produces the output

```
1325
```
telling us that the actor Henry Fonda has appeared in film with 1,325 different actors. A second option is to use the NetworkX function `nx.degree_centrality(G)`, which calculates the degree centrality of each node in the network.

The degree centrality of a node is determined by dividing the degree of the node by the maximum possible degree of a node, which in this case is simply the number of nodes in the network minus one (i.e., 379,858). The degree centrality of Henry Fonda, for example, is therefore calculated as 1,325 divided by 379,858, giving 0.00349. This figure can be interpreted as the proportion of actors in the network that Henry Fonda has acted with—in this case, just under 0.35%.

The following code creates a dictionary `D` that holds the degree centrality of all nodes in the network. As with previous examples, the top five actors according to this measure are then written to the screen.

```python
D = nx.degree_centrality(G)  
L = sorted(D.items(), key=lambda item: item[1], reverse=True)  
for i in range(5):  
    print(L[i][0], ":", L[i][1])
```
This code produces the output below. To determine the actual degree of these nodes (i.e., the number of different actors that each actor has worked with), we simply need to multiply these figures by the number of nodes minus one. We then find that Nassar has appeared with a massive 2,937 different actors; Sukumari with 2,549; Manorama with 2,511; Brahmanandam with 2,460; and Vijayakumar with 2,369. A listing of the top one-hundred actors is given in Appendix B.

| Actor       | Betweenness Centrality |
|-------------|------------------------|
| Nassar      | 0.00773183663689431    |
| Sukumari    | 0.006710402308230971   |
| Manorama    | 0.0064761042284222     |
| Brahmanandam| 0.00623654102322468    |
| Vijayakumar | 0.006610364925840709   |

One other notable actor on this list is the English actor Christopher Lee (1922–2015), who appears at position 11, having appeared on screen with 2,056 different actors. Our reasons for mentioning Christopher Lee in particular will become apparent in the next two subsections.

8.2 Betweenness Centrality

Betweenness centrality considers the number of shortest paths in a network that pass through a particular node. In social networks this helps to detect the “middlemen” that serve as a links between different parts of a network. It also helps to identify “hubs” in the network that, when removed, will start to disconnect parts of the network from each other. A useful analogy can be drawn with road networks of cities. Shortest paths that travel across the city will often pass through the same locations in the road network (consider the Arc de Triomphe in Paris, for example). As a result, the nodes at these locations can be considered more “central” to the network.

The formula for calculating the betweenness centrality of an individual node \( v \) in a network is as follows:

\[
C(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}
\]

where \( V \) is the set of nodes in the network, \( \sigma(s,t) \) is the number of different shortest paths between two nodes \( s \) and \( t \), and \( \sigma(s,t|v) \) is the number of these \( s-t \)-paths that are seen to pass through \( v \). In other words, the betweenness centrality of a node \( v \) is the sum of the fraction of all-pairs shortest paths that pass through node \( v \).

We can calculate the betweenness centrality of all actors in our social network using the NetworkX command `nx.betweenness_centrality(G)`. This uses an algorithm designed by Brandes [8]. However, executing this command on a network as big as ours is infeasible. This is because it involves having to calculate all of the shortest paths between \( k \) pairs of nodes, which would take a huge amount of calculation. (In technical terms, it involves using an algorithm that has a complexity of \( O(nm) \), where \( n \) is the number of nodes and \( m \) is the number of edges.) Luckily, we can make some savings in these calculations by using a sample of \( k \) nodes to estimate the betweenness centrality of all nodes [9]. This is carried out by the following code using a sample of 1,000 nodes. As before, the top five results are written to the screen. Some statements are also included to allow us to measure how long the calculation takes.

```python
start = time.time()
D = nx.betweenness_centrality(G, k=1000)
end = time.time()
print("Time taken =", end - start, "seconds")
L = sorted(D.items(), key=lambda item: item[1], reverse=True)
for i in range(5):
    print(L[i][0], ":", L[i][1])
```

The output of this code is shown below. It indicates that Christopher Lee appears on by far the highest number of shortest paths in the network, followed by Om Puri, Jackie Chan, Anupam Kher, and then Harrison Ford. As indicated, this calculation took approximately 10 hours to carry out on our computer (a 3.2 GHz Windows 10
machines with 8 GB RAM). This suggests that a full calculation using all nodes instead of just a sample would have taken something in the region of 150 days to complete. The top one-hundred actors for this measure are listed in Appendix B.

| Name                | Value          |
|---------------------|----------------|
| Christopher Lee     | 0.009681571845060294 |
| Om Puri             | 0.008096614699536075 |
| Jackie Chan         | 0.007916442261041035 |
| Anupam Kher         | 0.007905221698069204 |
| Harrison Ford       | 0.004888140405090745 |

**8.3 Closeness Centrality**

Like betweenness centrality, closeness centrality also considers shortest paths in a network. For a given node \( v \) it represents the mean shortest path length from \( v \) to all other nodes in the network \([10, 14]\). If an actor is found to be connected to other actors via short paths, they can therefore be considered to be quite central in the network. The formula used for calculating the closeness centrality of a particular node \( v \) is as follows:

\[
C(v) = \frac{1}{\left( \sum_{u \in V} d(v, u) \right) / n}
\]

where \( d(v, u) \) is the length of the shortest path (number of edges) between nodes \( v \) and \( u \), and \( n \) is the number of nodes in the network. Higher values of this measure therefore indicate a higher centrality. Note that we can also calculate the actual mean path length from a node \( v \) to all other nodes in the network by simply dividing 1 by \( C(v) \).

As with the previous example, calculating the closeness centrality of all nodes in our large network of actors would take too long on a single computer because, once again, it involves calculating the shortest paths between all pairs of nodes. In our case we make things easier by restricting our calculations to the top 1,000 actors according to the betweenness centrality measure from the previous subsection. The following code does this. First, it produces a list \( V \) of all actors in the network, ordered according to their betweenness centrality score. The closeness centrality is then calculated for each of the first 1,000 actors in this list. These are then ranked, and the top five are output.

```python
V = [L[i][0] for i in range(len(L))]  
D = {}  
start = time.time()  
for i in range(1000):  
    D[V[i]] = nx.closeness_centrality(G, V[i])  
end = time.time()  
print("Time taken =", end-start, "seconds")  
L = sorted(D.items(), key=lambda item: item[1], reverse=True)  
for i in range(5):  
    print(L[i][0], ":", L[i][1])
```

The following output shows that, of these actors, Christopher Lee is again the most central. Amazingly, we find that we can get from Christopher Lee to any other actor in the network in an average of just 2.88 hops. The next best-connected actors are then, respectively, Michael Caine (average of 2.917 hops), Harvey Keitel (2.922), Christopher Plummer (2.931) and Robert De Niro (2.936). The top one-hundred actors according to this measure are also listed in Appendix B.

| Name                | Value          |
|---------------------|----------------|
| Christopher Lee     | 0.34724000968978963 |
| Michael Caine       | 0.3428649311215694 |
| Harvey Keitel       | 0.3422440138966974 |
| Christopher Plummer | 0.34123099284135183 |
| Robert De Niro      | 0.34059459561239835 |
If we want, we can also take a closer look at the number of actors within a certain number of hops from a chosen actor. For example, the following code creates a dictionary \( D \) that holds the length of the shortest path between Christopher Lee and all other actors in the network. It then counts the number of actors of distance 0, 1, 2, and so on.

\[
D = \text{nx.shortest_path_length}(G, "Christopher Lee") \\
\text{print(collections.Counter(D.values()))}
\]

The output from this code tells us that all actors can be reached from Christopher Lee using fewer than nine hops. Exactly one actor can be reached in zero hops (Christopher Lee himself); 2,056 actors can be reached with one hop; 98,758 with two hops; and so on. In particular, note that over 86% of actors can be reached from Christopher Lee in fewer than four hops, and 99% in fewer than five hops.

\[
\text{Counter({0: 1, 1: 2056, 2: 98758, 3: 226350, 4: 48625, 5: 3655, 6: 369, 7: 36, 8: 9})}
\]

**8.4 Distribution of Actors’ Closeness Centrality Scores**

Finally, it is also interesting to look at the distribution of different actors’ closeness centrality scores. We have already seen that actors like Christopher Lee, Om Puri, and Michael Caine are very central and well connected, but what is the score of a “typical” actor. Once again, the expense of calculating shortest paths between all pairs of actors is prohibitively expensive for a large network like ours. Instead, the code below takes a random sample of 1,000 actors, calculates their closeness centrality scores, works out the mean and standard deviation of this sample, and then uses the bespoke function `doHistogramSummary()` to plot this information to the screen.

```
def doHistogramSummary(X, xlabel, ylabel, title):
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.hist(X, bins=20)
    plt.show()

random.shuffle(V)
D = {}
for i in range(1000):
    D[V[i]] = 1 / nx.closeness_centrality(G, V[i])
L = D.values()
print("Mean =", statistics.mean(L))
print("SD =", statistics.stdev(L))
doHistogramSummary(L, "Average distance to all Actors", "Frequency", "Closeness centrality distribution")
```

The output from this code is shown below. As we can see, the average distance between any two actors in this sample is just over 4.27 hops. As a comparison, this is slightly lower than the six hops hypothesised in the Six Degrees of Separation; however, it is slightly higher than the mean of 3.57 found in a similar analysis of Facebook friendships carried out by Facebook Research in 2016 [2].

```
Mean  = 4.2704787482812225
SD    = 0.4701311459607602
```
9 Conclusions and Further Discussion

This article has used tools from the NetworkX library to help determine the most important people in the social network of movie actors. Regardless of the measures used, the most central actors are consistently those who have or had very long acting careers, such as Christopher Lee, Nassar, Sukumari, Michael Caine, Om Puri, and Jackie Chan. This is quite natural, because long careers bring more acting opportunities, helping to improve an actor's connectivity in the network. To contrast these findings, Appendix C shows the twenty most central actors by decade. These statistics were found in the same way as above, but only used movies that were released in that particular decade. As we might expect, this causes many new names to crop up. For movies released in the 1950's, for example, actors such as Louis de Funes (1914–1983) and George Thorpe (1891–1961) seem to be very central; for the 1990's on the other hand, the most central actors are people like Samuel L. Jackson, Om Puri, Vijayakumar, Roshan Seth and Frank Welker.

There are other ways in which we might have performed this analysis. Alternative centrality measures are also included in the NetworkX library, such as page rank centrality, Eigenvector centrality, Katz centrality, and current-flow closeness centrality. A good review of these measures can be found in the book of Needham and Hodler [12]. In the future, it would also be interesting to add some kind of values to the edges of the network in order to give more information about the nature of an acting collaboration. This could include the number of minutes that both actors appeared on screen, the critical ratings of the movie, or the financial earnings. Such factors would certainly influence the set of central actors that are identified.

All materials from this study are available online:

- A complete listing of the dataset, code, and results tables can be found at http://www.rhydlewis.eu/movies/all.zip.
- A shorter web version of this document can be found at http://www.rhydlewis.eu/movies.

References

[1] Complete dataset. https://oracleofbacon.org/data.txt.bz2. Accessed 18/02/2020.
[2] Facebook Research: Three and half degrees of separation. https://research.fb.com/blog/2016/02/three-and-a-half-degrees-of-separation/ Accessed 18/02/2020.
[3] Internet movie database. https://www.imdb.com/ Accessed 18/02/2020.
A Movies per Actor and per Partnerships

The following table shows the top one hundred actors (left) and acting partnerships (right) according to the number of movies they have appeared in. Note that position 68 in the right list is occupied by a pair of actors called “TBA and TBA”. This is clearly a fault in the underlying dataset.

| # | Movies Per Actor | Movies Per Acting Partnership |
|---|-----------------|-------------------------------|
| 1 | Sukumari | 703 | Adoor Bhasi and Prem Nazir | 292 |
| 2 | Jagathy Sreekumar | 695 | Larry Fine and Moe Howard | 216 |
| 3 | Adoor Bhasi | 579 | Adoor Bhasi and Sankaradi | 207 |
| 4 | Brahmamandam | 576 | Adoor Bhasi and Bahadoor | 198 |
| 5 | Manorama | 558 | Brahmamandam and Ali | 193 |
| 6 | Sankaradi | 545 | Brahmamandam and Kota Srinivasa Rao | 170 |
| 7 | Prem Nazir | 518 | Adoor Bhasi and K. P. Ummer | 170 |
| 8 | Nedumudi Venu | 470 | Prem Nazir and Bahadoor | 163 |
| 9 | Bahadoor | 450 | Brahmamandam and Tanikella Bharani | 157 |
| 10 | Nassar | 438 | Adoor Bhasi and Jayabharathi | 156 |
| 11 | Meena | 413 | Sankaradi and Bahadoor | 155 |
| 12 | Mammootty | 395 | Senthil and Goundamani | 151 |
| 13 | Senthili | 381 | Harold Lloyd and Snub Pollard | 148 |
| 14 | Nagesh | 375 | Prem Nazir and Sankaradi | 147 |
| 15 | Oliver Hardy | 373 | Bebe Daniels and Harold Lloyd | 146 |
| 16 | Innocent | 371 | Bebe Daniels and Snub Pollard | 146 |
| 17 | Vijayakumar | 368 | Sukumari and Sankaradi | 145 |
| 18 | Shakti Kapoor | 365 | K. P. Ummer and Prem Nazir | 144 |
| 19 | Madhu | 357 | Jagathy Sreekumar and Sukumari | 141 |
| 20 | Mohanlal | 348 | Meena and Adoor Bhasi | 139 |
| 21 | Mithun Chakraborty | 347 | Jayabharathi and Prem Nazir | 138 |
|    | Name                        | Total | Year(s)                                                                 |
|----|-----------------------------|-------|-------------------------------------------------------------------------|
| 22 | Kota Srinivasa Rao          | 346   | 134                                                                     |
| 23 | Ali                         | 343   | 127                                                                     |
| 24 | Srividya                    | 333   | 126                                                                     |
| 25 | Prakash Raj                 | 329   | 123                                                                     |
| 26 | Kuthiravattam Pappu          | 329   | 122                                                                     |
| 27 | Tanikella Bharani           | 324   | 120                                                                     |
| 28 | Thilakan                    | 319   | 120                                                                     |
| 29 | Jayabharathi                | 318   | 120                                                                     |
| 30 | K. P. Ummer                 | 316   | 119                                                                     |
| 31 | Mala Aravindan              | 312   | 115                                                                     |
| 32 | Kaviyoor Ponnamma           | 310   | 114                                                                     |
| 33 | Siddique                    | 301   | 113                                                                     |
| 34 | Rekha                       | 296   | 111                                                                     |
| 35 | Sivaji Ganesan              | 295   | 110                                                                     |
| 36 | Anupam Kher                 | 290   | 110                                                                     |
| 37 | Lakshmi                     | 288   | 110                                                                     |
| 38 | N. T. Rama Rao              | 285   | 109                                                                     |
| 39 | Indrans                     | 283   | 108                                                                     |
| 40 | M. N. Nambiar               | 280   | 108                                                                     |
| 41 | Ambika                      | 276   | 108                                                                     |
| 42 | Jose Prakash                | 276   | 108                                                                     |
| 43 | Prathapachandran            | 276   | 107                                                                     |
| 44 | Asrani                      | 275   | 107                                                                     |
| 45 | MG Soman                    | 272   | 106                                                                     |
| 46 | Vadivelu                    | 271   | 105                                                                     |
| 47 | Vijayaraghavan              | 270   | 105                                                                     |
| 48 | Sheela                      | 265   | 104                                                                     |
| 49 | Saikumar                    | 264   | 104                                                                     |
| 50 | Aruna Irani                 | 264   | 104                                                                     |
| 51 | Mukesh                      | 261   | 103                                                                     |
| 52 | Devan                       | 258   | 103                                                                     |
| 53 | Paravoor Bharathan          | 257   | 101                                                                     |
| 54 | Venniradai Moorthy          | 256   | 100                                                                     |
| 55 | Rajkumar                    | 256   | 99                                                                     |
| 56 | Gulshan Grover              | 256   | 99                                                                     |
| 57 | Goundamani                  | 254   | 98                                                                     |
| 58 | Delhi Ganesh                | 252   | 98                                                                     |
| 59 | V. K. Ramasamy              | 252   | 97                                                                     |
| 60 | Kader Khan                  | 251   | 97                                                                     |
| 61 | Murali                      | 247   | 97                                                                     |
| 62 | Raymond Hatton              | 247   | 97                                                                     |
| 63 | Pandari Bai                 | 246   | 96                                                                     |
| 64 | Dharmendra                  | 244   | 96                                                                     |
| 65 | Kovai Sarala                | 244   | 96                                                                     |
| 66 | Harry Carey                 | 244   | 96                                                                     |
| 67 | KPAC Lalitha                | 244   | 96                                                                     |
| 68 | Salim Kumar                 | 242   | 95                                                                     |
| 69 | Manohala                    | 242   | 94                                                                     |
| 70 | Kalpana                     | 241   | 92                                                                     |
| 71 | Balakrishna                 | 241   | 92                                                                     |
| 72 | Mamukkoya                   | 240   | 91                                                                     |
| 73 | Prem Chopra                 | 240   | 90                                                                     |
| 74 | Cochin Haneefa              | 239   | 89                                                                     |
| 75 | Avinash                     | 239   | 88                                                                     |
| 76 | Allu Ramalingaiah           | 236   | 89                                                                     |
| 77 | K. S. Ashwath               | 236   | 88                                                                     |
| 78 | Charle                      | 236   | 88                                                                     |
B Centrality Analysis of the Entire Network

The following table lists the top one hundred actors according to degree centrality, closeness centrality and betweenness centrality. These statistics were calculated on the single connected network comprising 379,859 nodes (actors) and 8,612,493 edges. Note the presence of some spurious entries in the table such as “Tarzan”, “King Kong”, and “Sam”, which reveal some problems with the underlying dataset.

| #  | Actor Score | Actor Score | Betweenness Centrality Score | Actor Score | Closeness Centrality Score |
|----|-------------|-------------|-------------------------------|-------------|---------------------------|
| 1  | Nassar 2337  | Christopher Lee | 0.00568                      | Christopher Lee | 2.880 |
| 2  | Sukumari 2549 | Om Puri 0.00810 | Harrison Ford 0.00489        | Michael Caine | 2.917 |
| 3  | Manorama 2511 | Jackie Chan 0.00792 | Christopher Plummer 0.00791 | Robert De Niro | 2.931 |
| 4  | Brahmanandam 2460 | Anupam Kher 0.00791 | John Hurt 0.00467 | John Gielgud | 2.954 |
| 5  | Vijayakumar 2369 | Harrison Ford 0.00489 | Louis de Funes 0.00405 | Samuel L. Jackson | 2.975 |
| 6  | Prakash Raj 2349 | Klaus Kinski 0.00470 | Geraldine Chaplin 0.00395 | Ben Kingsley | 2.977 |
| 7  | Jagathy Sreekumar 2223 | Anupam Kher 0.00791 | George Kennedy 0.00394 | Max von Sydow | 2.979 |
| 8  | Mithun Chakraborty 2142 | Tarzan 0.00467 | Robert De Niro 0.00489 | Ernest Borgnine | 2.981 |
| 9  | Rekha 2116 | Marcello Mastroianni 0.00448 | Martin Sheen 0.00448 | Dennis Hopper | 2.985 |
| 10 | Nedumudi Venu 2063 | Rutger Hauer 0.00448 | Sean Connery 0.00373 | David Carradine | 2.988 |
| 11 | Christopher Lee 2056 | Kabir Bedi 0.00373 | Max von Sydow 0.00361 | David Carradine | 2.988 |
| 12 | John Carradine 2027 | Jeanne Moreau 0.00405 | Saeed Jaffrey 0.00371 | Terence Stamp | 2.988 |
| 13 | Meena 1944 | Gerard Depardieu 0.00403 | Saeed Jaffrey 0.00371 | John Gielgud | 2.992 |
| 14 | Anupam Kher 1943 | Isabelle Huppert 0.00395 | Saeed Jaffrey 0.00371 | Trevor Howard | 3.003 |
| 15 | Emory Parnell 1912 | George Kennedy 0.00394 | Saeed Jaffrey 0.00371 | Vanessa Redgrave | 3.003 |
| 16 | Frank Welker 1912 | Stellan Skarsgard 0.00371 | Saeed Jaffrey 0.00371 | Gerard Depardieu | 3.004 |
| No. | Actor (Name)       | Actor (Name)       | Actor (Name)       | Actor (Name)       | Actor (Name)       |
|-----|------------------|------------------|------------------|------------------|------------------|
| 24  | Shakti Kapoor    | Joseph           | 0.00318          |   | 3.004          |
| 25  | Saikumar         | Shabana Azmi     | 0.00315          |   | 3.010          |
| 26  | Mammootty        | Donald Sutherland| 0.00309          |   | 3.011          |
| 27  | Devan            | Michael Caine    | 0.00305          |   | 3.011          |
| 28  | Nagesh           | Amrish Puri      | 0.00303          |   | 3.004          |
| 29  | Manobala         | Jean-Claude Van Damme | 0.00296 |   | 3.015          |
| 30  | Kalpana          | Alex             | 0.00294          |   | 3.016          |
| 31  | Paul Fix         | John Savage      | 0.00286          |   | 3.016          |
| 32  | Irving Bacon     | Michel Piccoli   | 0.00284          |   | 3.018          |
| 33  | Delhi Ganesh     | I. S. Johar      | 0.00282          |   | 3.018          |
| 34  | Mohanlal         | Martin Sheen     | 0.00280          |   | 3.019          |
| 35  | J. Farrell MacDonald | Naseeruddin Shah | 0.00279 |   | 3.019          |
| 36  | Siddique         | Gulshan Grover   | 0.00279          |   | 3.022          |
| 37  | Omo Puri         | George Baker     | 0.00275          |   | 3.022          |
| 38  | Geetha           | Dennis Hopper    | 0.00273          |   | 3.022          |
| 39  | Indrans          | Harvey Keitel    | 0.00270          |   | 3.022          |
| 40  | Raymon Hatton    | Jason Flemyng    | 0.00266          |   | 3.023          |
| 41  | Jackie Chan      | Thomas Kretschmann | 0.00262 |   | 3.023          |
| 42  | Eric Tsang       | Haluk Bilginer   | 0.00259          |   | 3.023          |
| 43  | Louis de Funes   | Liam Neeson      | 0.00255          |   | 3.023          |
| 44  | Madhu            | King Kong        | 0.00253          |   | 3.025          |
| 45  | Vijayaraghavan   | Mithun Chakraborty | 0.00253 |   | 3.026          |
| 46  | Mickey Rooney    | Daniel Olbrychski | 0.00252          |   | 3.026          |
| 47  | Charles Lane     | Udo Kier         | 0.00252          |   | 3.026          |
| 48  | Jackie Shroff    | Rade Serbedzija  | 0.00251          |   | 3.026          |
| 49  | Ambika           | Charlotte Rampling | 0.00247          |   | 3.027          |
| 50  | Innocent         | Danny Trejo      | 0.00242          |   | 3.028          |
| 51  | Michael Caine    | Ahn Sung-ki      | 0.00242          |   | 3.032          |
| 52  | Denny Trejo      | Claudia Cardinale| 0.00242          |   | 3.032          |
| 53  | Senthil          | Omar Sharif      | 0.00241          |   | 3.032          |
| 54  | Jayaram          | David Carradine  | 0.00240          |   | 3.033          |
| 55  | Gulshan Grover   | Sean Connery     | 0.00237          |   | 3.033          |
| 56  | Marcello Mastroianni | John Carradine | 0.00236          |   | 3.033          |
| 57  | Kovai Sarala     | Roshan Seth      | 0.00225          |   | 3.034          |
| 58  | Murali           | Anthony Quinn    | 0.00223          |   | 3.035          |
| 59  | Pierre Watkin    | Prem Chopra      | 0.00222          |   | 3.036          |
| 60  | Vivek            | Ashish Vidyarthi | 0.00222          |   | 3.036          |
| 61  | Ward Bond        | Tom Alter        | 0.00221          |   | 3.036          |
| 62  | Srividya         | Franco Nero      | 0.00220          |   | 3.036          |
| 63  | Dharmandra       | John Hurt        | 0.00219          |   | 3.037          |
| 64  | Gerard Depardieu | Shin Seong-il    | 0.00217          |   | 3.037          |
| 65  | Samuel L. Jackson| Michael Kelly    | 0.00217          |   | 3.038          |
| 66  | Avishek          | Shashi Kapoor    | 0.00215          |   | 3.038          |
| 67  | Mukesh           | Bruno Ganz       | 0.00213          |   | 3.040          |
| 68  | Nizhalgal Ravi   | Eric Roberts     | 0.00212          |   | 3.041          |
| 69  | Rajesh           | Mohan Agashe     | 0.00212          |   | 3.041          |
| 70  | Sharat Saxena    | Maria            | 0.00212          |   | 3.042          |
| 71  | Klaus Kinski     | Rohini Hattangadi| 0.00210          |   | 3.042          |
| 72  | Russell Hicks    | Trevor Howard    | 0.00210          |   | 3.042          |
| 73  | Naseeruddin Shah | Paul Guiffoyle   | 0.00210          |   | 3.043          |
| 74  | Asrani           | Willem Dafoe     | 0.00207          |   | 3.044          |
| 75  | George Chandler  | Christopher Plummer | 0.00207 |   | 3.044          |
| 76  | Prabhu           | Sam Anderson     | 0.00204          |   | 3.047          |
| 77  | Donald Sutherland| Fernando Rey     | 0.00204          |   | 3.047          |
| 78  | Hubert von Meyerinck | Anthony Hopkins | 0.00204          |   | 3.047          |
| 79  | Anthony Quinn    | Anil Kapoor      | 0.00202          |   | 3.047          |
| 80  | Thilakan         | Vittorio Gassman | 0.00201          |   | 3.048          |
C  Centrality Analysis by Decade

The following table lists the top twenty actors in movies released by decade according to degree centrality, closeness centrality and betweenness centrality. Once again, note the presence of a small number of strange entries in this tables, such as actors with the single names “Art”, “Joseph”, “David” and “John”.

| #  | Actor                                      | Degree Centrality | Closeness Centrality | Betweenness Centrality |
|----|--------------------------------------------|-------------------|----------------------|------------------------|
| 1  | Mary Pickford                              | 353               | 0.14071              | 0.04026                |
| 2  | George Fawcett                             | 327               | 0.06066              | 0.03548                |
| 3  | Karl Platen                                | 269               | 0.02036              | 0.01528                |
| 4  | Alfred Paget                               | 261               | 0.01816              | 0.01276                |
| 5  | Bebe Daniels                               | 253               | 0.01616              | 0.01088                |
| 6  | Tully Marshall                             | 251               | 0.01607              | 0.01084                |
| 7  | Marguerite Clark                           | 251               | 0.01515              | 0.01084                |
| 8  | Walter Long                                | 245               | 0.01494              | 0.01064                |
| 9  | Bud Jamison                                | 245               | 0.01406              | 0.01040                |
| 10 | William Elmer                              | 244               | 0.01390              | 0.01025                |

1910–1920 (single connected network with 9,282 nodes (actors) and 109,181 edges.)

| #  | Actor                                      | Degree Centrality | Closeness Centrality | Betweenness Centrality |
|----|--------------------------------------------|-------------------|----------------------|------------------------|
| 1  | Mary Pickford                              | 353               | 0.14071              | 0.04026                |
| 2  | George Fawcett                             | 327               | 0.06066              | 0.03548                |
| 3  | Karl Platen                                | 269               | 0.02036              | 0.01528                |
| 4  | Alfred Paget                               | 261               | 0.01816              | 0.01276                |
| 5  | Bebe Daniels                               | 253               | 0.01616              | 0.01088                |
| 6  | Tully Marshall                             | 251               | 0.01607              | 0.01084                |
| 7  | Marguerite Clark                           | 251               | 0.01515              | 0.01084                |
| 8  | Walter Long                                | 245               | 0.01494              | 0.01064                |
| 9  | Bud Jamison                                | 245               | 0.01406              | 0.01040                |
| 10 | William Elmer                              | 244               | 0.01390              | 0.01025                |

1920–1930 (single connected network with 15,219 nodes (actors) and 297,025 edges.)

| #  | Actor                                      | Degree Centrality | Closeness Centrality | Betweenness Centrality |
|----|--------------------------------------------|-------------------|----------------------|------------------------|
| 1  | Mary Pickford                              | 353               | 0.14071              | 0.04026                |
| 2  | George Fawcett                             | 327               | 0.06066              | 0.03548                |
| 3  | Karl Platen                                | 269               | 0.02036              | 0.01528                |
| 4  | Alfred Paget                               | 261               | 0.01816              | 0.01276                |
| 5  | Bebe Daniels                               | 253               | 0.01616              | 0.01088                |
| 6  | Tully Marshall                             | 251               | 0.01607              | 0.01084                |
| 7  | Marguerite Clark                           | 251               | 0.01515              | 0.01084                |
| 8  | Walter Long                                | 245               | 0.01494              | 0.01064                |
| 9  | Bud Jamison                                | 245               | 0.01406              | 0.01040                |
| 10 | William Elmer                              | 244               | 0.01390              | 0.01025                |
|   | Name                           | Score  | Degree | Year          |
|---|--------------------------------|--------|--------|---------------|
| 1 | Emory Parnell                  | 1066   | Pal    | 1930–1940     |
| 2 | Pierre Watkin                 | 971    | Dev Anad | 1930–1940     |
| 3 | Irving Bacon                  | 863    | Rossano | 1930–1940     |
| 4 | Russell Hicks                | 839    | Amir Banu | 1930–1940     |
| 5 | Ray Walker                  | 835    | Arturo | 1930–1940     |
| 6 | Addison Richards             | 753    | Kamala Kotnis | 1930–1940     |
| 7 | Joseph Crehan                | 736    | Hans Klering | 1930–1940     |
| 8 | Jerome Cowan                 | 734    | Yelena Tyapkina | 1930–1940     |
| 9 | Lloyd Corrigan               | 732    | Signe Hasso | 1930–1940     |
| 10 | John Liel                    | 715    | Marcel Dalio | 1930–1940     |
| 11 | Thurston Hall                | 699    | Anjali Devi | 1930–1940     |
| 12 | Chester Clute                 | 698    | Hans Straat | 1930–1940     |
| 13 | Byron Foulger                | 688    | Mai Zetterling | 1930–1940     |
| 14 | Selmer Jackson               | 687    | Erich Ponto | 1930–1940     |
| 15 | Richard Lane                 | 685    | Ingrid Bergman | 1930–1940     |
| 16 | Edward Gargan                | 676    | Florence Marly | 1930–1940     |
| 17 | Roy Barcroft                 | 658    | Fortunio Bonanova | 1930–1940     |
| 18 | Douglas Fowley               | 652    | Pushpavalli | 1930–1940     |
| 19 | Charles Halton                | 649    | Kishore Sahu | 1930–1940     |
|   | Harry Davenport | Virgilio Teixeira | Matthew Boulton |
|---|----------------|------------------|-----------------|
| 20| 624            | 0.01277          | 3.221           |

1950–1960 (single connected network with 36,549 nodes (actors) and 786,464 edges.)

|   | Name                  | Degree | Page  |
|---|-----------------------|--------|-------|
| 1 | Louis de Funes        | 1110   | 20    |
| 2 | Sam Kydd              | 773    | 20    |
| 3 | Sid James             | 755    | 20    |
| 4 | Richard Wattis        | 685    | 20    |
| 5 | Eric Pohlmann         | 650    | 20    |
| 6 | Emory Parnell         | 645    | 20    |
| 7 | Morris Ankrum         | 607    | 20    |
| 8 | Geoffrey Keen         | 578    | 20    |
| 9 | Louis de Funes        | 1110   | 20    |
| 10| Martin Boddy          | 535    | 20    |
| 11| Marianne Stone        | 534    | 20    |
| 12| Michael Ripper        | 532    | 20    |
| 13| Ernst Waldow          | 530    | 20    |
| 14| Reinhard Kolldehoff   | 528    | 20    |
| 15| Cyril Chamberlain     | 526    | 20    |
| 16| Myron Healey          | 525    | 20    |
| 17| Whit Bissell          | 524    | 20    |
| 18| Nerio Bernardi        | 513    | 20    |
| 19| Lyle Talbot           | 509    | 20    |
| 20| Laurence Naismith     | 507    | 20    |

1960–1970 (single connected network with 39,217 nodes (actors) and 704,331 edges.)

|   | Name                  | Degree | Page  |
|---|-----------------------|--------|-------|
| 1 | Nagesh                | 700    | 20    |
| 2 | John Le Mesurier      | 654    | 20    |
| 3 | Klaus Kinski          | 639    | 20    |
| 4 | Manorama              | 619    | 20    |
| 5 | Terry-Thomas          | 605    | 20    |
| 6 | David Lodge           | 588    | 20    |
| 7 | Warren Mitchell       | 582    | 20    |
| 8 | Marianne Stone        | 574    | 20    |
| 9 | John Wayne            | 561    | 20    |
|10 | Narasimharaju         | 561    | 20    |
|11 | Jayanthi              | 554    | 20    |
|12 | Balakrishna           | 540    | 20    |
|13 | Rajkumar              | 539    | 20    |
|14 | Jean-Paul Belmondo    | 534    | 20    |
|15 | Fernando Sancho       | 515    | 20    |
|16 | Sylva Koscina         | 508    | 20    |
|17 | Robert Morley         | 503    | 20    |
|18 | Kirk Douglas          | 492    | 20    |
|19 | Graham Stark          | 489    | 20    |
|20 | Richard Wattis        | 489    | 20    |

1970–1980 (single connected network with 47,221 nodes (actors) and 751,203 edges.)

|   | Name                  | Degree | Page  |
|---|-----------------------|--------|-------|
| 1 | Manorama              | 691    | 20    |
| 2 | Sankaradi             | 674    | 20    |
| 3 | Adoor Bhasi           | 668    | 20    |
| 4 | Bahadoor              | 617    | 20    |
| 5 | Lakshmi               | 556    | 20    |
| 6 | Helen                 | 551    | 20    |
| 7 | Jayabharathi          | 521    | 20    |
| 8 | Prem Nazir            | 514    | 20    |
| 9 | Klaus Kinski          | 508    | 20    |
| 10| Balakrishna           | 503    | 20    |
|11 | John Carradine        | 502    | 20    |
|12 | K. P. Ummer           | 496    | 20    |
|13 | Ku Feng               | 492    | 20    |
|   |   |   |   |   |
|---|---|---|---|---|
| 1 | Sukumari | 484 | Harry Andrews | 0.01126 |
| 2 | Christopher Lee | 482 | Oleg Vidov | 0.01073 |
| 3 | Donald Pleasence | 480 | Sylvia Miles | 0.01027 |
| 4 | Manjula | 475 | Sanjeev Kumar | 0.01090 |
| 5 | Nagesh | 470 | Christopher Lee | 0.00975 |
| 6 | Denholm Elliott | 466 | George Kennedy | 0.00997 |
| 7 | Hsu Hsia | 465 | William Thomas | 0.00955 |

1980–1990 (single connected network with 50,628 nodes (actors) and 776,358 edges.)

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | Sukumari | 766 | Amrish Puri | 0.04610 |
| 2 | Ambika | 765 | Saeed Jaffrey | 0.02497 |
| 3 | Anuradha | 752 | Roy Chiao | 0.01607 |
| 4 | Jayamalini | 722 | Martin Sheen | 0.01454 |
| 5 | Mithun Chakraborty | 665 | Tom Alter | 0.01304 |
| 6 | Seema | 663 | Roy Kinnear | 0.01326 |
| 7 | Radha | 654 | Janaki | 0.01228 |
| 8 | Manorama | 649 | John Gielgud | 0.01200 |
| 9 | Silk Smitha | 640 | James Fox | 0.01159 |
| 10 | Vishnuvardhan | 628 | Christopher Lee | 0.01150 |
| 11 | Geetha | 623 | Max von Sydow | 0.01146 |
| 12 | Jagathy Sreekumar | 622 | Klaus Kinski | 0.01095 |
| 13 | Shakti Kapoor | 620 | John Hurt | 0.01095 |
| 14 | Srividya | 602 | Isabelle Huppert | 0.01015 |
| 15 | Mammootty | 593 | Jackie Chan | 0.00995 |
| 16 | Nedumudi Venu | 583 | Klaus Maria Brandauer | 0.00967 |
| 17 | Madhavi | 578 | Igor Yasulovich | 0.00944 |
| 18 | Amrish Puri | 570 | Marcello Mastroianni | 0.00915 |
| 19 | Andy Lau | 566 | Everett McGill | 0.00886 |
| 20 | Rekha | 565 | Rohini Hattangadi | 0.00849 |

1990–2000 (single connected network with 56,962 nodes (actors) and 966,791 edges.)

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | Vijayakumar | 1065 | Om Puri | 0.03197 |
| 2 | Senthil | 1038 | Roshan Seth | 0.03107 |
| 3 | Goundamani | 865 | Shabana Azmi | 0.01726 |
| 4 | Srividya | 857 | Suman | 0.01631 |
| 5 | Brahmanandam | 815 | Danny Denzongpa | 0.01532 |
| 6 | Nassar | 806 | Greta Scacchi | 0.01502 |
| 7 | Murali | 802 | Maggie Cheung | 0.01450 |
| 8 | Frank Welker | 796 | Nagma | 0.01419 |
| 9 | Andy Lau | 794 | Shakti Kapoor | 0.01408 |
| 10 | Mithun Chakraborty | 781 | Captain Raju | 0.01401 |
| 11 | Venniradai Moorthy | 781 | Gulshan Grover | 0.01072 |
| 12 | Jagathy Sreekumar | 772 | Stellan Skarsgard | 0.01057 |
| 13 | Shakti Kapoor | 763 | Valeria Golino | 0.00985 |
| 14 | Sukumari | 755 | Frank Welker | 0.00915 |
| 15 | Meena | 751 | Shashi Kapoor | 0.00908 |
| 16 | Manorama | 732 | King Kong | 0.00847 |
| 17 | Thilakan | 726 | Rekha | 0.00813 |
| 18 | Delhi Ganesh | 709 | Tcheky Karyo | 0.00787 |
| 19 | Charle | 696 | Michael Ironside | 0.00736 |
| 20 | Rekha | 683 | Pete Postlethwaite | 0.00704 |

2000–2010 (single connected network with 91,150 nodes (actors) and 1,441,852 edges.)

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | Brahmanandam | 1307 | Om Puri | 0.03160 |
| 2 | Nassar | 1129 | Jackie Chan | 0.02348 |
| 3 | Prakash Raj | 1108 | David Carradine | 0.01230 |
| 4 | Ashish Vidyarthi | 1022 | Anupam Kher | 0.01108 |
| 5 | Tanikella Bharani | 1008 | Snoop Dogg | 0.00958 |
| 6 | Ali | 987 | Ashish Vidyarthi | 0.00864 |
| 7 | Jagathy Sreekumar | 973 | Juliette Binoche | 0.00738 |
|   | Name                  | Degree | Page 1998-2002 | Degree | Page 2010-2020 | Distance | Degree | Page 1998-2002 | Degree | Page 2010-2020 | Distance | Degree | Page 1998-2002 | Degree | Page 2010-2020 |
|---|----------------------|--------|----------------|--------|----------------|----------|--------|----------------|--------|----------------|----------|--------|----------------|--------|----------------|
| 8 | Venu Madhav          | 921    | 0.00732        | Brian Cox | 3.162          |
| 9 | Sunil                | 910    | 0.00682        | Owen Wilson | 3.171          |
| 10 | M. S. Narayana       | 887    | 0.00641        | Snoop Dogg | 3.177          |
| 11 | Kota Srinivasa Rao   | 887    | 0.00641        | Steve Coogan | 3.177         |
| 12 | Kalabhavan Mani      | 876    | 0.00580        | Steve Buscemi | 3.181         |
| 13 | Vivek                | 858    | 0.00573        | John Cleese | 3.185          |
| 14 | Vijayakumar          | 833    | 0.00558        | Danny Trejo | 3.186          |
| 15 | Vadivelu             | 825    | 0.00548        | Morgan Freeman | 3.186        |
| 16 | Dharmavaranapu Subramanyam | 811 | 0.00545         | Peter Stormare | 3.189       |
| 17 | Cochin Haneefa       | 805    | 0.00536        | Willem Dafoe | 3.189         |
| 18 | Devan                | 801    | 0.00529        | Luke Wilson | 3.190          |
| 19 | Om Puri              | 761    | 0.00528        | Woody Harrelson | 3.190      |
| 20 | Salim Kumar          | 759    | 0.00525        | Thomas Kretschmann | 3.191  |

2010–2020 (single connected network with 108,739 nodes (actors) and 1,756,677 edges.)