Chess ratings: Leveraging Network Methods to Predict Chess Results

Lazar Milikic, Lars Quaedvlieg
EE-452 Final Project Report

***Abstract*—In this report, we address the problem of estimating chess player ratings and match outcome prediction by leveraging network approaches based on their past games and results. We propose a rank regression framework that learns a mapping from input features to match outcomes while also implicitly learning player ratings. We explore numerous techniques such as hand-crafted features and GNNs to learn embeddings and train the regression model. Furthermore, we propose different regularization techniques to capture the similarities in player rankings. Ultimately, our models accomplish the winning results on the corresponding Kaggle challenge outperforming the traditional methods. Our findings indicate that utilizing advanced GNN-based approaches to this problem does not provide a benefit over techniques relying on cleverly hand-crafted features capturing the network properties.**

I. Introduction

The Kaggle competition titled "Chess Ratings - ELO versus the Rest of the World" held in 2010 presented the challenge of forecasting chess match results by leveraging player ratings derived from their historical performance. The organizers of the competition hypothesized that when predicting the chess match results it is necessary to estimate the current ability of each player (i.e., rating, skill, form, etc.) and have a mechanism that can evaluate the expected outcome of a chess game between two players [1]. Predicting a player's current ability and how it compares to other active players has a wide range of applications and is a critical concern in the chess community[1]. For example, ratings are used to determine tournament seeds, which can often impact how the event unfolds (it is normally in the interest of tournament organizers to have the best players competing towards the end of the tournament).

Because players develop a network of chess games through their contests, we argue that players' ratings may be closely tied to the structural roles that their respective nodes have within the graph of all games. As a result, the objective of our research is to estimate the current rating of the players utilizing network approaches based on their prior games and results. Finally, we assess the reliability of our estimations based on the predictive power of future unknown outcomes; for instance, if the rankings are correctly calculated, the higher-ranked player should perform better in the near future.

[1]The importance of the problem is well illustrated in the fact that at the time this was the most popular Kaggle competition in terms of participation.

II. Related Work

The most common rating estimation method is the ELO rating system [2]. The method's fundamental idea is to progressively update the players' ratings after each event. The rating is updated based on the difference between the player's expected and actual performance given his rating and rating of his opponents. To predict the result of the game, ELO uses a logistic curve over the difference in players' assessed ratings.

The winning proposal at the aforementioned Kaggle competition was the rating system ELO++ submitted by Yannis Sismanis [3]. Similarly to ELO ratings, ELO++ employs a single rating for each player and predicts the outcome of a game using the sigmoid function over the difference in ratings between the players. Moreover, ELO++ adds a regularization technique that prevents overfitting with the main idea that a player's rating should be similar to the rating of his opponents while accounting recency of games.

In difference to previous proposals, in this work, we approach solving this problem in two phases. First, we obtain the network features of each player, and then we use these features to learn the players' ratings such that the prediction error of future unknown results is minimized. As in the previous proposals, we put more weight on the recent outcomes and apply the logistic curve over the difference in ratings of the players to predict the probability of the white player winning. Inspired by the outstanding performance of ELO++, we consider regularizing the models using communities detected within the network.

III. Exploration

A. Dataset

The organizers provide the pre-divided data into training, validation, and test set. However, we discover that the test set does not contain the labels required for further evaluations and that the validation dataset is simply a subset of the training set; thus, we decide to disregard these two sets entirely. Therefore for our task, we rely solely on the training dataset that comprises 65,053 thousand games and their outcomes played between approximately 7 thousand chess players over the course of 100 months.

The data can be interpreted as a weighted directed multigraph where each player represents a node and edges represent games between two players. Edge goes from the white

to the black player, and each link contains two attributes: a temporal marker (signifying the month in which the game occurs) and a match score (1 if white wins, 0.5 if draw, 0 if black wins[2]). The graph is multigraph because two players can play more than once between each other, and in total, there are 9154 parallel links.

Edge attribute analysis reveals a strong increase in the number of matches over the last recorded months such that roughly 30% of all matches occur between the 91st and 100th month. Looking at the game outcomes, we learn that around 44% matches are finished with draws, while in 32% of cases, the white player wins, and we only witness a victory of the black player in approximately 23% of games. Thus, we conclude that there is a certain bias (advantage) towards white players winning, which is expected as they make the first move and can dictate how the game develops.

B. Global Properties of the graph

We construct the graph *gamesG* using the edges from the provided dataset as described above. The graph initially contains 77 connected components. However, we discovered the existence of the giant component containing over 97% of nodes and 99.8% edges. Hence, we can discard the miniature components as they do not bring valuable predictive information and probably represent amateur players competing against each other.

Furthermore, we conclude that the directionality of the *gamesG* graph for many of the standard graph analyses may not be meaningful, as the direction of the edge only helps us keep track of who was white and who was a black player. For instance, since we wish to track all the players that a player competed with, we should perhaps disregard the edge directions. Hereafter, for the following graph results, unless stated otherwise, the analyses are performed on the undirected version of our graph.

Hence, we construct a multilink graph with 7115 nodes and 64,926 edges. The graph density is 0.0026, which means that the graph is relatively sparse, matching the property of real-world networks [4]. The average node degree representing the average number of games expected for players is 18.25, which appears to be slightly above the range of typical average degrees found in real-world social networks (we can consider the chess games network as a social network to some extent) [5].

The average shortest path is 4.010. According to the small-world property, the average shortest path of this network should be approximately 3.054. However, this value is not entirely reliable because we have a graph with numerous parallel links (that do not contribute to additional graph connectivity). Consequently, we attempt to correct the result by disregarding the multi-links. In this scenario, we obtain 3.222, which is considerably closer to the actual value, however, we cannot infer that the graph fully satisfies the small-world property. This can be attributed to the presence of leagues where the similarly-rated players compete frequently among themselves, fostering strong connections. Yet, outside of these leagues, there are limited interactions, as weaker players rarely participate in tournaments with stronger players, and vice versa.

According to the average clustering coefficient, the likelihood of two neighbors of a node playing against each other is roughly 0.179. On the other hand, for the global clustering coefficient, we obtain 0.1258. Comparing this result to the average clustering coefficient reveals a notable decline, which can be interpreted as a result of the bracket format of some chess competitions. For instance, if player A defeats both players B and C, players B and C do not face each other.

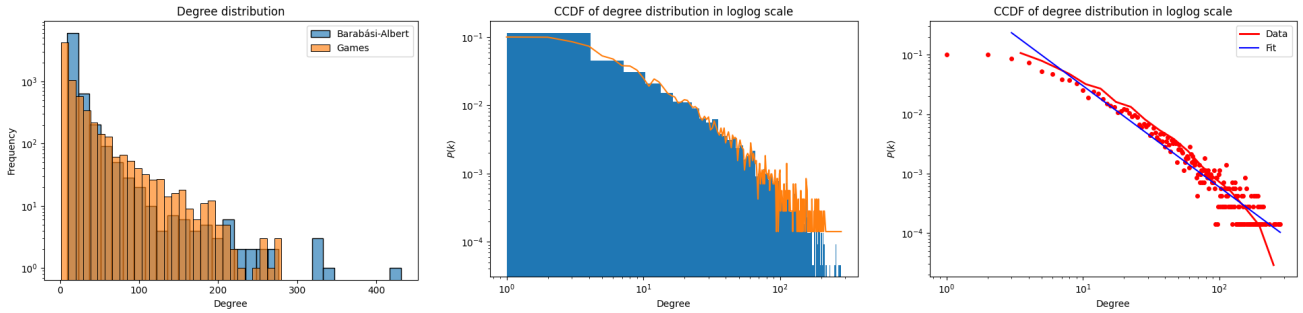
Figure 1a of the node degree distribution of *gamesG* (in practice representing the distribution of the number of games played by players) reveals a heavy-tailed distribution. The existence of hubs is apparent as there are players who participate in far more matches than others. Further investigation confirms the node degree distribution follows power law since the second moment is 1181.42 which is significantly larger than the average node degree, and we observe that its corresponding complementary cumulative distribution function (CCDF) is forming a line on a log-log scale. See Figure 1b. Hence, we conclude that *gamesG* is a scale-free network. Additionally, we are able to find a fit for our power-law distribution. See Figure 1c. The predicted exponent is $\gamma \approx 1.70$, which means that the average distance is expected to be *constant* with respect to the number of nodes, i.e., average distances are expected to be even less than what the ultra-small property predicts [4]. However, as we have shown, in practice, this is not the case because *gamesG* is not even entirely under the small-world regime. The reason for this mismatch can be found in a large number of parallel links between the same pairs of nodes that do not effectively contribute to higher graph connectivity.

C. Comparison with Network Models

Table I displays how the *gamesG* graph compares with the best corresponding fits of some network models. All three models provide a good fit for the number of edges and average degree. Regarding the average shortest path and average clustering coefficient the best match gives the WS network model, while for the global clustering coefficient, we cannot find a model that can approximate it properly. In Figure 1a, we observe that the BA model matches the degree distribution of our network rather closely (random networks do not match our degree distribution since we show that it does not follow a poison distribution but a power law).

As a result of the foregoing, we can say that our network is not random. In other words, we can deduce that players

[2]In practice, the existing ranking systems interpret the match score as the probability that the white player wins a chess match.



(a) Node degree distributions on log-scaled y-axis **(b)** CCDF of degree distribution on log-log scale **(c)** Power-law fit for degree distribution

Figure 1: Node degree distribution (left) of *gamesG* follows the power law since its corresponding CCDF is a line on the log-log scale (middle) and we are able to find parameters of the power law distribution that form a good fit (right).

are not typically matched up at random, but rather that there is a more organized mechanism in place that selects who should play versus whom. That system might be set up so that higher-rated players generally play versus higher-rated players (and vice versa). Furthermore, due to the elimination aspect of many chess competitions, players who win consistently may be expected to play more matches than players who lose more. See here for more details about the format of the chess tournaments.

Furthermore, based on all our observations, we may conclude that Barabási-Albert is also not a very suitable fit for our network. One of the key assumptions for BA models is preferential attachment, which states that the likelihood of a new node connecting to node i is directly proportional to i's node degree. Effectively, this means that the players who have played the most games will play even more matches in the future compared to the rest of the competitors. However, as we can see, this is not always the case in practice. Although players with higher rankings are anticipated to play more matches than those with lower ones, there is undoubtedly more to it. For instance, rankings vary over time, and older players may lose their positions to newer ones, causing them to play fewer official matches or possibly retire completely.

Model	N	$\|E\|$	$\bar{k}$	$\bar{d}$	$\bar{C}$	CG
ER ($p = 0.003$)	7115	64914	18.25	3.37	0.003	0.003
WS ($\beta = 0.37$)	7115	64035	18	3.62	0.176	0.171
BA ($q = 9$)	7115	63954	17.98	3.06	0.013	0.011
gamesG (ours)	7115	64926	18.25	4.01	0.179	0.126

Table I: Comparison of *gamesG* with Erdos-Rényi (ER), Watts-Strogatz (WS), and Barabási-Albert (BA) models. N - number of nodes; $|E|$ - number of edges; $\bar{k}$ - average node degree; $\bar{d}$ - average shortest path; $\bar{C}$ - average clustering coefficient; CG - global clustering coefficient

D. Node properties

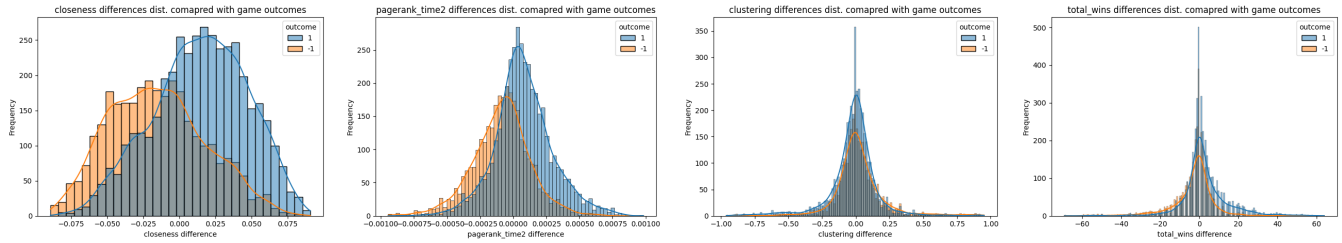
We examine the distinct features of the individual nodes in order to determine which of them might be reliable indicators of players' ratings and match winners.

In order to observe the predictive power of each selected feature, we look at the difference in feature values between the white and black players and the distributions of these differences for edge pairs of different game outcomes. For instance, looking at the games over the last recorded 4 months, we plot the distribution of the difference in the selected features between white and black players when white wins and when black wins. If this feature is expected to be higher for the winning player, we should observe the distribution shift when white wins towards the right, and towards the left when black wins. See Figure 2. In addition, we perform the t-test in order to ensure that means of the two aforementioned distributions are significantly distinct.

Therefore, we find the following features computed on undirected version of the graph to be favorable:

- **Closeness centrality:** It can illustrate the structural role of a node in the graph which could reveal a lot about the player's rank. For example, hubs, whom everyone wants to play against, are most likely the highest-rated players. The shift in Figure 2a shows that having a higher closeness centrality than your opponent means a higher likelihood of winning.
- **PageRank:** A well-known algorithm for assessing the importance of nodes in a network [6]. It could be an important indicator of a player's quality because if a player's neighbors are important and play many games, that player could also be rated higher. See Figure 2b. In addition, we observe a stronger shift if more importance is given to the more recent edges using Equation (1).
- **Node degree:** We regard that playing more recent games than your opponent is correlated with having a higher chance to win matches.
- **Eigenvector centrality:** High when a node has well-connected neighbors; we observe that a player who plays with plenty of well-connected nodes is going to have a higher likelihood to win matches.
- **Betweenness centrality:** The nodes with the highest betweenness centrality represent players that are in between leagues (e.g., recently earned promotion, hence they are in good form).

In Figure 2c, we display an example of a feature that does not seem to correlate with match-winning likelihoods and player ratings. Finally, we analyze how well we can



(a) The distributions of differences for closeness centrality

(b) The distributions of differences for PageRank

(c) The distributions of differences for the clustering coefficient

(d) The distributions of differences in the total number of wins

Figure 2: The distributions of differences between white and black players when white wins (label 1) and when black wins (label -1) computed for the matches happening over the last recorded 4 months.

predict the winner of the match if one player has more wins. Since we look at the match outcomes of the last 4 recorded months, we discard edges representing the games from this period when computing this property to avoid data leakage. Surprisingly, looking at the wins in the past is not a very reliable predictor of future match outcomes as the shift seems to be much weaker compared to some other examined properties. See Figure 2d. Only, according to Equation (1), putting more weight on the more recent match outcomes gives us a more significant correlation between this property and match outcomes. Thus, when training models to predict match results, we should prioritize recent matches as they appear to be more revealing of future game outcomes.

E. Community structure of the graph

We are convinced that our graph contains an underlying community structure such that communities form leagues where similarly rated players play together. Hence, we believe that if we can correctly predict these leagues, we could use them for the regularization of our models [3]. Therefore, it is desirable to have multiple smaller communities in order to facilitate regularization over a relatively small number of nodes. This reasoning is based on the understanding that it is unrealistic to demand that thousands of nodes possess similar rankings. To predict such communities we use again the undirected version of the *gamesG* graph (all games should be treated equally) and two following methods.

1) Louvain Communities: The premise is that a group of people who often play against each other are of similar rankings and establish a league. Since the players' rating is a fluid notion and players can advance between ratings and leagues, we employ Louvain Communities community detection algorithm [7] using temporally weighted edges to form communities, hoping that we can obtain communities where players have many recent matches between each other. Also, we experiment with building Louvain communities using different resolution parameters (higher resolution increases preference for smaller communities). Selecting the resolution of 2.5 gives 51 communities and graph modularity 0.56, which is a moderately strong value and shows that gamesG is indeed community-structured.

Interestingly, global clustering coefficients within communities are now close to or even larger than average clustering coefficients, which is not the case when looking at global graph features. This could be due to the Swiss chess tournament structure, in which everyone plays against everyone, which is the most popular tournament format when competitors are of comparable quality.

2) Spectral clustering: We also perform spectral clustering with the combinatorial or normalized Laplacian features and the K-means algorithm. However, the experimentation indicates that using combinatorial Laplacian does not give meaningful clusters (always predicts one giant cluster containing the majority of nodes). Hence, we use normalized Laplacian and select the number of clusters such that we obtain smaller communities while maximizing graph modularity. Subsequently, we choose 60 as the number of clusters and obtain the modularity of 0.362. As a result, for most of the predicted clusters, we seem to get high clustering coefficients, and most of the games played within those communities seem to be played in the later months, which are desirable properties. Moreover, spectral clustering has grouped some nodes that did not have any matches between each other, which is intriguing as those players may be lower-ranked ones that in general do not have many games and can be put in the same low-tier cluster.

IV. Exploitation

In this section, we formalize our approach to conducting chess rating estimation through the chess match forecasting challenge. We describe the different approaches we have explored in an attempt to solve the problem. Initially, in Section IV-A, we discuss the framework employed in this project. Then, in Section IV-B, we utilize multiple hand-crafted features that were discussed in Section III. In Section IV-C, we expand the model by using the learned unsupervised node embeddings [8]. Finally, in Section IV-D, we discuss an end-to-end GNN algorithm to learn features, make predictions, and estimate player ratings.

A. Rank Regression

We present a rank regression framework that we have developed to attempt to predict player rankings and match outcomes based on hand-crafted or learned features and regularization techniques. Formally, the goal of the framework is to learn a mapping from the input features to the probability that the white player wins, with the added value

of learning player ratings implicitly. We approach this as a supervised learning problem, where we are given a graph of historical game outcomes.

In this graph, the limited number of games played by each player, averaging around 18 (average node degree) games, poses a risk of overfitting. Moreover, players' form and abilities, as well as their ratings, fluctuate and evolve over time. To address this, recent outcomes are given a higher weight due to their increased relevance as it is done when computing ELO++ [3].

$$t^{ij}_{\text{norm}}(\gamma) = \left(\frac{1 + t_{ij} - t_{\min}}{1 + t_{\max} - t_{\min}} \right)^{\gamma} . \tag{1}$$

In Equation (1), a normalized form of time-weighting is implemented, where t is in months ($t_{\max}$ and $t_{\min}$- max and min months considered). Furthermore, the hyperparameter $\gamma \in [1, \infty)$ controls the strength of recent matches. When γ is 1, the scaling is linear, while for higher values the relevance of older results degrades exponentially with time.

Rank regression model: We develop the following predictive model for the match outcomes:

$$\hat{\sigma}(\mu_i, \mu_j) = \text{sigmoid}\left(W\left(\mu_i - \mu_j\right) + b\right) .$$

Here, $\text{sigmoid}(\cdot)$ aims to predict the probability of white winning between player i (white) and j (black), given their respective feature vectors $\mu_i, \mu_j \in \mathbb{R}^{m \times 1}$. These are either hand-crafted or learned feature vectors of dimensionality m. Furthermore, $W \in \mathbb{R}^{1 \times m}$ and $b \in \mathbb{R}$ are learnable weights and a bias. Consequently, $W\mu_i$ and $W\mu_j$ are scalars that can be interpreted as learned ratings of the players, similar to ELO ratings, since the larger the predicted value $W\mu$ is, the larger probability for this player to win a chess match. In addition, the bias b can be seen as an intrinsic difference between wins starting with black or white pieces. We let $r_i = W\mu_i$ be the ranking of player i.

Neighborhood and community regularization: The ranking of a player is expected to exhibit similarities with the rankings of other players within their neighborhood or community. This insight serves as a basis for regularization in models as in ELO++ [3]. Three potential approaches are explored in this regard:

1) Neighborhoods: For each player i with the neighborhood $\mathcal{N}_i$, the neighborhood ranking is determined as a weighted average, considering the rankings of the opponents j in $\mathcal{N}_i$:

$$n_i = \frac{\sum_{\mathcal{N}_i} t^{ij}_{\text{norm}}(2) r_j}{\sum_{\mathcal{N}_i} t^{ij}_{\text{norm}}(2)} .$$

2) Densely connected communities: Since players may occasionally face opponents of different ranks, densely connected communities of players are identified. By weighting with the normalized recency, these communities are expected to consist of players who have recently played numerous matches together. The Louvain community detection algorithm is employed to find communities that maximize modularity. According to the conclusions from exploration, we select the resolution of 2.5 and use edges weighted with $t^{ij}_{\text{norm}}(2)$ to detect communities. For each community $\mathcal{C}$, the average rating is computed as a simple average rating of the nodes within the community:

$$n_i = \frac{1}{|C|} \sum_{i \in \mathcal{C}} r_i .$$

3) Spectral clustering: A similar approach to community detection is applied using spectral clustering. This technique aims to identify communities based on the underlying spectral properties of the graph. Following, the findings during the exploration phase, we use symmetric normalized Laplacian to compute node features and 60 as the number of clusters. Again, the average rating of a cluster (community) is evaluated as a simple average of its node rating.

Final loss function design: To train the models effectively, a suitable loss function is essential. The following loss function is cleverly designed, incorporating the predictions and regularization terms:

$$\mathcal{L} = \sum_{i,j} t^{ij}_{\text{norm}}(\gamma) \cdot \text{BCE}(\hat{\sigma}_{ij}, \sigma_{ij}) + \lambda \cdot \sum_{i} \left(r_i - n_i\right)^2 . \tag{2}$$

This approach is adapted from [3] to use binary cross-entropy instead of mean-squared-error, and to utilize different community detection approaches for normalization.

Here, λ represents the regularization parameter. The first term captures the binary cross-entropy error between the predicted outcome probability $\hat{\sigma}_{ij}$ and true outcome σ_{ij} weighted by recency, for each game of players i and j. The second term regulates the player rankings by penalizing the difference between the ranking of a player r_i and their average neighborhood or community ranking n_i. By optimizing this loss function, the models aim to strike a balance between accurate predictions of match outcomes and capturing the similarity of player rankings within their respective neighborhoods or communities.

To optimize the rank regression model, we employ gradient-based optimization methods.

Model evaluation: We employ the RMSE (root mean square error) variation used in the Kaggle competition called Player/Month-aggregated RMSE (PM-RMSE) to evaluate the precision of our models and their predictive ability to estimate ratings and, subsequently, future results. We utilize this error to rank and compare our findings to the outcomes of the actual competition. The loss is evaluated as follows:

$$PM - \mathcal{R}MSE = \sqrt{\frac{1}{|S|} \sum_{i} (\hat{\rho}^t_i - \rho^t_i)^2} . \tag{3}$$

where ρ_i^t is the true aggregated score (probability of winning) for a month t and player i, and ρ_i^t is our prediction of the respective value.

Additionally, we assess the models' accuracy in predicting chess match results. Although we primarily train our models to predict the probability of the white player winning, we can under certain assumptions perform the game outcome prediction. Hence, we presume that if the predicted probability for the white player to win is below 33.33%, the black player wins. On the other hand, if the model predicts probability above 66.66%, we predict that the white player wins, otherwise, we infer a draw. This way all possible outcomes receive an equally-sized interval to be selected.

B. Hand-Crafted Features

This section discusses a baseline model for constructing a feature vector μ_i for a player i. In order to anticipate player ranking and match results with a small amount of training data, it is crucial to focus on a limited set of variables that provide relevant information. This section mentions the features that are primarily considered favorable for predicting match outcomes, as discussed in detail in Section III-D. We view this algorithm as a baseline for more complex algorithms introduced further down the paper.

Selected features:

- Number of games weighted by $t^{ij}_{\text{norm}}(2)$: This feature assigns greater importance to recent games by weighting the number of games played by each player with the time metric defined previously.
- PageRank weighted by $t^{ij}_{\text{norm}}(2)$: Similar to the previous feature, it is calculated on weighted edges with the square of time passed since the game.
- Closeness: Closeness is a measure of centrality in a network capturing the proximity of a player to others.
- Eigenvector centrality weighted by $t^{ij}_{\text{norm}}(2)$: high if a node is playing with nodes with lots of recent matches.
- Betweenness: Betweenness centrality assesses the extent to which a player lies on the shortest paths between other players.

Because we want to account for all games played by each participant, all features are computed on the undirected form of the *gamesG* network built from the links of the training set entries.

C. Learned unsupervised node embeddings

In an attempt to improve on the existing hand-crafted features, we resort to using learning node representations that are particularly envisioned to have strong discriminative power for the given tasks. We employ the well-known Node2Vec [8] and Laplacian eigenmaps [9] embeddings to extract node features μ_i for each player i.

Node2Vec: Node2Vec is a popular graph embedding technique that captures the structural information of a network by mapping nodes to low-dimensional vectors. This approach enables the models to learn the expressive characteristics of players based on their interactions and connections within the game network. The Node2Vec algorithm incorporates a random walk strategy to explore the network and capture the neighborhood of each node. By balancing the exploration of neighboring nodes and the exploitation of already visited nodes, Node2Vec can effectively capture the network's structural characteristics.

We have seen strong indications in previous works and our explorations that players within the same communities and neighborhoods should have similar ratings. Thus, we set $p = 1$ and $q = 0.1$. This way we capture homophily, i.e., nodes from the same communities have comparable embeddings and thus have comparable ratings. Moreover, to emphasize the higher relevance of the connections representing more recent games, when computing Node2Vec we use $t^{ij}_{\text{norm}}(2)$ as edge weights.

Laplacian eigenmaps: We also attempt to apply Laplacian eigenmaps where node embeddings are determined by the first D non-trivial eigenvectors of graph Laplacian. Since exploration showed us that clustering using combinatorial Laplacian does not give any meaningful results, we use symmetric normalized Laplacian on unweighted *gamesG* to obtain the node features.

Again, as was the case with hand-crafted features, we use the undirected version of our graph that is built with the edges of the training set.

D. End-To-End Learning with Graph Neural Networks

Finally, we present a graph neural network (GNN) approach to learning the match scores and player rankings end-to-end. Some potential benefits of GNNs over Node2Vec include the ability to capture more complex graph structures, end-to-end learning, and the incorporation of initial node and edge feature information.

Given a graph $G(V, E, X, Y) \in \mathcal{G}$, where G is a graph in the set of all graphs $\mathcal{G}$ with vertex set V and edge set E, and corresponding vertex and edge feature matrices X and Y, let us denote a convolution layer of a GNN as a function $X' = f_\theta(V, E, X, Y)$, parameterized with learnable weights θ. The convolution updates the current node embeddings X into X'. This update rule is the backbone of many popular convolution layers, such as Graph Convolution Networks [10] (GCN) and Graph Attention Networks [11] (GAT). We experiment with these two types of layers since GATs are considered good for learning node-level features and GNCs are a good baseline method.

Furthermore, since we have a multi-graph with quite a high connectivity, some over-smoothing is expected for a deep GNN. We attempt to address this by using Differential Group Normalization [12] (DGN), which learns to create groups and normalizes nodes within the same group independently to increase their smoothness and separate node

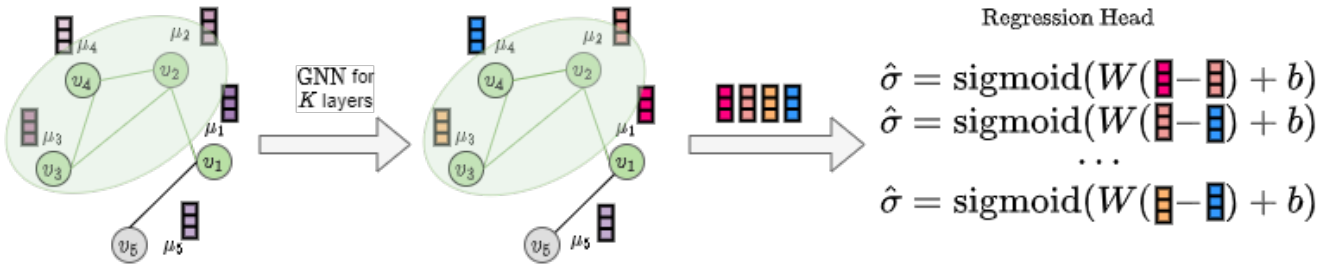


Figure 3: An example of the training procedure of the end-to-end GNN architecture with an example graph and initial node embeddings. The green region, with the green vertices and edges, represents the subgraph that is sampled by the mini-batch sampler during training. For each edge in the sampled subgraph, the node embeddings are updated and extracted with the GNN. The final embeddings are then put through the regression head to compute the prediction and learn the player rankings.

distributions among different groups to significantly alleviate the over-smoothing issue.

As node features, we ultimately decided to use only the "closeness" as an initial embedding for the nodes since it has been shown to be well correlated with the likelihood of winning a match. Moreover, the GNN also utilizes the edge feature $t^{ij}_{\text{norm}}(2)$, which is a measure of the recency of the matches. These features are selected by initially using ones and random embeddings, and systematically checking other hand-crafted features.

We use the GeLU [13] activation function, which is said to be particularly well-suited for GNNs. By stacking convolutional layers, DGN layers, and activations, we construct a GNN for extracting node embeddings. These node embeddings are then used as input features to the rank regression model.

We train the GNN with the rank regression head end-to-end in batches, by sampling a subgraph of nodes and edges from the graph on the depth of the network. It iterates over a set of edges in mini-batches, yielding a subgraph induced by the edge mini-batch. This technique was utilized in [14]. The full setup is depicted in Figure 3.

V. Experiments

In this section, we discuss the experiments that we have performed. First, we describe the setup of the experiments, after which we present the results and their interpretations.

A. Experimental Setup

Validation and Test sets: As we are only given labels for the training set and the validation set is only a subset of the first, we must split our dataset to properly evaluate our models. Following the discussions during the original Kaggle competition [1] that suggest that taking the last couple of months for validating results should give a good estimate of the performance on the competition's test set, we use matches from the 97th and 98th months as the validation set and last two months (99th and 100th) for the test set.

Optimization: We use AdamW [15] optimizer with default PyTorch [16] settings for the training of all our models.

Hyperparameter validation: Choose the hyperparameters γ and λ and select one of 3 community structures (from IV-A) to minimize PM-RMSE on the validation set.

Training with the Hand-crafted features: As described in the respective section, we train the model with the 5 hand-crafted features. Since all considered features have only positive values we use the min-max scaler for normalization.

Training with learned unsupervised node embeddings: We test performance with various emebddings sizes: $\{8, 16, 32, 64\}$.

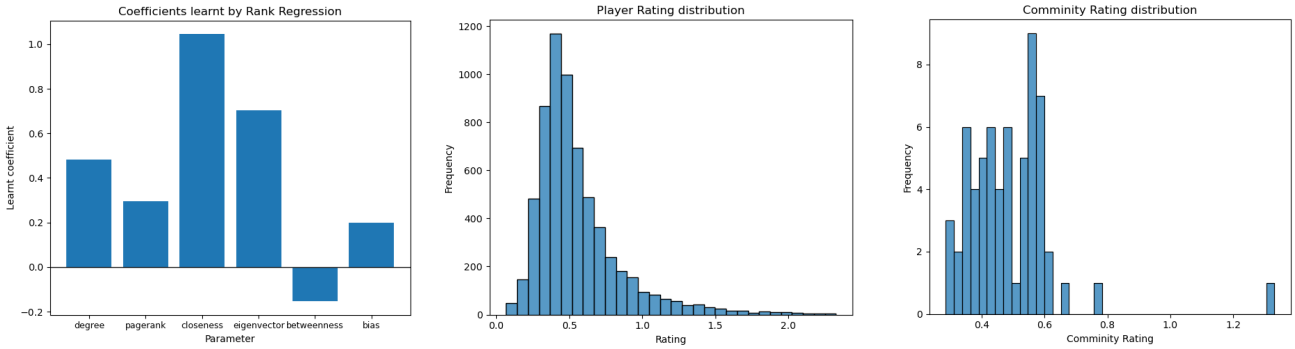
Training the GNN: Initially, we train the method on a model with two hidden layers of size 64 and 32 respectively using GCN layers and GAT layers. For the GAT layers, 8 and 4 attention heads are utilized consecutively. Then, we test with different parameters for the best-performing model and observe whether the addition of Differential Group Normalization has an impact on the performance of deeper models. The number of groups was selected using the best validation set performance.

B. Results

In Table II we show the performances of 4 presented approaches. Based on the scores we obtained for PM-RMSE, all our models would rank as top 15 on the leaderboard provided in [1]. Moreover, the models using the Hand-crafted features and GNN would rank first and second respectively. However, it is critical to bear in mind that these results are based on an entirely different and slightly smaller dataset (albeit the test set is carefully chosen to ensure that the results are as reliable as possible), so our claims should be approached with a dose of skepticism. On the other hand, despite the GNN being the best-performing model, the prediction accuracies for match outcomes are not particularly impressive. However, having that the model performances are significantly better than random guessing, the difficulty of the problem, and the unavoidable noise that comes with predicting future outcomes in any sport, and

Model	PM-RMSE	Accuracy [%]
Hand-crafted	**0.67327**	47.38
Node2Vec	0.69982	46.61
Laplacian eigenmaps	0.70013	46.75
GNN	0.68230	**48.35**

Table II: Comparison of the models' performances on the test set. For more details about evaluation metrics refer to Section IV-A.



(a) Feature importance for Rank Regression **(b)** Distribution of the predicted players' rating **(c)** Distribution of average community ratings

Figure 4: Analyses of the results and ratings predicted by the Rank Regression based on Hand-crafted features

also that our models are not designed to explicitly predict match outcomes (only the probability of white winning), and that we do not tune the hyperparameters of our models to maximize accuracy, we claim that results can be considered satisfying. Surprisingly, we get the worst results for models using learned unsupervised node embeddings. Additional analyses show that these models are strongly biased toward predicting the draw. This could be an inherited consequence of the design of the embedding prediction algorithms. They often produce very similar embeddings between neighbors, hence, the estimated ratings must also be similar; thus, a draw is usually predicted.

Experiments on the validation data show that PM-RMSE is maximized when $\gamma = 2$ for all models, confirming the significance of placing more relevance on the recent games.

Moreover, we find that we should regularize the Hand-crafted features model using the Louvain Communities with $\lambda = 0.1$. For both Node2Vec and Laplacian eigenmaps-based models the performance is maximized when using embedding dimensions of 16. We regularize Node2Vec's model with nodes' neighborhoods and $\lambda = 0.75$, while the model using Laplacian eigenmaps executes the best when regularized by spectral clusters and $\lambda = 0.1$.

For the GNN, we found that the GCN model did not yield meaningful results. The best performance was achieved using a 6-layer GAT model with 64-dimensional node embeddings. This model utilized 8 attention heads in each layer, respectively. We also tried incorporating Differential Group Normalization, which resulted in lower training loss but unstable validation loss. Moreover, neighborhood regularization did not improve the model's performance and incurred significant computational overhead due to changing features at each step. We believe that the GNN embeddings already capture similar neighborhood structures, rendering regularization less important.

C. Player's ratings by Rank Regression

We define a novel ranking system that uses Hand-crafted features-based model (the best-performing model on PM-RMSE loss). Figure 4a shows the resulting learnt feature weights W and bias b. We observe that *closeness* has the highest coefficient and is the by far the most important feature while PageRank surprisingly is not as useful as we expected. Furthermore, whereas exploratory analyses suggest that higher betweenness centrality is associated with a higher likelihood of winning, the model learns the exact opposite (this misinterpretation occurs most likely as the effect of the white player's advantage is stronger than the effect of betweenness "disadvantage," which reverses the observed distribution shifts from Section III). Finally, the model learns $b \approx 0.2$ which confirms that there is an intrinsic advantage to starting as a white player.

Thus, we compute the rating for the player i using the learnt feature weights and selected hand-crafted features: $r_i = W\mu_i$. Figure 4b displays the distribution of the predicted ratings which is notably alike to the ELO rating distribution from [3] having a long tail for ultra-successful players that are by a strong margin better than the rest. The predicted ratings are in the range $[0,\ 2.5]$. Thus, we conclude that according to the learned value for b, the advantage that the white player has is not negligible, especially in the duels between mid and low-rated players.

Finally, Figure 4c presents the distribution of average ratings across predicted Louvain communities for the Hand-crafted features-based model. As anticipated, we get highly diverse ratings for different communities, most notably one with distinctly dominant players on the far left (the top-tier community of the strongest players).

VI. Conclusions

We present a novel supervised learning methodology for accurately forecasting chess game scores and player ratings. Our approach leverages network-based techniques and their inherent characteristics to construct more effective models tailored to this specific task. The natural correspondence between the problem and the graph formulation enhances our approach, enabling it to outperform traditional methods. We argue that a comprehensive exploration of network-based techniques for learning rating systems can pave the way for innovative encodings. Specifically, our findings suggest that cleverly hand-crafted features based on network properties outperform even advanced GNN-based learning algorithms while at the same time helping us advance the understanding and develop more sophisticated frameworks for modeling the entity qualities.

REFERENCES

[1] J. Sonas, "Chess ratings - elo versus the rest of the world," 2010. [Online]. Available: https://kaggle.com/competitions/chess

[2] M. E. Glickman, "A comprehensive guide to chess ratings." [Online]. Available: http://www.glicko.net/research/acjpaper.pdf

[3] Y. Sismanis, "How i won the "chess ratings - elo vs the rest of the world" competition," *ArXiv*, vol. abs/1012.4571, 2010.

[4] A.-L. Barabasi and M. Posfai, *Network science*. Cambridge University Press, 2017.

[5] N. Meghanathan, "Maximal assortative matching and maximal dissortative matching for complex network graphs," *The Computer Journal*, vol. 59, 11 2015.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford infolab, Tech. Rep., 1999.

[7] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner, "Maximizing modularity is hard," 2006.

[8] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[9] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 06 2003. [Online]. Available: https://doi.org/10.1162/089976603321780317

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[11] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48 550, 2017.

[12] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," *Advances in neural information processing systems*, vol. 33, pp. 4917–4928, 2020.

[13] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[14] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[15] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.