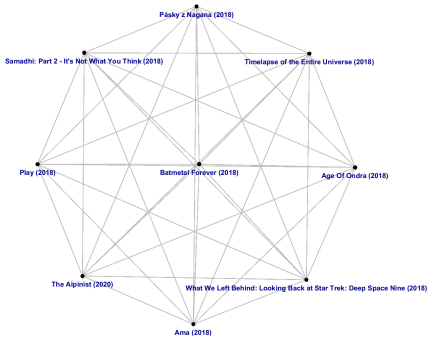
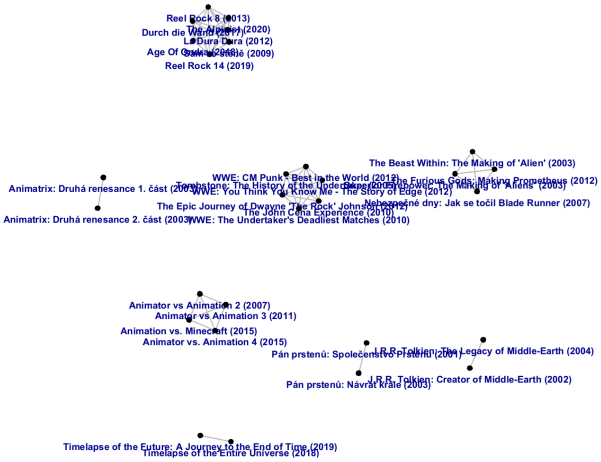
ČSFD Network Analysis

František Szczepanik, MFF UK





Theoretical background and motivation - feel free to skip this part

Network science is an academic field closely related to Graph Theory that studies complex networks such as the human brain, the Internet, or various social networks. One subject of interest in network science is network symmetry. In many regards, complex networks behave randomly and their structure is similar to those of random models, despite that, it has been shown that numerous symmetries emerge in large complex networks[1]. These definitions of symmetries are mostly based on automorphism groups of the graph. Recently, a new definition of Approximate Symmetry was introduced, which has the ability to express levels of symmetry of a graph that does not have any non-trivial automorphisms, but still apparently contains some symmetry[2] [3].

In my thesis, I am trying to improve Simulated Annealing by guiding it with different graph centralities to efficiently compute Approximate Symmetry (AS) for a given graph, because such AS is represented by a Permutation, so the solution space is large. In this project, I will work towards constructing a complex network consisting of Films, and then examine it and look for higher levels of symmetry than symmetry in a random graph with an equal number of nodes and edges. Ideally, I would want to see that there is higher symmetry in the Film Graph than in the random Graph.

User Guide for Film Network Console

Welcome to the Film Network Console. This Console-like interface allows you to interact with and analyze a film network. You can load nodes (films) into the network, create edges (links between films) based on certain conditions they need to fulfill, visualize the network, export the network data, and fetch and download more film data from the ČSFD Portal for new networks.

How the program operates

This section provides a brief description of the program on a high level.

As a user, you can create networks based on certain conditions and then visualize them, but more importantly, export them and analyze them more thoroughly (e.g. in

Python using NetworkX).

The first part is fetching the data from ČSFD. This is done by going to ČSFD and running your detailed query (on Films). You can do that here.

After getting the results, you copy the Search Parameters of the query. That is the suffix of the URL coming after `searchParams`. So, as an example, if the URL corresponding to your detailed query looks like this: `https://www.csfd.cz/podrobne-vyhledavani/?searchParams=rlW0rKOyVwco`, it means the Search Parameters are `rlW0rKOyVwco`. You will use them to obtain all the links associated with the Search on all pages of the search. The method `fetch links` is dedicated to that. All the Films Links will be stored in the directory FilmLinks, but the user does not have to specify that when naming the target file.

In all functions that are downloading data from ČSFD, it is important to set some reasonable timeout between calls so as not to get banned. The default is 30 seconds, even though a smaller time window is also fine.

The method `fetch links` gets all the links associated with the search and stores them in the target file. The method `download` then iterates over that file and one by one fetches the data for the Films. Again, the Film Data files are stored in the directory FilmData, but the user does not specify it. What is already included in the program is Data and Links for Czech and Czechoslovak films after the year 1989 until today with at least one rating, altogether about 3500 films (and some filtered German and American films with good ratings). So the user can start by experimenting with that data.

After downloading the Data and parsing it in a legible format, you can load the Films~Nodes into the nodes of the graph (network) with the command `load nodes`. To specify which of the films should be included, you can filter them using a Node Condition. That is done by typing out a simple JSON String with optional fields. See below for a detailed specification of the possible Conditions for both Nodes and Edges. The conditions should always be inline strings.

After adding the nodes to the network, it will be populated with edges (command `create edges`). An edge will be placed between two Films if they satisfy some BiPredicate. The most common usage will be having N common actors.

You can also visualize the network by the `visualize` command. The GraphStream Java library is used for that. I find it much less intuitive and effective than Python's NetworkX, but it suffices for demonstrating that the program does what it should do.

The conditions are implemented as Functional Interfaces. The class ConditionFactory uses the Factory Method Pattern to produce these Conditions from the String. All of the Parsers, Downloaders, and Media Entities should be implemented in an extensible way (implementing interfaces) so that creating networks from other fields can be possible and easy in the future.

Practical guide

Starting the Program

To start the program, simply run the `Main` class (after you download Maven dependencies). You will be presented with a console-like interface where you can enter commands.

Commands

This is the current list of available commands:

- `load nodes` or `l` - Load nodes from a file with a condition on nodes.
- `create edges` or `c` - Populate the graph with edges based on a condition for tuples of films.
- `visualize` or `v` - Visualize the current graph using the Graphstream library.
- `export` or `x` - Export the graph to a GraphML file (has to have a .graphml extension).
- `fetch links` or `f` - Fetch and dump film links from search parameters.
- `download` or `d` - Download and dump film data from links.
- `help` or `h` - Print the help message.
- `exit` or `e` - Exit the program.

Detailed Command Usage

Load Nodes

Command: `load nodes` or `l`

Description: Loads nodes into the network based on a specified condition.

Steps:

1. Enter a JSON string for the node condition.
2. Enter the source file name containing the film data.

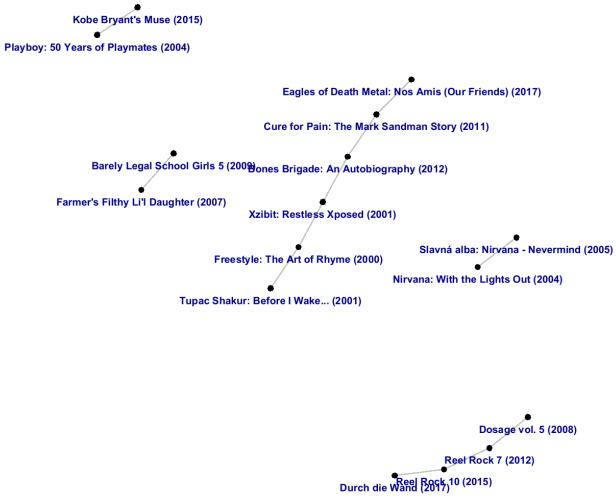
The values for the keys are always arrays of conditions. The conditions should always be inline strings. The possible fields of the condition are:

- duration: specify the length of the movie by < and >. E.g.: `"duration":[">50","<120"]` for films between 50 and 120 minutes in length
- rating: `"rating":[">90"]` for films with rating higher than 90 %
- dateCreated: specify the year. Supports = too. E.g.: `{"dateCreated":[=2018]}`
- actors: specify the actors that should or should not play in the movie. For films with Morgan Freeman: `"actors": [{"contains": "Morgan Freeman"}]`. For films without Bob Ross: `"actors": [{"notcontains": "Bob Ross"}]`. For films with Morgan Freeman and without Bob Ross: `"actors": [{"notcontains": "Bob Ross"}, {"contains": "Morgan Freeman"}]`
- directors: same as actors, but the key name is `directors`.

An example of a composite condition:

```
{"duration":[">50","<120"],"rating":[">40"],"dateCreated":["<2018"],"actors":
[{"notcontains":"Morgan Freeman"},{"notcontains":"Bob Ross"}],"directors":
[{"notcontains":"Christopher Nolan"}]}
```

We can run it on American films with an edge condition of one common actor and get a graph looking like this:



Valid Keys for Node Condition:

- `rating`
- `actors`
- `directors`
- `dateCreated`
- `duration`

Create Edges

Command: `create edges` or `c`

Description: Creates edges in the network based on a specified condition. Is very similar to load nodes.

Steps:

1. Enter a JSON string for the edge condition.

Example:

```
{ "commonActors": 2, "commonDirector": true}
```

Valid Keys for Edge Condition:

- `commonActors`, takes a number. Specifies the number of common actors between two films for them to be linked
- `commonDirector`, takes true/false. Two links are linked only if they share/do not share a director. (If there are multiple directors, then at least one of them).

Visualize

Command: `visualize` or `v`

Description: Visualizes the current network graph using GraphStream. No isolated vertices are shown - the graph would be too cluttered.

Export

Command: `export` or `x`

Description: Exports the current network graph to a GraphML file.

Steps:

1. Enter the target file name for export (e.g., `network.graphml`). The file is automatically placed in the `Exports` directory.

Fetch Links

Command: `fetch links` or `f`

Description: Fetches film links from the search results of a detailed query.

Steps:

1. Enter the search parameters.
2. Enter the timeout between calls in seconds. (Press enter for the default: 30 seconds.)
3. Enter the target file name for saving film links.

Download

Command: `download` or `d`

Description: Downloads film data from links and saves it to a target file.

Steps:

1. Enter the source file name with film links.
2. Enter the target file name for saving film data.
3. Enter the timeout between calls in seconds. (Press enter for the default: 30 seconds.)

Notes

- For more detailed information about each function, refer to the respective class and method documentation.

Exiting the Program

To exit the program, type `exit` or `e`.

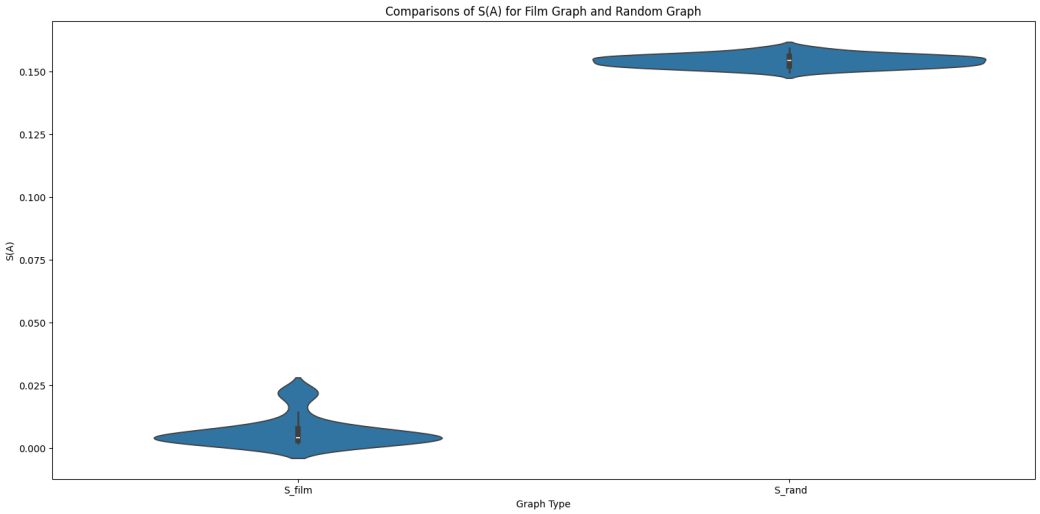
Help

To display the help message with all available commands, type `help` or `h`.

Final remarks

After exploring one of the Film Networks to Python and comparing its Approximate Symmetry to one of a random network with an equal number of nodes and edges, I confirmed my hypothesis stated in the theoretical background. Unsurprisingly, a Film Network consisting of top-rated USA Films with at least 2 common actors exhibits much higher symmetry than a completely random network. That is due to directors continuously working with a group of actors, due to long sagas, actors playing in a specific genre, etc.

Below is a plot comparing one of the versions of Approximate Symmetry, S(A), of the Film network and a random graph. The lower the S(A), the better, and it is a value scaled between 0 and 1. S(A) equal to 0 means a perfect automorphism was found.



Disclaimer

This project serves academic purposes only. Any commercial use is prohibited. I download only public data, never any user data, so I do not violate GDPR in any way.
Contact: szczjr@gmail.com