Technology
Arts Sciences
TH Köln

Netzwerk- und Graphentheorie

Projekt-Dokumentation

Christopher Toth
Matrikelnr: 11116218

Fabio Schlößer Vila
Matrikelnr: 11141923

Dozent: Dr. Petru Nicolaescu

July 20, 2021

Contents

1 Introduction

The project 'Oracle of GeT_RiGhT' was conceptualized and executed as part of the 'Network and Graph Theory' module at the University of Applied Sciences Cologne. The primary objective of this project was to study and apply network analysis to a real network, and to document our findings.

For our network we chose the player network within the esports game 'Counter-Strike: Global Offensive'. Our data source is Liquipedia(1), an esports Wikipedia with an extensive collection of player and tournament data spanning from the beginning of the century to today. We processed lots of tournaments to construct a graph in which the player-nodes are connected whenever they had played together in at least one tournament.

This resulted in a network with more than 2000 nodes and over 15000 edges, which we then processed further to extract important network analysis data. As the primary functionality of our project, we developed an algorithm to find paths between players and display them in the style of the 'Oracle of Bacon'(7).

2 Related Work

The 'Oracle of Bacon' website lets users find links between actors. After inputting two different names, the user is presented with the shortest path (or one of the shortest paths) between the two. Actors are connected whenever they have previously appeared alongside each other in any movie or show.

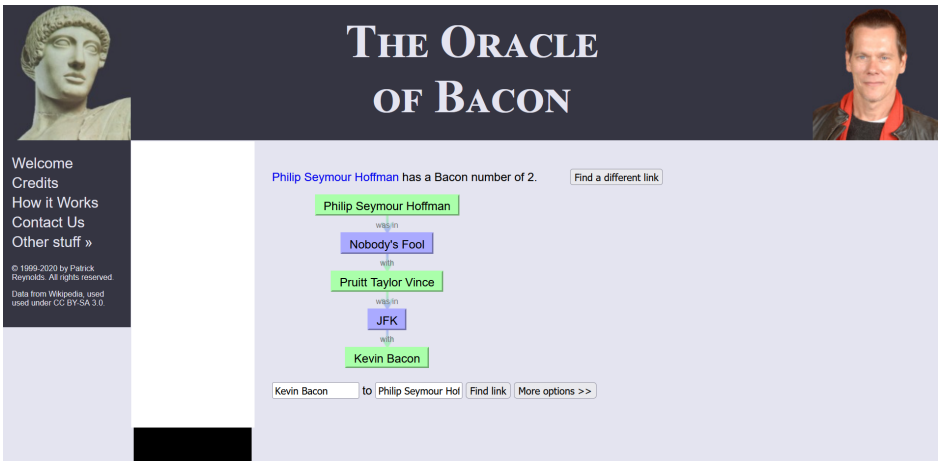

Figure 1: 'Oracle of Bacon'

This website gets its name after famous actor Kevin Bacon, for whom the developers continually calculate the average 'Kevin Bacon number, which, at the

time of writing, was at 3.170(8). This value describes how many 'hops' between individuals are (on average) necessary to get from any actor to Kevin Bacon. We took this concept and abstracted it to serve our esports purposes. Instead of actors, we have players. Instead of movies or shows, we have tournaments. And instead of Kevin Bacon, we have Christoper 'GeT_RiGhT' Alesund(4)

The second most important related work is the *liquipediapy*(3) library. It is a Python implementation to access the Liquipedia API(2). We had to modify it to fit our purposes, but some core calls remain the same and we mostly utilize their request handling to extract data from the website.

3 The 'Oracle of GeT_RiGhT'

The core concept behind the 'Oracle of GeT_RiGhT' is to create a shortest path algorithm that finds links between players and visualizes them.

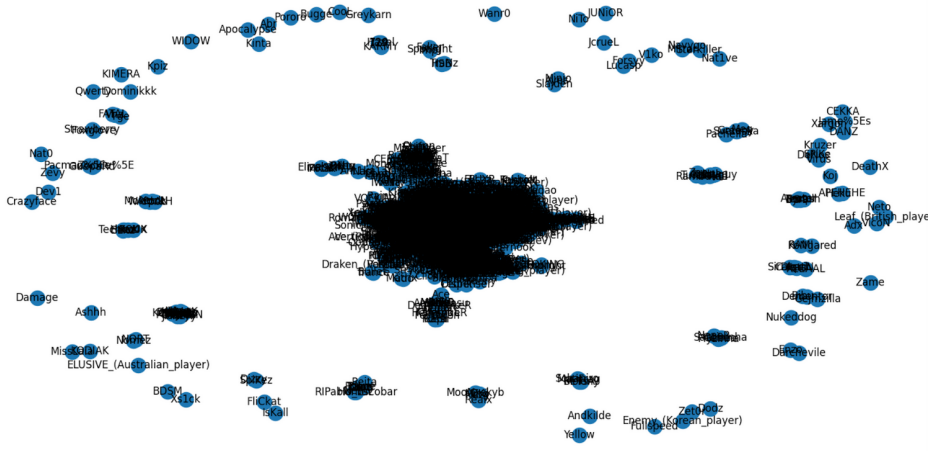

Figure 2: The CS:GO player network

Figure 2 shows our network of Counter-Strike players. There are a few outliers, but the vast majority (over 94%) of players are concentrated into one large component.

The Oracle algorithm, as we call it, can now calculate paths between individual players (provided they are connected). It will then output a shortest path and, upon user request, include the tournaments the players jointly competed in, and a visualization of the path within the network.

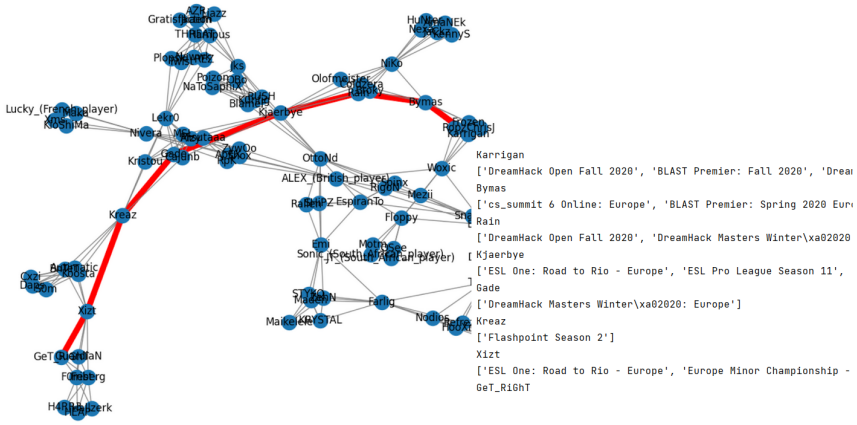

Figure 3: Path between 'karrigan' and 'GeT_RiGhT'

Figure 3 illustrates one shortest path between Finn "karrigan" Andersen and GeT_RiGhT. The left shows our algorithm's visual output, while the box on the right shows the textual output. Note that, for this example, we chose to only include player data from 2020.

This is the core functionality of the 'Oracle of GeT_RiGhT'. The visual and tournament outputs can be turned off and a tie strength parameter defines how many tournaments a pair of players has to have competed in together in order to be considered linked.

4 Implementation

Our implementation is Python based and lacks a graphical user interface. However, the functionalities are easily accessible via designated methods. Before we delve into our implementation, we need to lay down the basic terminology that guides our understanding of the CS:GO competitive scene and has thus informed our concept for the 'Oracle of GeT_RiGhT'.

4.1 Terminology

Player A player is a singular individual who has competed in at least one Counter-Strike tournament, or tournaments of the game's respective sequels (e.g. Counter-Strike: Source and Counter-Strike: Global Offensive). This player has to have a Liquipedia entry, which provides us with a unique page ID from Liquipedia. We can use this to identify them even when that specific player appears with different nicknames throughout their tournament history.

Team For ease of use, we have stuck with the term 'team' throughout this document. The word might be ambiguous in the sense that some might associate it with the organization (for instance, the team 'Evil Geniuses' referring to its management and corporate structure). We, however, only use it to reference a specific lineup at a given tournament. At MLG Columbus 2016, Vincent "Happy" Cervoni Schopenhauer and Nathan "NBK-" Schmitt competed in a team, just like Danylo Ihorovych "Zeus" Teslenko and Ioann "Edward" Sukhariev(5). A group of players counts as a team whenever these individuals have stepped onto the server alongside each other, which also includes showmatch lineups. This is something that we will discuss in length later on.

Tournament A tournament is an event at which players have competed against each other in teams. Our network constructs links between every team's players at that specific tournament.

Tournament Tier The tournaments on Liquipedia are sorted into various categories reflecting their scale and importance for the scene. While they have a comprehensive set of rules that categorize each and every tournament, we want to broadly summarize the tournament 'tiers' that we are making use of in this project:

- S-Tier: Largest events, professional players in attendance
- A-Tier: Large events, professional and semi-professional players in attendance
- B-Tier: Smaller events, semi-professional and professional players in attendance

These descriptions obviously don't cover the exact specifications for the tournament tiers, but should provide a rough understanding.

4.2 Important Methods and Classes

In order to keep this project documentation fairly short, we are only going to briefly touch on the classes and methods we have implemented. The code is structured into separate classes for the Algorithms, the API implementation, the analyses we conducted, and the main function we use to test and execute everything.

The code is accessible via *https://github.com/f4nz0/oracle_of_getright* and open to everyone for inspection or duplication.

Classes and Files The algorithms we implemented can be found under *Algorithms.py* and *VisualizeCommunities.py*. We chose to arrange them separately from the rest of the code. *CSGOAPI.py* and *liquipediapymod.py* contain modified *liquipediapy* library contents since we struggled with acquiring the information we needed through the base functionalities provided there. *Helpers.py* is an accumulation of background methods we used to extract or write JSON data, to convert between between player IDs and names, and to generate the network out of the JSON database. *Analysis.py* is a collection of analyses we conducted to test our various hypotheses on the dataset. Finally, *APITest.py* is a simple main method where the user can execute their own tests.

Methods The core functionality of the 'Oracle of GeT_RiGhT' can be accessed via a call to *Algorithms.oracle_algorithm*. The method requires the user to input a graph, labels and weights which have previously been generated by *Helpers.network_from_json*. The network generator takes the start date, end date, and tournament tier as optional arguments. This enables the user to filter for tournaments (and thus player links) that have only taken place between two cutoff dates, as well as limit the data to S-Tier, S- and A-Tier, or all tournaments in the database. The Oracle algorithm receives optional arguments to enable visual output and set the minimum tie strength (i.e. tournaments players jointly competed in).

Users can print out basic information on their network via *Algorithms.print_metadata*. This already provides data on the number of player nodes, links, percentage of players in the largest connected component, and the average shortest path within the network.

Our project includes a variety of network analysis algorithms which we will not delve into with this documentation. Enthusiasts are welcome to try them out themselves and take cues from the example usages provided in our own analyses.

5 Analysis of the CS:GO Network

After creating and assembling the various tools for our project, we set out to utilize them and demonstrate their functionalities as part of this documentation. While we have already shown the 'Oracle of GeT_RiGhT' to work, the remaining algorithms can yield interesting results as well.

5.1 Hypothesis 1: Bridges Between Male and Female Players

'There are only few connections between male and female players. The player 'juliano' is an important connecting piece, whose absence from the network would break the communities apart.

We initially came to form this hypothesis after we observed Julia 'juliano' Kiran appearing in all Oracle algorithm paths between male and female players that we tested. Our general idea was that, since she had played alongside Kenny "KennyS" Schrub at the MLG Major Showmatch in 2016(5), she was the fastest (and maybe the only) link between male and female players. We then tested whether the bridge algorithm from the NetworkX Python library(6) would output her as a bridge.

However, since the algorithm only considers individual edges and juliano has played with multiple men and women, none of her individual edges could have been removed to break the components apart. This gave us another idea. What if we simply removed her from the graph altogether?

This led us on a wild chase to find all links between male and female players. After we had removed juliano, others sprung up and even after omitting six players, we still found new paths. This obviously proved our initial hypothesis wrong. But we still wanted to know whether juliano was in fact important for connecting the male and female scenes.

It turns out that she actually is a vital link. Removing her from the graph meant an average shortest path (ASP) increase from 4.524 to 4.591. Even the removal of GeT_RiGhT only resulted in an increase to 4.535. This goes to show that she is a useful link between the two scenes. As an additional example, we tested how the path between Meyssa "Missa" Bellouati and GeT_RiGhT changed after removing juliano from the graph. It turned out to increase from four to seven steps!

5.2 Hypothesis 2: Average Shortest Path in 2014 and 2020

'In 2014, the average shortest path between players was higher than in 2020 due to the network expanding and international teams introducing shorter paths.'

This hypothesis was fairly easy to test. We simply had to take a look at the average shortest path in the CS:GO network from 2014 and the network from six years later.

```
ASP in 2014: 4.626845931026945
ASP in 2020: 4.613262951665994
Difference in %: 0.2935688709638118
ASP in 2014 (S-Tier): 4.419033517297079
ASP in 2020 (S-Tier): 4.092727501746269
Difference in % (S-Tier): 7.384103656909935
```

Figure 4: 'Decrease in Average Shortest Path 2014-2020'

We found the results from Figure 4 to not be as expected. A 0.29% decrease in the ASP was far below any figure we would have imagined. However, we discovered that the low amount of change was mostly driven by a large number of unconnected or barely connected components of low-level competitors. When we narrowed our search down to just the S-Tier of tournaments, the ASP decreased considerably. Here, we observed a 7.38% decrease, which goes to show that, at the very least, the network of high-profile players has grown together more.

5.3 Hypothesis 3: National and International Communities in 2015 and 2020

'In 2015, players were mostly split into national communities, whereas the communities in 2020 are international.'

To test our hypothesis, we implemented a few algorithms from NetworkX(6) to find cliques and communities in our network.

- greedy_modularity
- Girvan_Newman
- Louvain
- clique_percolation
- find_largest_clique
- small_world_coefficient

In 2015, teams were most likely assembled from players from the same region. Therefore the communities found by community algorithms should in each case consist of players from the same region. We expect that the communities found in 2020 are not clustered in national communities anymore because the CS:GO scene has somewhat shifted to have lots of multinational teams. In addition, the players already competing in 2015 will likely have created new links as they have switched teams since then.

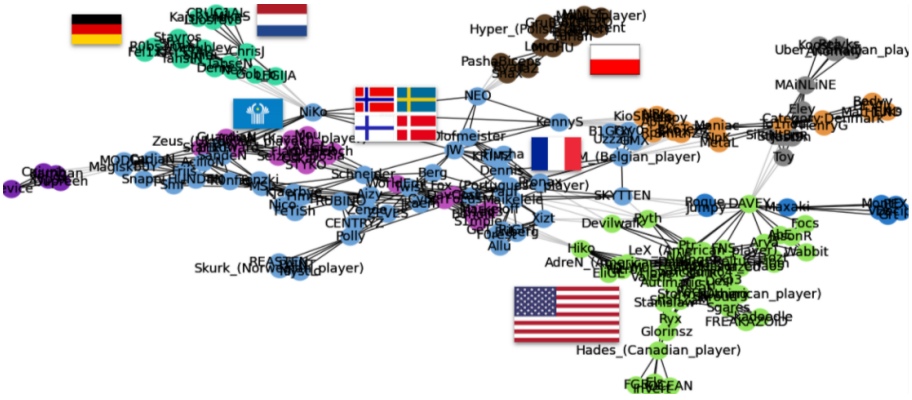

Figure 5: 'Communities 2015'

Figure 5 shows the visualization of the CS:GO network in 2015 with different colors for each community found. The country flags illustrate that the communities are in large parts national communities.

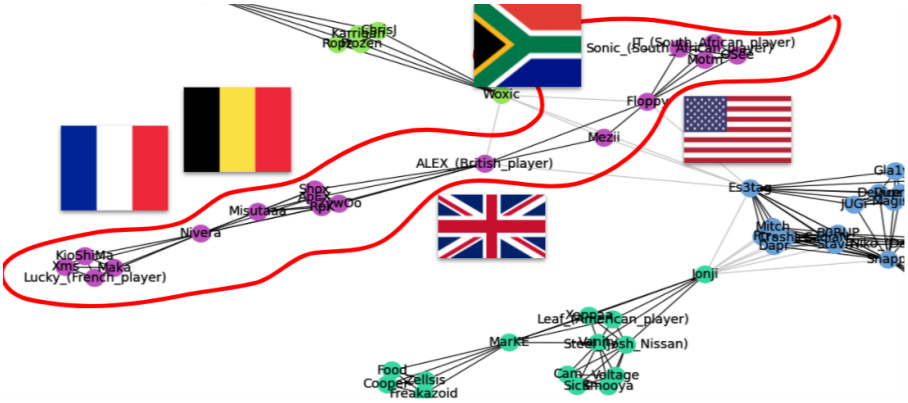

Figure 6: 'communities 2020 zoomed in'

Figure 6 shows the network in 2020. To provide a better visual example, we zoomed into the violet-colored community and set flags near the players to point out that the community consists of different nationalities - five in this case. There are many communities like this one where the color does not at all correlate with the players' nationalities.

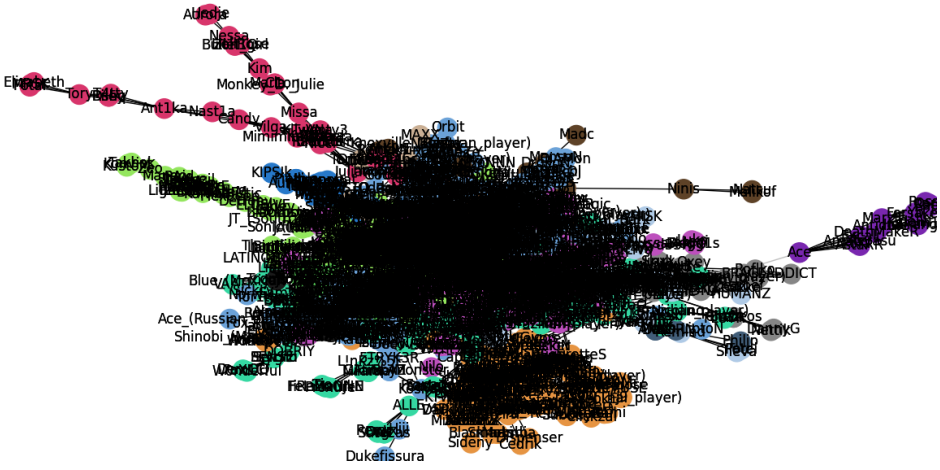

Figure 7: 'Greedy_Modularity of 2015-2020'

Figure 7 shows the network from 2015 to 2020. The big component is highly connected, as can be seen in the zoomed plot in Figure 8, but still there are many communities. For instance, we can observe the female community in the upper left corner, the nodes visualized in red.

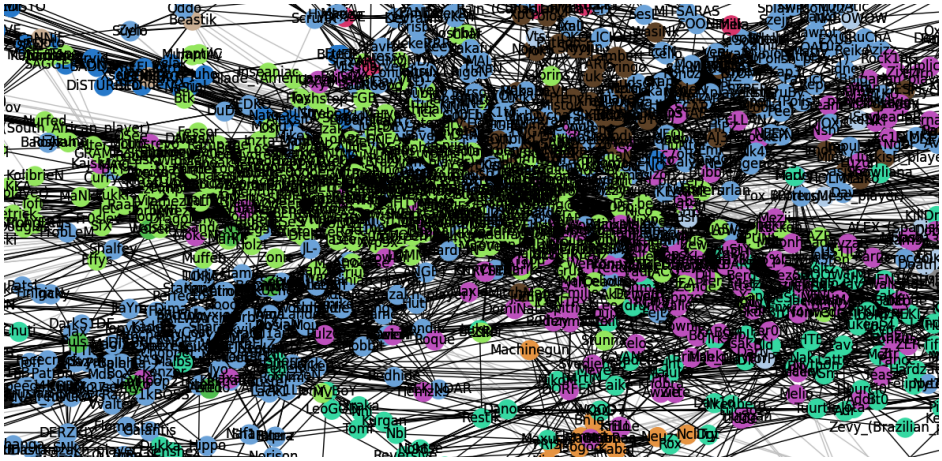

Figure 8: 'Greedy_Modularity of 2015-2020 zoomed in'

We also used the small-world-algorithm on the big component of the 2015 network to check if the network has small world characteristics. The results of sigma(8.168844 $>$ 1 = small world) and omega(-0.388746 near 0 = small world) showed that the network in 2015 is a small world.

The plotted networks and the results of the algorithms were as expected. Over the years, national communities have grown to be international communities and the CS:GO scene is a big component with a few outliers.

6 Future Work

The concept behind the 'Oracle of GeT_RiGhT' has obviously not reached its natural conclusion upon the completion of this university project. Most areas are still in development and can be extended further. For instance, the tools and methods available are only accessible via the Python project itself. For end users with no experience in programming, it's virtually impossible (or at least much too tedious) to use the 'Oracle of GeT_RiGhT'.

So if this project were to be continued, our first and highest priority would be building a web interface similar to the 'Oracle of Bacon' that served as inspiration for our concept. This would include the visualizations and provide drop-down menus and auto-fill options for player names.

Furthermore, the dataset available does not cover the entire CS:GO scene or even the data available on Liquipedia. Due to time constraints, we were only able to lift data on S-, A-, and B-Tier tournaments. We are still missing out on C-Tier events and Qualifiers, which might actually connect a lot of the unconnected components we have observed in our final graph.

Currently, there is also no automated way to lift data off of Liquipedia. This means that we manually had to insert the URLs to the different tournament lists (S-Tier, A-Tier, ...) and then let our modified liquipediapy handle the API requests. It would obviously beneficial to continually check for new tournaments and insert those into the database in real-time, enhancing the accuracy of the 'Oracle of GeT_RiGhT' and keeping it up to date.

The project can also be extended with additional network analysis algorithms. We have only provided a superficial set of tools to engage with the CS:GO player network. The networkx library offers a lot more and new algorithms could easily be integrated into the existing code base.

Finally, we failed to include nationality data in the dataset. This would have helped us create an algorithm to actually calculate the correlation between communities and nationalities for different tiers and time-spans, which is something we unfortunately had to do visually and by hand in our analysis. Redownloading the dataset and augmenting it with nationality tags for every player would greatly enhance the information we can extract from the communities.

7 Conclusion

The 'Oracle of GeT_RiGhT' succeeds in more than just its basic functionality. We provide a robust algorithm seeking links between players, one which can also visualize the shortest connection path. In addition to that, it takes input as to how many tournaments players have to jointly participate in to be considered 'connected'.

The database - while incomplete - includes enough information to filter for lots of different player and tournament attributes. While we settled to not include coaches and substitutes, the network assembly algorithm lets users generate their graphs whichever way they prefer.

We have extended the core functionality to allow for metadata output of the network, as well as a variety of network analysis tools that can be easily applied to the generated player graph. We have proven these tools to work by conducting our own research on the graph, yielding interesting and insightful results on the way.

While the database remains incomplete and the project would benefit from additional network analysis tools, we consider this project to be a success and will leave it for future researchers to expand and complete.

References

1 Liquipedia. `https://liquipedia.net/`.

2 Liquipedia API. `https://liquipedia.net/api-terms-of-use`.

3 Liquipedia API implementation in Python. `https://github.com/c00kie17/liquipediapy`.

4 Liquipedia-Entry for GeT_RiGhT. `https://liquipedia.net/counterstrike/GeT_RiGhT`.

5 Liquipedia-Entry for MLG Columbus 2016. `https://liquipedia.net/counterstrike/MLG/2016/Columbus`.

6 NetworkX: Network Analysis in Python. `https://networkx.org/`.

7 The Oracle of Bacon. `https://oracleofbacon.org/`.

8 The Oracle of Bacon, Kevin Bacon Number. `https://oracleofbacon.org/onecenter.php`.