Scientific Modeling Computer Laboratory

Project: Time Evolving Networks

Final Report

Ádám Gergely Szabó

9th of May, 2022

1 Introduction

This project is about exploring MTMT's co-partnership network, which evolves in time as more publications get submitted to the site. MTMT's main goal is to host a site that maintains high quality publications, meaning the submitted works are often checked and rated for their quality. The data stored by the site is publicly available, thus data can be gathered from the site without registration.

This project's goal is to explore the co-partnership network of MTMT. In this work, we will look at this network in different given states or its subsets, see how it develops in time, calculate different central indicators, apply different group searching methods and embeddings. The work will be mostly done in Python3 language that will be utilized in the Jupyter Notebook framework.

The main goal is to analyse how the network topology changes between time steps, if possible, give reasoning to why those changes happen and understand and connect them to real life phenomenons. In this report, most progress was made towards detecting components and groups by different methods and connect them together between time steps, if possible.

2 Progress

2.1 Defined Networks Defined in Time Steps

A couple words have to said about this as it might became confusing later on. The gathered data covers publication over many years and authors can have another attribute: when they did work together. Carefully, but graphs for given time steps can be defined at least two ways: for a given time interval, and from beginning to given time. Both have significant meaning in real life: the first one tracks activity and changes, the second one follows connections made between authors. The data has a cutoff already in place: any publication above 20 is skipped. This means that people for a given publication highly likely to have contacted each other.

The two ways have different length in steps and different amount of steps: the time

intervals will be 2 years wide, while the steps until a given year will be 1 year. These seemed to be sufficient enough to be able to find connections between timesteps.

2.2 Z-score and Time Evolving Subgraph

Previously, the Z-score was defined by the following:

$$z = \frac{\langle m_i \rangle_g - \langle m_i \rangle_{rand}}{\sigma_{rand}} \tag{1}$$

where $\langle m_i \rangle_g$ is the number of times that motif i was found in the original network, $\langle m_i \rangle_{rand}$ is the average of the number of times motif i was found and σ_{rand} the deviation of the number of times motif i was found in the graph. We can do this for multiple time steps to see how the z-score, the significance of motifs change in time. (1) (2)

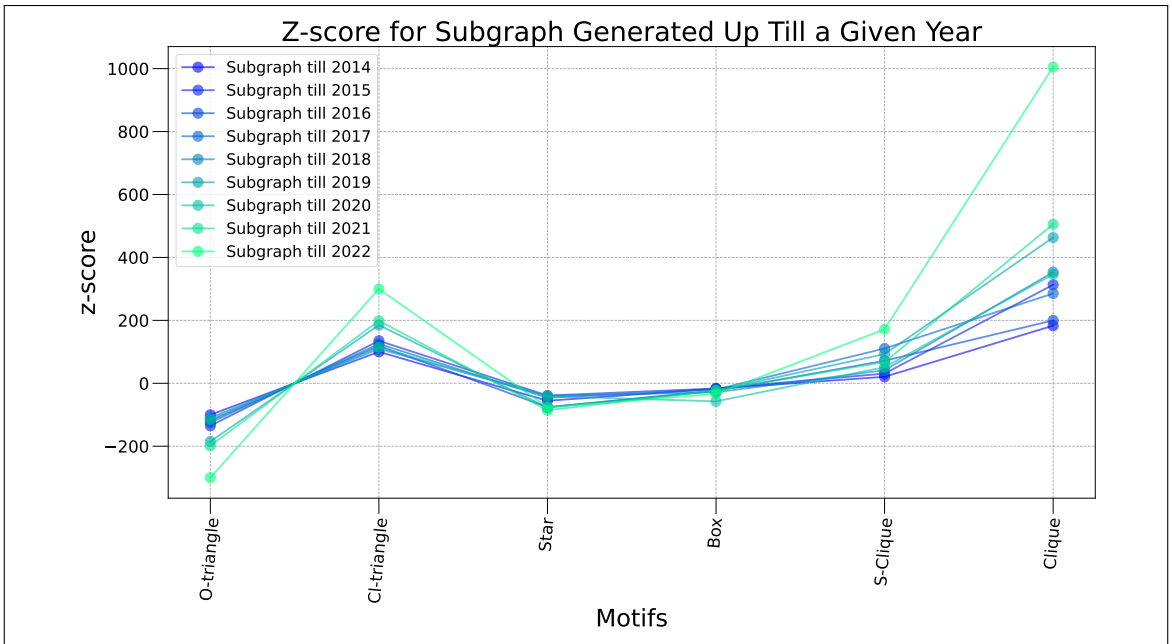

Figure 1: Z-score for subgraph evolving in time. It is visible that more highly clustered motifs gain more relevance, while the less clustered ones gain lower values. This graphs lists any connections that have been every made thus it is understandable that this change is presented.

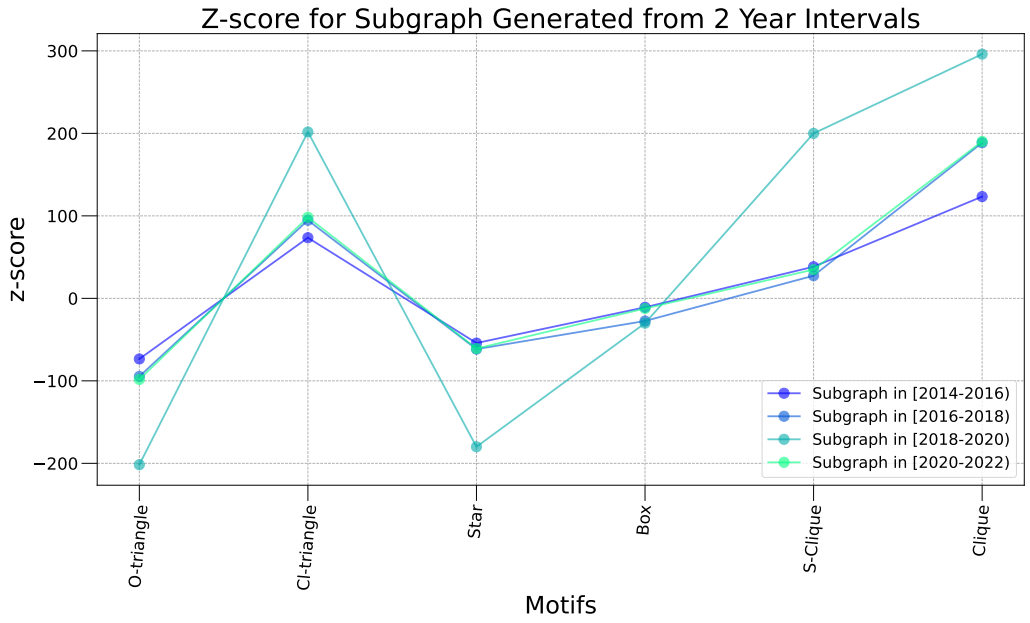

Figure 2: Z-score for subgraph evolving in time. We can see Z-score calculated from the subgraph generated for a given time interval. We this, we can track the changes between steps: from step to step it becomes higher for highly clustered motifs, while there is a drop for 2020-2022. This is caused by the incompleteness of 2022 as it has data from 3 months. The shape of the curve are similar, except for (2018-2020), which may be due to Covid-19 restrictions and quarantines.

2.3 Components in the Time Evolving Network

Identifying individual components are easy: these are sets of nodes that are separated from the rest of the graph and one could understand these as isolated groups in the network. The idea of analysing smaller components was thrown away as the size of them in incomparable to the size of the network, hence the generation of the subgraph from multiple authors as sources.

While the network itself has many nodes and edges, identifying components are fast which allows us to peek into their changes. Using these, it is easy to develop a method that tracks changes between time steps: comparing components in two given time locations, comparing how much from the previous step's components were found in the next step's components, connection can be made. With this, it is easy to define a graph from connection components together between time steps. In this case, the usage of a directed graph is advised. We can give meaning to incoming degree as the components ascendant and outgoing degree as descendants. The following events can happen to component:

- Birth (zero ascendants)
- Existence (one ascendant or one descendants)
- Merge (two or more ascendants)
- Death (zero descendants)
- Disintegration (two or more descendants)

by measuring how much of the component overlaps with its ascendants and descendants, we could define more, but these are sufficient for now. (3) (4)

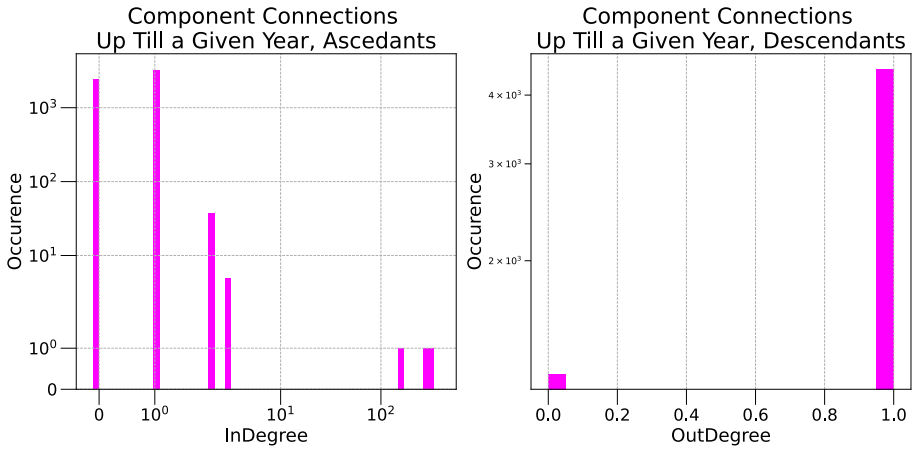

Figure 3: Component connections created from the whole network, generated from beginning up till given year and connecting components by measuring the overlap.

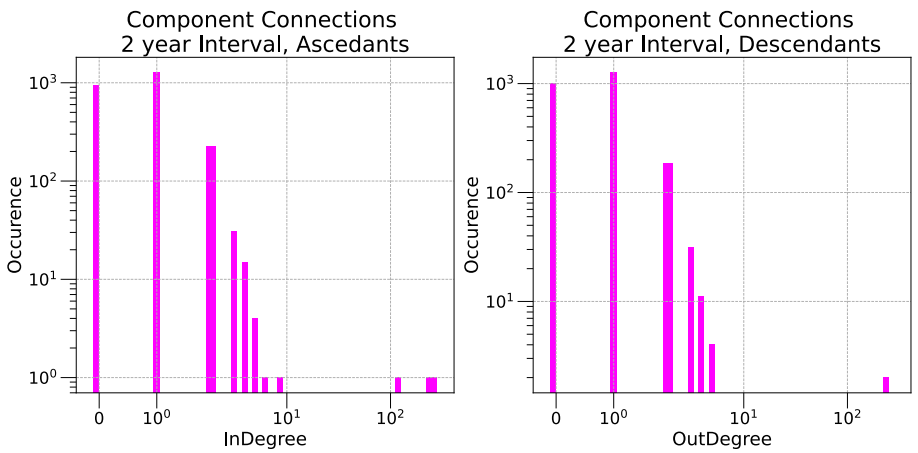

Figure 4: Component connections created from the whole network, but the network is generated from edges taken from a given time interval. In this method, all of the events can be realized from watching the indegree and outdegree

As it is shown in the figures (3) (4), we can see all events happening between time steps. But it is known from previously that the giant component (a component that

has large amount of nodes in the graph) is present in the network. What happens with that? It turns out that if we generate the network from the beginning until a given year, the giant component is a subject of constant merging. (5)

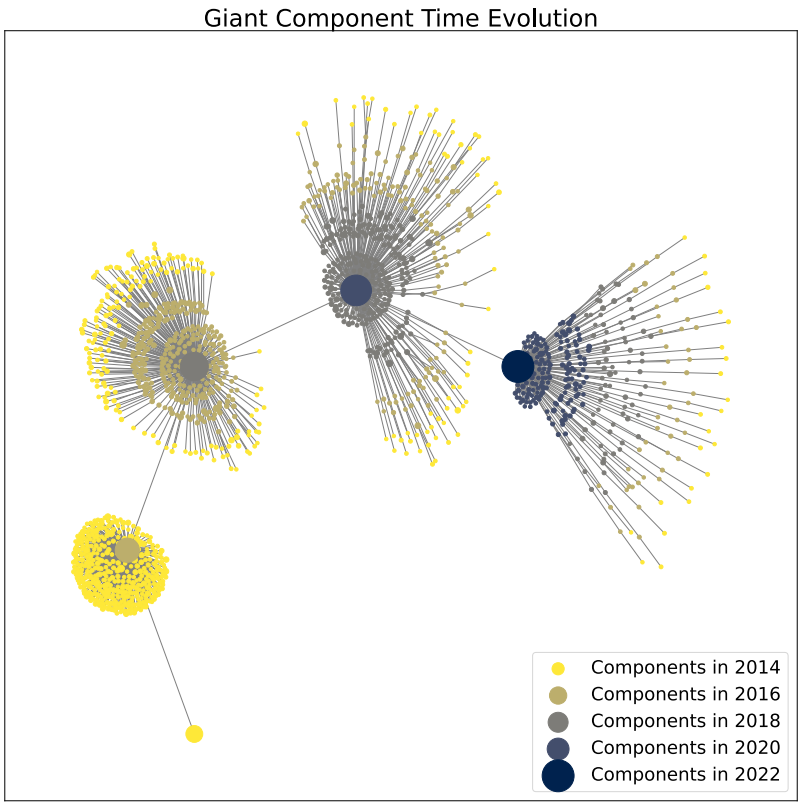

Figure 5: The adventure of the giant component generated from the network from the beginning till a given

With this momentum looking into the component size changes between time steps could give us more insight to how these components change in time and how persistent the giant component is if create the network in given time intervals.

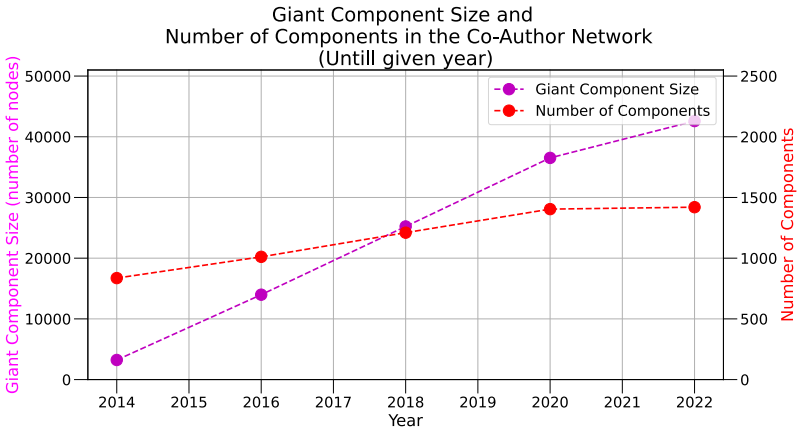

Figure 6: Giant component size and number of components changing in time.

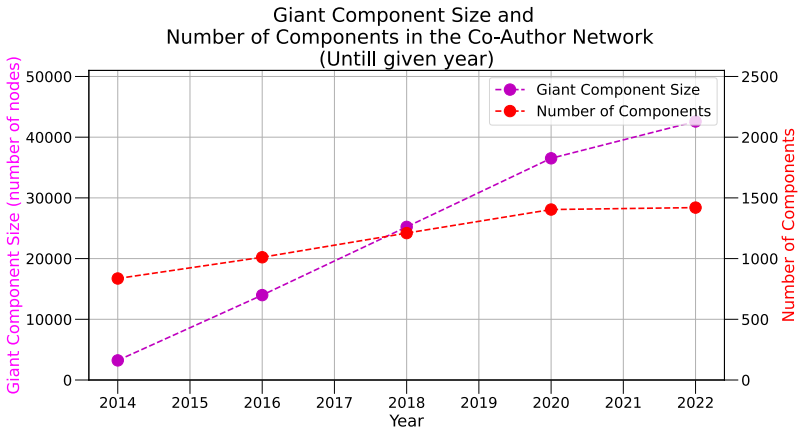

Figure 7: Giant component size and number of components for given time intervals. The incompleteness of 2022 is somewhat visible here.

At this point, a refined method is ready to be used on communities, given by community detection methods.

2.4 Community Finding

In the following, multiple community finding methods will be applied for the subgraph which was generated from multiple authors as source for neighborhood exploration. For its speed and extensiveness of iGraph, the following results were calculated with iGraph.

2.4.1 Greedy Modularity

Greedy modularity is a simple community finding method, coming from the family of hierarchical clustering methods. It starts by giving every node a community and joins two community together in each step with respect to the highest increase of modularity. Modularity (Q) for this method defined below:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{2}$$

where A is the adjacency matrix k_i is the degree of i, γ is the resolution parameter, $\delta(c_i, c_j)$ is 1 of i and j is the same community and zero otherwise.

We can go further with this:

$$Q = \sum_{c=1}^{n} \left[\frac{L_c}{m} - \gamma \left(\frac{k_c}{2m} \right)^2 \right] \tag{3}$$

Where L_c is the number of intracommunity links for community c, k_c is the sum of degrees in the nodes in community c and we iterate through over all communities c. With this, we are ready to do the community detection for the subgraph. (8)(9)

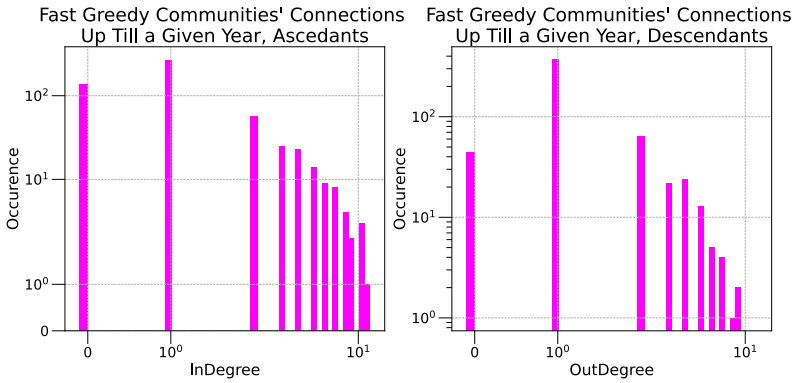

Figure 8: Connections between communities found by the fast greedy modularity method. Subgraph was generated from beginning up to given year. All defined events are visible.

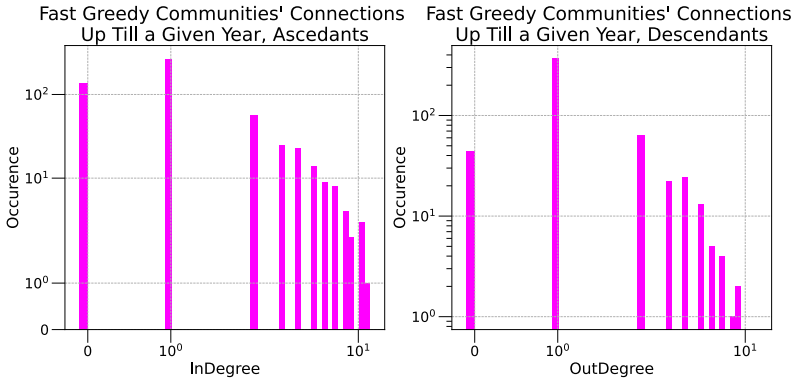

Figure 9: Connections between communities found by the fast greedy modularity method. Subgraph was generated for given time intervals. All defined events are visible.

2.4.2 Multilevel (Louvain)

Multilevel (Louvain) is a faster community finding method, which starts with the same step as greedy modularity: every node is its own community. In the first part of the step, node i is removed from its own community to its neighboring node j.

$$\Delta Q = \left(\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2} \right)^2 \right) - \left(\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2mn} \right)^2 \right) \quad (4)$$

where Σ_{in} is the sum of all wight of the links inside community i, Σ_{tot} is the sum of all the weight of the links to nodes in the community i is moving to, k_i is the weighted degree of i, $k_{i,in}$ is the sum of all weights of the links between i and other nodes in the community that i is moving to and m is the sum of all weights of all links in the network. The results are the following: (10)(11)

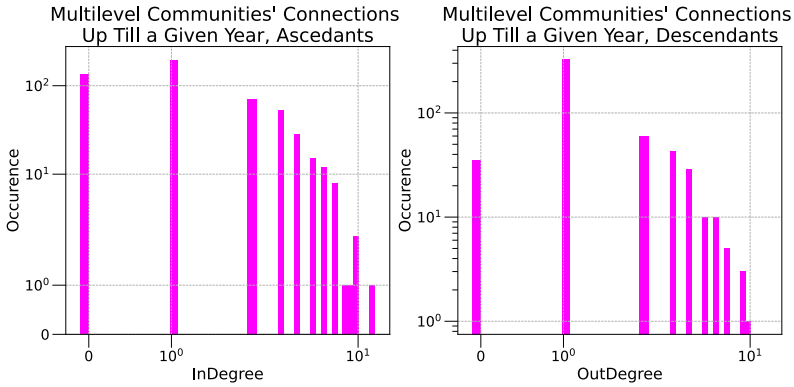

Figure 10: Connections between communities found by the multilevel (Louvain) method. Subgraph was generated from beggining up to given year. All defined events are visible.

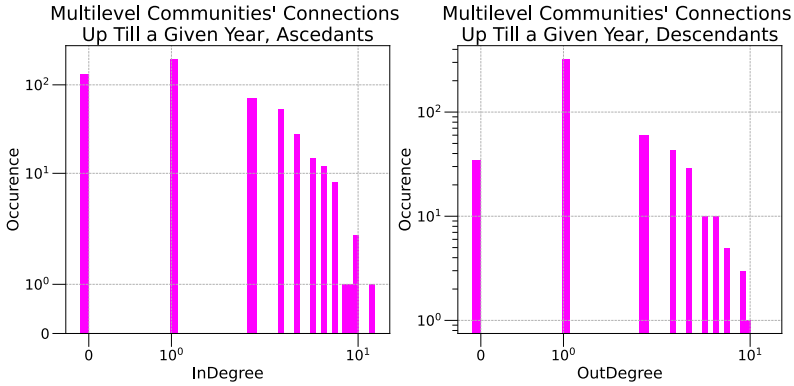

Figure 11: Connections between communities found by the multilevel (Louvain) method. Subgraph was generated for given time intervals. All defined events are visible.

These results are similar to what we have seen with greedy modularity.

2.4.3 Infomap

Infomap algorithm is different from the ones before in many ways, but the first is that it tries to minimize the cost function which means it uses community partitions of

the graph as a Huffmann code that compresses information about a random walker exploring the graph and the mapping equation and Shannon's source coding theorem. The map equation looks like the following:

$$L(M) = qH(Q) + \sum_{m=1}^{n_m} p_o^m H(P_m) \tag{5}$$

where q is the sum of exit probabilities for each community, $H(Q)$ is the average codelength of movement between communities, p_o^i is the stay probability for a random walk in community c and $H(P)$ is the average codelength for module codebook m. The results are the following: (12)(13)

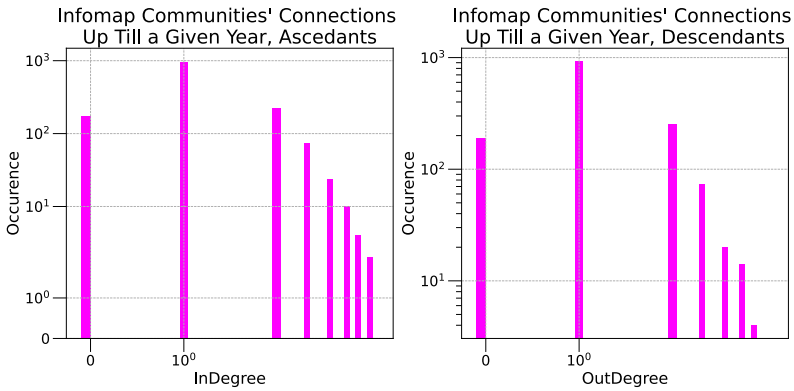

Figure 12: Connections between communities found by the infomap method. Subgraph was generated from beggining up to given year. All defined events are visible.

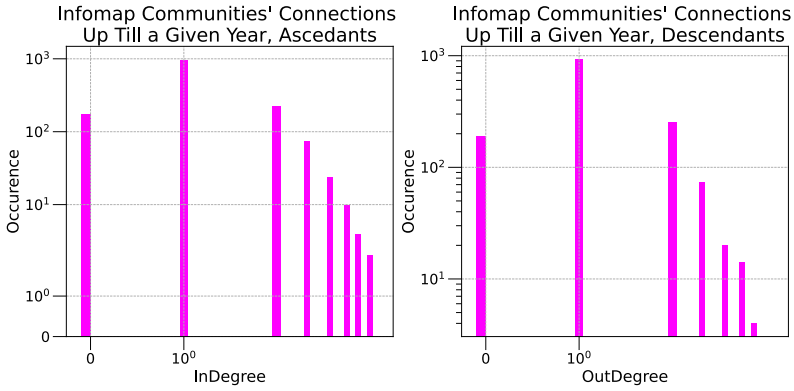

Figure 13: Connections between communities found by the infomap method. Subgraph was generated for given time intervals. All defined events are visible.

These results are much different results compared to what we seen with other methods.

2.4.4 Differences in Community Finding Methods

We seen thet greedy modularity and multilevel (Louvain) gave similar results, while infomap was somewhat different. The reason is that infomap found almost twice the amount of communities in steps compared to greedy modularity or multilevel (Louvain). (14)

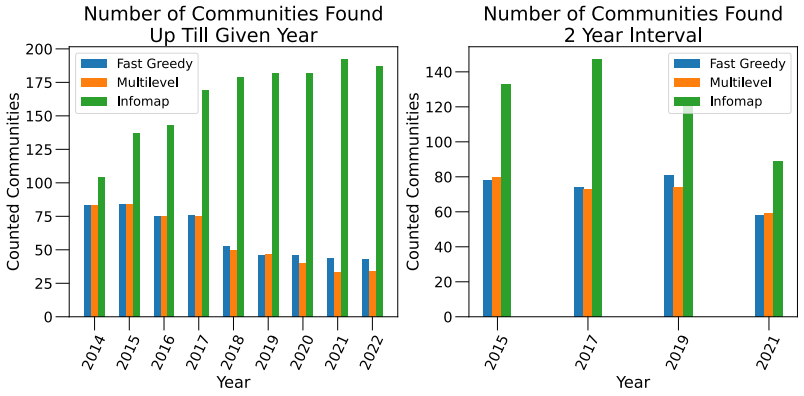

Figure 14: The number of communities found in a given time.

3 Conclusion

In this report, large step happened in regard of exploring the change in of the topology of the network. Only three community finding method were applied from the many available, but these method gave enough insight to how communities form and change if they remain active by publishing publications. Further improvement would be the usage if different networks or the whole network as a subject of community finding ass the latter community finding methods scale well, almost linearly with network size.

References

[1] Albert-László Barabási. Network science. *http://networksciencebook.com*, 2012.

[2] Aaron Clauset, M. E. J. Neumann, and Cristopher Moore. Finding community structure in very large networks. 2004.

[3] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx, in proceedings of the 7th python in science conference (scipy 2008), 2008.

[4] Xiaoming Liu, Johan Bollen, Michael L. Nelson, and Herbert Van de Sompel. Co-authorship networks in the digital library research community, 2005.

[5] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. 2009.