Analizing a card game: solving Magic Commander's metagame using centralities

Milanello Tommaso, Ruta Matteo, Tokayev Dmytro
Github: https://github.com/matteo-ruta/EDHNet

1 Motivation

Trading card games can be very complex, and *Magic the Gathering* is no exception[1]. This kind of game requires the players to apply not only during the match but also while building their deck because it affects how the player should behave in the game itself. During its history, the game has evolved into many different modalities due to its malleability and multiplicity in terms of strategies and game plans. These modalities, called *formats*, introduce limitations about legal cards, deck-building and sometimes even additional rules to the "core" of the game.

One of the most popular formats nowadays is Commander. Created by the community, this format has gained a lot of popularity, especially in the last ten years. The spirit of Commander has always been to give the players as much freedom as possible, making the "state of the format" (also called *metagame*) to become the most heterogeneous in the whole of *Magic*.

The goal of this project is to investigate this variety. We are focusing on finding very important cards for the format, which are more efficient or versatile than others. In this way, we hope to get a reliable definition of the power level of a card (a problem that particularly affects Commander) which can be used to monitor the "health" of the format, or simply help players deck-building more consciously. We plan to do this by the use of graphs, more precisely by computing centralities for each card (represented by nodes) and cluster coefficient, and then use them to cluster the data hoping to get some significative results.

2 Dataset description

Our data is organized as an undirected weighted graph with $|V| = 18647$ nodes and $|E| = 7681274$ edges, where each node represents a card appearing in at least one Commander's deck, and each edge represents a connection between cards that have been played together in the same deck at least once. Each node in the graph contains the name of the card it represents, so that we are able to interpret the results with the actual game statistics. Each edge is weighted with the number of decks in our dataset where the two cards were played together. In other words, edges encode the information about the metagame.

We retrieved a big collection of decks via web scraping from EDHRec. Due to the impossibility of getting the whole dataset, we first extracted in-game information about the most played decks, then we used that information to cluster them and extract a coreset. In order to do this, we first retrieved from EDHRec the list of the 600 most played commanders by the community (in this format, each deck has a special card, called commander, which is a central part of the strategy). We clustered them based on some in-game characteristics, obtaining 100 sampled commanders. Then we downloaded the decklists proportionally to their real presence in the meta, as reported from EDHRec. We decided to remove from the deck collection each basic land (a very common and uninteresting type of card) and each commander, to reduce skew.

The idea behind the sampling is to maintain a sort of representativity for our sample. However, it's clear how having the complete EDHRec database (or some similar) would have better represented the state of the game.

2.1 Edge weights

We used inverted weights for computing our scores. This is due to the meaning of our weights, which is more similar to the "frequency" rather than the "cost". In this way, the algorithms would have considered as shortest paths the most "frequent" ones, so the ones with higher weights. We adopted two different approaches to his purpose: first, we tested our scores with inverted weights, then we did the same but adopting the following function as the "inversion operator"

$$inv(x_i) = max\{x \in X\} + 1 - x_i \tag{1}$$

where x_i here represents the weight for a given edge i and X is the set of edge weights.

In this way we swapped the values on the edges, making the most frequent the thinner ones and vice versa. With this approach, we were able to bound the numbers inside the edges, also avoiding the use of meaningless real numbers for our representation.

3 Theoretical Background

In order to understand the experiments presented later in this paper, it is essential to introduce some theoretical background. This section will focus on key concepts in node analytics. Node analytics refers to the study of the roles and properties of individual nodes within a network. The following measures will be discussed: closeness centrality[2], betweenness centrality[2], PageRank centrality[3], and local clustering coefficient[4]. All the definitions that follow will consider an undirected graph with weighted edges.

3.1 Closeness Centrality

Closeness centrality measures how close a node is to all other nodes in a network. It is defined as the fraction of the number of nodes minus 1 and of the sum of the shortest path distances from a given node v to all the other nodes:

$$C_C(v) = \frac{n-1}{\sum_{u \in V, u \neq v} d(v,u)} \tag{2}$$

where:

- $d(v,u)$: the shortest path distance between nodes v and u,
- V: the set of all nodes in the graph.

Higher closeness centrality indicates that a node can reach other nodes more quickly.

3.2 Betweenness Centrality

Betweenness centrality quantifies the extent to which a node lies on the shortest paths between other nodes. It is given by:

$$C_B(v) = \sum_{s \neq v \neq t, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3}$$

where:

- σ_{st}: the total number of shortest paths between nodes s and t,
- $\sigma_{st}(v)$: the number of those shortest paths that pass through node v.

Nodes with high betweenness centrality often serve as bridges or critical intermediaries in the network.

3.3 PageRank Centrality

PageRank centrality measures the importance of a node based on its connections and the importance of its neighbors. It is defined iteratively as:

$$PR(v) = \frac{1-d}{N} + d \sum_{u \in \mathcal{N}(v)} \frac{w_{uv} \cdot PR(u)}{\sum_{z \in \mathcal{N}(u)} w_{uz}}$$

where:

- d: damping factor, which accounts for the probability of following a random edge versus "teleporting" to any node,

Table 1: Algorithms used for experiments

Metric	Algorithm	Completeness	Computing time
Closeness Centrality	Dijkstra's algorithm	Complete	13 minutes
Betweenness Centrality	Brandes' algorithm	Complete	17 minutes
PageRank Centrality	Power Iteration Method	Complete	1 second
Clustering Coefficent	Zhang-Horvath	Complete	1 minute

- N: total number of nodes in the graph,
- $\mathcal{N}(v)$: the set of nodes adjacent to v,
- w_{uv}: weight of the edge between nodes u and v,
- $\sum_{z \in \mathcal{N}(u)} w_{uz}$: sum of weights of edges connected to node u.

PageRank centrality assigns higher scores to nodes connected to other highly-ranked nodes.

3.4 Local Clustering Coefficient

The local clustering coefficient of a node measures the degree to which its neighbors form a complete subgraph (i.e., are interconnected). It is defined as:

$$C_L(v) = \frac{\sum_{u,z \in V} \mathcal{A}_{vu}\mathcal{A}_{uz}\mathcal{A}_{vz}}{\sum_{u \neq z, u,z \in V} \mathcal{A}_{vu}\mathcal{A}_{vz}} \tag{4}$$

where:

- $\mathcal{A}$: the weighted adjacency matrix.

The local clustering coefficient is the fraction of triangles containing v among all the potential triangles containing v.

4 Experiment results

Our initial approach considered betweenness centrality, closeness centrality and local clustering coefficients of the nodes to project each card into a continuous space of three components. In our expectations, the obtained embedding would have been used to find clusters among the cards, hoping to extract the desired information. This idea presented a few problems, mainly related to the data and on how they satisfied our intent. We are presenting them in the following sections, explaining how they influenced our final work. The clustering has been modelled as a k-means problem.

All our operations on graphs were implemented using graph-tool[5], while the clustering was done using scikit-learn[6].

4.1 Algorithms

In our experiments we opted for the implemented versions of the algorithms in graph-tool. Algorithms and their respective computing times are reported in Table 1.

4.2 Local Clustering Coefficient

We analyzed the local clustering coefficient for each node, and we compared it with the closeness. Our results were very disappointing: based on our interpretations, a very central node should have had an high local clustering coefficient, due to the massive amount of triangles in its neighborhood. This would have helped to understand how easily a card can be used in relation to other two cards, allowing interesting conclusions regarding the concept of "strategy" in the game. Instead, what we found was the opposite trend. Very poor and unknown cards resulted to have the highest local clustering coefficient, highly separating the two categories. After realizing this, we decided to try another centrality score (PageRank) to embed the data. The interpretation we gave to PageRank depended on the interpretation we gave to a path in our graph.

A path in our graph represents a chain of synergic cards, in the sense that two consequent nodes saw play together. PageRank could have encoded how often a card appeared in a random chain of these, so similarly in a random deck.

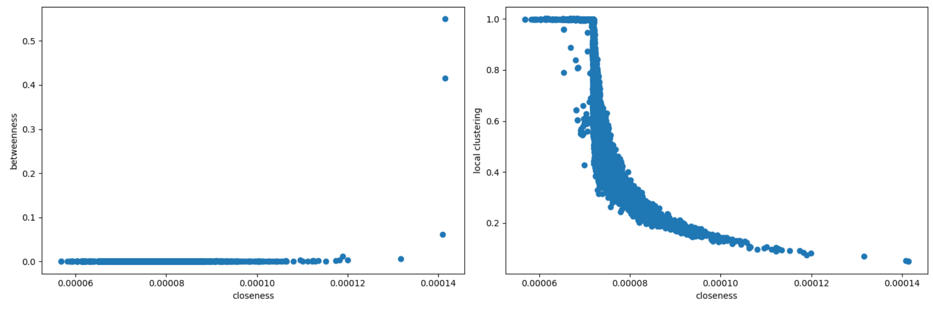

Figure 1: On the left, relation among betweenness and closeness.
On the right, relation among local clustering coefficient and closeness

4.3 Betweenness centrality

We expected to obtain an almost uniformly distributed set of data from the embedding, so that the clustering output would be easy to interpret. Instead, our data was "bad shaped", in the sense that one specific indicator (the betweenness centrality) was very skewed towards low values. The only few exceptions (see Figure 1) where points with the highest value possible for both the other centrality scores. We decided therefore to remove this component from the embedding, considering it only as redundant information, which wasn't adding anything useful for the vast majority of the nodes in the graph.

5 Analysis and Conclusion

It's interesting to briefly explore what we found "on top" of our embedding space, in the sense which cards appears to be central in the *metagame*. Looking at the clustering we obtained, one cluster contains cards with the highest scores on both the centralities. This cluster contains fundamental cards for the format, like "Command Tower", "Arcane Signet" and the massively abused "Sol Ring", each of one is a real symbol of the format. Finding these, anyway, was the easy part, since they are almost played in any deck. They kinda make up the basement of Commander's deck-building. In Table 2 we reported all the cards contained in this cluster and also confronted their scores with the data on EDHRec.

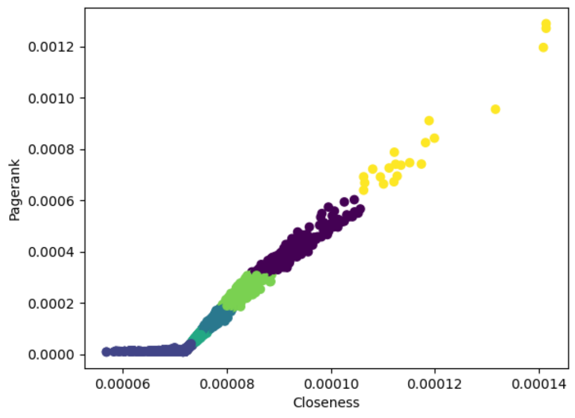

Figure 2: Plot of the clustering on the embedding space

Based on the data contained in Table 2, we can say that the embedding captures different information than the statistics on EDHRec. Exploiting the structure of the graph, our results gave a lot of importance

Table 2: Comparing of cards' scores in the best cluster with their presence on EDHRec

Card name	**Closeness (% of max)**	**PageRank (% of max)**	**EDHRec Presence***
Sol Ring	1.414e-04 (100.00%)	1.289e-03 (100.00%)	84%
Command Tower	1.414e-04 (100.00%)	1.271e-03 (98.59%)	74%
Arcane Signet	1.409e-04 (99.63%)	1.196e-03 (92.80%)	72%
Swords to Plowshares	1.174e-04 (83.03%)	7.421e-04 (57.57%)	63%
Path to Exile	1.101e-04 (77.88%)	6.644e-04 (51.54%)	46%
Cultivate	1.135e-04 (80.25%)	7.376e-04 (57.22%)	46%
Counterspell	1.095e-04 (77.45%)	6.920e-04 (53.68%)	43%
Exotic Orchard	1.317e-04 (93.11%)	9.556e-04 (74.12%)	36%
Rampant Growth	1.063e-04 (75.17%)	6.402e-04 (49.66%)	35%
Farseek	1.121e-04 (79.30%)	6.730e-04 (52.20%)	34%
Reliquary Tower	1.182e-04 (83.59%)	8.256e-04 (64.04%)	30%
Swiftfoot Boots	1.125e-04 (79.52%)	7.414e-04 (57.51%)	28%
Lightning Greaves	1.128e-04 (79.74%)	6.953e-04 (53.94%)	25%
Evolving Wilds	1.189e-04 (84.08%)	9.113e-04 (70.68%)	21%
Path of Ancestry	1.199e-04 (84.81%)	8.429e-04 (65.38%)	21%
Fellwar Stone	1.152e-04 (81.44%)	7.473e-04 (57.97%)	21%
Rogue's Passage	1.065e-04 (75.32%)	6.690e-04 (51.89%)	20%
Myriad Landscape	1.063e-04 (75.18%)	6.925e-04 (53.71%)	19%
Terramorphic Expanse	1.122e-04 (79.35%)	7.883e-04 (61.14%)	16%
Commander's Sphere	1.081e-04 (76.41%)	7.225e-04 (56.04%)	15%
Chromatic Lantern	1.112e-04 (78.64%)	7.268e-04 (56.38%)	10%

* Percentage refers to the total number of decks that could run that specific card.

to cards like "Fellwar Stone" and "Chromatic Lantern", which can be played potentially in every deck (due to the deck-building restriction of Commander, which puts a limitation on the *colours* of cards you can play based on your commander). In this way, we are exploring the meta not simply by looking at how much a card is played, but also how much a card could potentially be played, since the more a card can be played with any Commander, the more its centralities will grow.

We noticed this trend even in other clusters, and even for very particular types of cards. One widespread type of card in Magic is the *land* type, which represents the basic resource for the actions in the game. A lot of lands were situated in high clusters, even if their effect was relatively weak. We interpreted this fact as another sign of the importance for our embedding of the connections between the cards: the more a card is connected with others (and lands are very connected, since they represents the base of the game), the more it accumulates importance.

So in conclusion, what we obtained with this experiment is a synthetic representation of "importance" classes for Commander's metagame. We was able to extract very used cards and very versatile cards in the cluster with the most important points, which can be considered a sort of "core" for the format. What we wasn't able to get, instead, is a clear class of very strong cards, which goes beyond the basic aspect of the game and that increases the power level to the state-of-the-art. In order to achieve this result, probably a GNN would have better fitted our case-study, allowing us to extract more complex information from the nodes.

Table 3: Group contributions distribution

Member	Code section	Fraction of work
Milanello Tommaso	Implementation of embedding part, clustering	around 30%
Ruta Matteo	Data scraping, transformation of data into a compatible format, plotting sections for visual analysis	around 35%
Tokayev Dmytro	Computation of scores/centralities, organization of the graph object, analysis of performances	around 35%

Table 4: Hardware used for experiments

Member	CPU	GPU	RAM
Milanello Tommaso	Intel(R) i7-1280P @ 2.00 GHz	NVIDIA GeForce GTX 1650	16GB DDR4
Ruta Matteo	Intel(R) i7-10510U @ 1.80GHz	NVIDIA GeForce MX250	16GB DDR4
Tokayev Dmytro	Intel(R) i5-8300H @ 2.30GHz	NVIDIA GP107M	16GB DDR4

Group member contributions

During the project, we always discussed together ideas and intuitions and came up together on a way to put them in practice. Here we report how we divided the work in terms of code sections. In the work ratios are considered also all the hours spent together discussing.

See Table 3 for details about group contributions distribution.

See Table 4 for details about hardware used by group members.

References

[1] Alex Churchill, Stella Biderman, and Austin Herrick. *Magic: The Gathering is Turing Complete.* 2019. arXiv: 1904.09828 [cs.AI]. URL: https://arxiv.org/abs/1904.09828.

[2] Linton C Freeman. “Centrality in social networks: Conceptual clarification”. In: *Social Networks* 1.3 (1979), pp. 215–239. DOI: 10.1016/0378-8733(78)90021-7.

[3] Lawrence Page et al. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[4] Duncan J Watts and Steven H Strogatz. “Collective dynamics of ’small-world’ networks”. In: *Nature* 393.6684 (1998), pp. 440–442. DOI: 10.1038/30918.

[5] Tiago P. Peixoto. “The graph-tool python library”. In: *figshare* (2014). DOI: 10.6084/m9.figshare.1164194. URL: http://figshare.com/articles/graph_tool/1164194 (visited on 09/10/2014).

[6] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.