UNIVERSITY OF OXFORD

PART C DISSERTATION

Network Analysis in Team sports and Applications to English Premier League

RATTANA PUKDEE

MASTER OF MATHEMATICS

TRINITY 2020

Abstract

Data science is increasingly becoming a vital factor in competitive sports due to its quantitative nature and more available data. Networks analysis is one of data-orient approaches that focuses on the structure of a system by modelling the system with a graph. This paper examines applications of Network analysis to analyse football data, in particular, the English Premier League. We extended current techniques and also developed new methods to rank football players and to derive football players chemistry using only publicly available dataset. The results from our study are comparable to one from the media and comparable to existing works but require much less data.

Acknowledgements

I would like to thank the following people for their support, without whose help this work would never have been possible, Dr.Ebrahim Patel for fruitful supervisions and discussions both on academics and football, Prof.Renaud Lambiotte for delivering the motivating C5.4 Network course and answering my questions which helped inspire the idea of this study, my friends especially Panit for their thought-provoking football talks and practical sessions on the PlayStation and finally, my family for their continuous help and support.

Contents

Chapter 1

Introduction

Data science is increasingly becoming a vital factor in competitive sports. Until now, there have been applications in the field of individual sports such as tennis [1],[2] or team sports such as football [3],[4],[5], American football [6],[7],[8], or baseball [9],[10]. The main tasks include score prediction, teams/players ranking or finding suitable players for a team. The main advantage of these data-orient methods is they usually provide quantitative results, enabling meaningful comparisons between players or sports teams. Moreover, these methods are less prone to human bias and could give us a totally different aspect from the traditional methods, e.g. scouts. The increasing popularity was also depicted by a Hollywood film, Moneyball, which was made from a non-fiction book based on a true story [11]. The past decade has seen a huge improvement in technology, including processing power and data collections. As a result, more data are available, which made data science a more favourable method. Many professional sport clubs are paying more attention to data science and well-known clubs such as the Los Angeles Lakers, and Manchester City even has their data science teams [12],[13]. Within the next few years, data science is perhaps destined to become an indispensable component in sports.

Although various techniques such as Machine learning, Statistics or Network analysis have been studied in the literature, the aim of this paper is to investigate applications of Network analysis techniques to team sports and, in particular, to English Premier League football data. The main tool of Networks analysis is a graph, where we model instances as nodes and pairwise interactions between instances as edges in the graph. Interactions can be bilateral such as friendships in a social network [14], [15] or unilateral as in a citation network [16], [17]. We can also add more information on interactions as weights in the corresponding edges, called weighted graphs. Subsequently, mathematical tools are applied to analyse such graphs. Existing tools include centrality measures [18], community detection [19], [20] or nodes ranking [21], [22]. However, one should keep in mind that a network is only an abstraction of the original system [23] and we mainly keep information about the structure of a system. Despite the limitation of this method, Networks have many successful applications in the field of Biology,e.g. protein network [24], [25] or in the field of Economics for Social network [26],[25] or in Sports, e.g. passing networks [8],[4].

In the networks literature, several studies have been carried out on Football networks. For example, in [27],[5],[28], the authors investigated players passing network to gain insights range from identifying play patterns, player importance, to result in predictions. They converted a passing data to a directed network and apply network measures such as Betweenness, Closeness centrality [29]. In contrast, [30],[4] examined football on a larger scale where they analysed match results data and provided alternative ranking for national football teams. The main tool of their analysis is an infamous PageRank algorithm [21], a random-walk based method which was developed originally for websites ranking. In addition to this manner, [31] extended a dynamic-ranking system [1] to rank national football teams which yield a comparable result to the official FIFA ranking. The method relates to work in [8] on ranking college American football teams which are regarded as a generalisation of the Katz centrality [32].

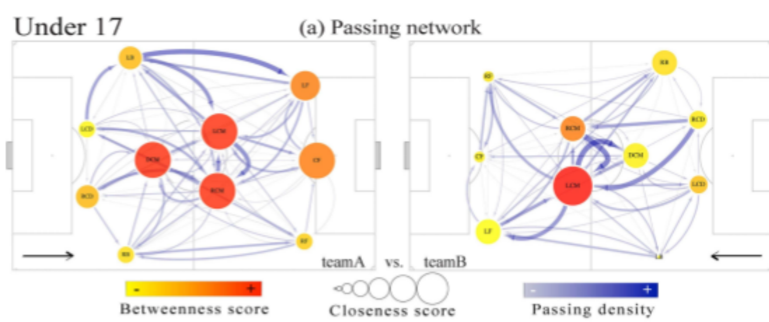

Figure 1.1: A passing network for U17 Portuguese football players [28]

As mentioned earlier, recent developments in data collections lead to a more complex dataset and to illustrate this; we chose an example from [33]. They proposed a human-interpretable yet complete framework to describe actions on a football pitch, as shown in figure 1.2. Such framework opens doors to an uncharted research area that was not explored before such as football players chemistry[34].

	TIME	PLAYER	ACTION	P_{scores}	VALUE
1	92m4s	S. Busquets	pass	0.03	0.00
2	92m6s	L. Messi	pass	0.02	- 0.01
3	92m8s	S. Busquets	pass	0.03	+ 0.01
4	92m11s	L. Messi	take on	0.08	+ 0.05
5	92m12s	L. Messi	pass	0.17	+ 0.09
6	92m14s	A. Vidal	shot	1.00	+ 0.83

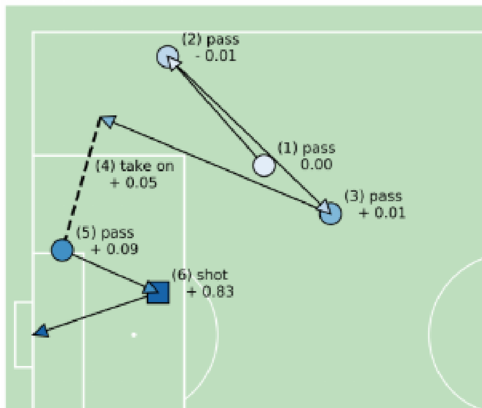

Figure 1.2: Action sequences of Barcelona's final goal in their 3-0 win against Real Madrid on December 23, 2017 [33]

Unfortunately, passing networks and action sequences for English Premier League are not publicly available. Although match results are accessible, we aim to work with a more complex dataset in order to gain insights for players as well. In the light of popularity in football fantasy, which is a game in which participants assemble an imaginary team of real-life footballers and score points based on those players' actual statistical performance or their perceived contribution on the field of play [35], we chose to examine the official Fantasy Premier League dataset in this paper due to its availability and reliability (the data is provided by Opta which is one of the best data providers in the world and is also a provider of action sequences data in figure 1.2). We will see more detail about the dataset in the next section. The aim of our work was to broaden the current knowledge of football players ranking and players' chemistry. In the literature, previous works have only focused on ranking players by their individual performances such as the number of goals or passes which does not take into account information regarding teammates or opponent players in each match. Networks possess the potential to address this issue, and they are works in national football teams ranking using Networks. Most of the applications applied the PageRank algorithm [21]. The paper seeks to extend the works to rank football players in the English Premier League and to examine the use of other methods for node ranking than the PageRank.

On the other hand, there has been little discussion on finding players chemistry in the literature. We propose methods to assess chemistry based on a correlation between players' performance and role-based similarity in directed networks [36]. To assess the results, we will compare our ranking with the ranking from the media. For players chemistry, we will compare with the chemistry measures from [34] that calculated chemistry directly from action sequences data as in [33]. The paper is divided into four sections. The first section gives preliminary definitions and theorems for Network analysis. The second section examines the Fantasy Premier League dataset. In third section analyses methods for ranking football national teams and extensions to players ranking. The next chapter looks at players chemistry from role-based similarity and correlations.

Chapter 2

Preliminary

In this section, we give a brief overview and definitions for Networks. The definitions that we use here are based largely on Oxford C5.4 Network course [23]. The main tool in Networks analysis is graph, as mentioned earlier. A *graph* is defined as $G = (V, E)$ where V is a set of nodes, and E is a set of edges. Each edge $e \in E$ is defined by $e = (v_i, v_j)$ when $v_i, v_j \in V$. We say that a node v_i is adjacent to a node v_j if $e = (v_i, v_j) \in E$. In the case of undirected graphs, the order of v_i and v_j does not matter. While for directed graphs, $e = (v_i, v_j)$ indicate a link from v_i to v_j. We called $e = (v_i, v_j)$, an *incoming edge* to v_j and an *outgoing edge* from v_i. We can assign a weight on each edge in the case of weighted graphs. An undirected graph G with n vertices can be represented by an $n \times n$ *adjacency matrix* A with

$$A_{ij} = \begin{cases} 1 & \text{if node } v_i \text{ is adjacent to node } v_j \\ 0 & \text{otherwise} \end{cases}$$

If a network is weighted, A_{ij} can be assigned as the weight on the edge (v_i, v_j). In undirected graphs, an adjacency matrix is symmetric while it is not necessarily true for directed graphs. Throughout this paper, we mostly focus on weighted directed graphs. We adapted for a definition of an adjacency matrix A for weighted directed graphs as follows

$$A_{ij} = \begin{cases} w_{ij} & (v_i, v_j) \in E \text{ with a weight } w_{ij} \\ 0 & \text{otherwise} \end{cases}$$

For each node, we define a *degree* as the number of edges that contain that node. In undirected graphs, the degree for the ith node is given by

$$k_i = \sum_{j=1}^{n} A_{ij}$$

For directed graphs, we have *in-degree* which is the number of incoming edges to the node and *out-degree* which is the number of outgoing edges from the node. These are given by

$$k_i^{\text{in}} = \sum_{j=1}^{n} A_{ji}$$

and

$$k_i^{\text{out}} = \sum_{j=1}^{n} A_{ij}.$$

The in-degree and out-degree are defined in the same way for weighted directed graphs. It is natural to assume that nodes with high degrees are important. In fact, *degree centrality* is one of the simplest centrality measures that rank nodes importance based on their degrees. However, there are many more ideas when we look at the question "When a node is important ?" For example, for a graph in figure 2.1, the node v has degree 2, which is the lowest degree, but if we remove the node v, the graph will be disconnected with 2 components. If this graph represents a data centre of a tech company where nodes represent servers and edges indicate that one can transfer data from one server to the other. We would regard v as the most important node in the graph as if the server v is broken down; we will not be able to transfer data from one component to the other (left to right). A centrality measure such as the *Betweenness centrality* could highlights this aspect. In this paper, we will work with the *Katz centrality* and *PageRank*

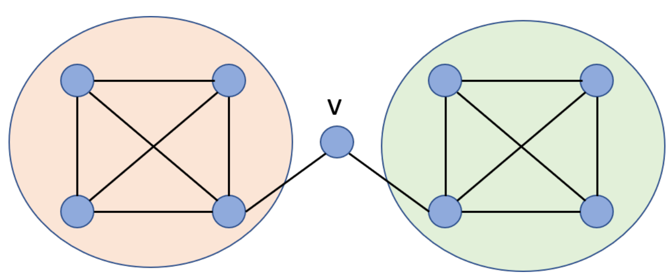

Figure 2.1

centrality. The Katz centrality adapted the idea that a node that can be reached more from other nodes via a path (of any length) is more important. While PageRank is defined as a stationary distribution of a random-walk on networks. A node that the walker visits more often is more important. These will be dealt with more in detail later in this paper.

Chapter 3

Dataset

We use the Official Fantasy Premier League data in this paper. The data is available on the official website, https://fantasy.premierleague.com/, but we will use scrapped data from [37]. As mentioned before, players' score points are based on their performance on the pitch, and figure 3.1 shows an example of a fantasy squad. Each player has fantasy points which are calculated from the various factors as in the table 3.1.

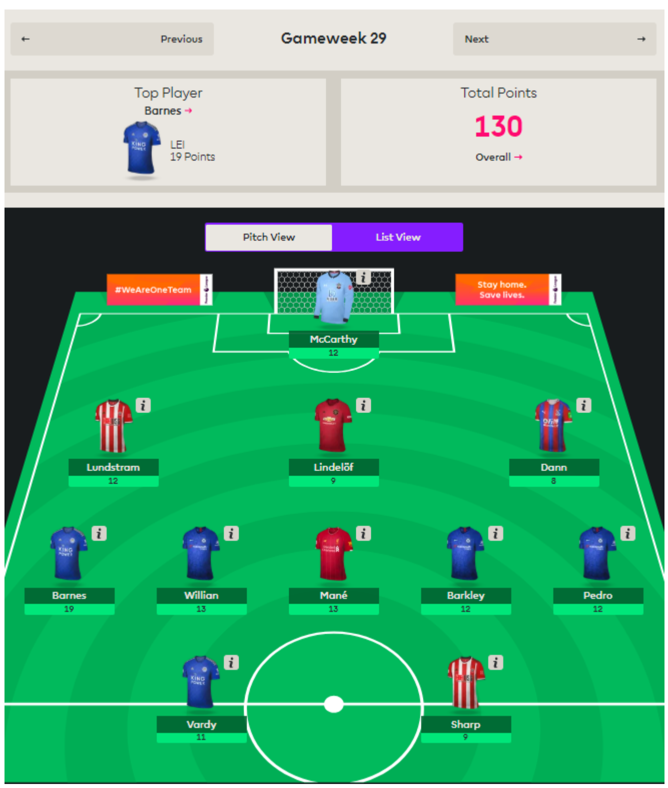

Figure 3.1: Example Premier League Fantasy squad

Action	**Points**
For playing up to 60 minutes	1
For playing 60 minutes or more (excluding stoppage time)	2
For each goal scored by a goalkeeper or defender	6
For each goal scored by a midfielder	5
For each goal scored by a forward	4
For each goal assist	3
For a clean sheet by a goalkeeper or defender	4
For a clean sheet by a midfielder	1
For every 3 shot saves by a goalkeeper	1
For each penalty save	5
For each penalty miss	-2
Bonus points for the best players in a match	1-3
For every 2 goals conceded by a goalkeeper or defender	-1
For each yellow card	-1
For each red card	-3
For each own goal	-2

Table 3.1: Players earn fantasy points based on the following statistics

The fantasy data includes traditional statistics such as goals scored, assists, clean sheets, saves, penalty missed, but it also includes the Bonus Points System (BPS). The BPS attempts to find the best performing players in each match regardless of their positions by utilising a range of statistics in the match. The three best performing players will be awarded bonus points to their total fantasy points. Table 3.2 shows how the BPS score is calculated. We can see that the score includes key statistics such as "making an error which leads to a goal", "creating a big chance", or "missing a big chance" which are not taken into account directly when we calculate a player's fantasy points. It can thus be reasonably assumed that BPS is a better metric for measuring players' performance.

Figure 3.2 shows the distribution of average BPS per game and points per game for players in the 2018/2019 season. We chose only players that played more than 900 minutes in that season. The main reason that we are interested in the Fantasy Premier league data is not only because the data is reliable, we believe that the fantasy points can be a good candidate for a performance metric that is unbiased for players in every position. If we use the number of goals scored as a performance metric, almost every attacker is better than every defender in the league as defenders' main role is not scoring a goal but to prevent it from happening. Such a problem makes it hard to compare players with different position, e.g. defenders and attackers. The Premier league fantasy point combined different aspects of football and therefore is a potential candidate for our performance metric.

The figure 3.3 shows a scatter plot between players' total BPS and fantasy points in the

Action	BPS
Playing 1 to 60 minutes	3
Playing over 60 minutes	6
Goalkeepers and defenders scoring a goal	12
Midifielders scoring a goal	18
Forwards scoring a goal	24
Assists	9
Goalkeepers and defenders keeping a clean sheet	12
Saving a penalty	15
Save	2
Successful open play cross	1
Creating a big chance (a chance where the receiving player should score)	3
For every 2 clearances, blocks and interceptions (total)	1
For every 3 recoveries	1
Key pass	1
Successful tackle (net)	2
Successful dribble	1
Scoring the goal that wins a match	3
70 to 79% pass completion (at least 30 passes attempted)	2
80 to 89% pass completion (at least 30 passes attempted)	4
90%+ pass completion (at least 30 passes attempted)	6
Conceding a penalty	-3
Missing a penalty	-6
Yellow card	-3
Red card	-9
Own goal	-6
Missing a big chance	-3
Making an error which leads to a goal	-3
Making an error which leads to an attempt at goal	-1
Being tackled	-1
Conceding a foul	-1
Being caught offside	-1
Shot off target	-1

Table 3.2: Players earn BPS points based on the following statistics

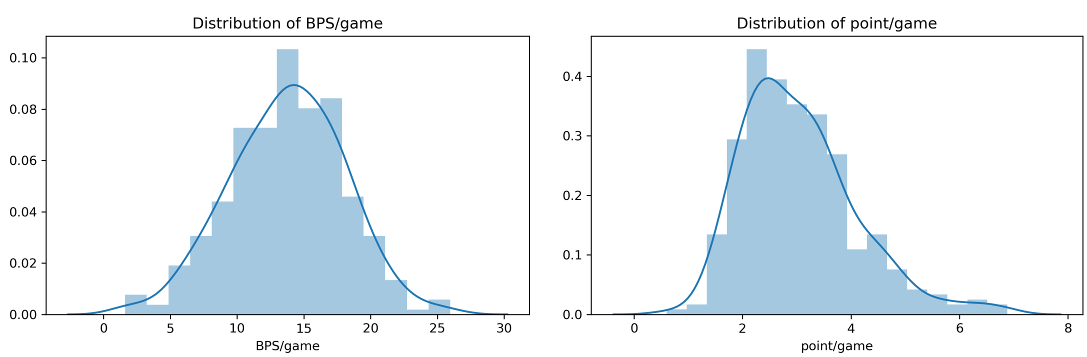

Figure 3.2: Histogram of BPS per game (left) and fantasy point per game (right) for players in the 2018/19 season

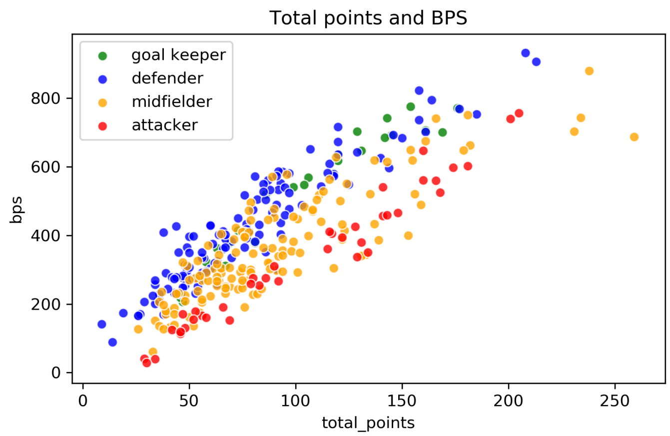

Figure 3.3: Scatter plot of BPS and fantasy points for players in the 2018/19 season

2018/2019 season. Some players apparently played more than others, but we can still see a positive relationship between the fantasy points and BPS. Figure 3.4 illustrates a histogram of points and BPS from every match in the season. It can be seen that the fantasy points give similar distributions across different positions while strikers suffer from lower BPS scores. These findings suggest that we should use total fantasy points instead of BPS when comparing players from different positions. Next, figure 3.2 shows average fantasy points per game and average BPS per game for players in each position. Although players in each position obtained points from similar distributions as in figure 3.4, their average scores are quite different. A general trend is that attackers have the highest variation of BPS per game and fantasy points per game, while goalkeepers have the lowest variation. Also, attackers have the lowest average BPS per game compared to other positions. This may suggest that the Premier League Fantasy is designed to give more BPS to goalkeepers and defender in order to compensate bonus point for these positions. From now, we take the Premier League Fantasy points as our performance metrics, but we will also look at the BPS parallelly as it contains more information about players and has a wider range of score (players with the same points can have different BPS score). Throughout this paper, the positions 1,2,3,4 refer to Goalkeepers, Defenders, Midfielders/Wingers, Strikers, respectively.

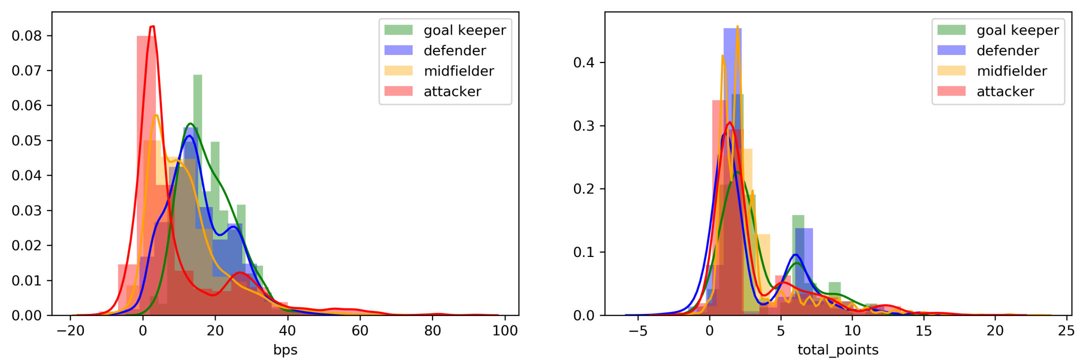

Figure 3.4: Histogram of all BPS (left) and all fantasy points (right) for each position in the 2018/19 season

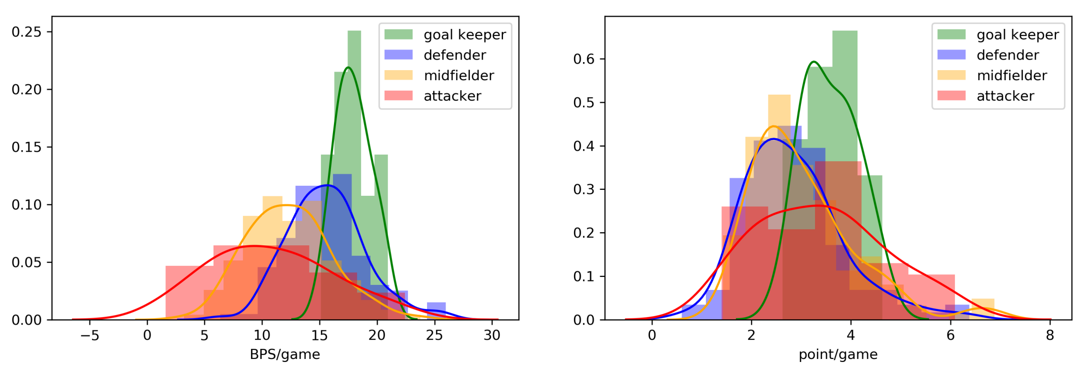

Figure 3.5: Histogram of BPS per game (left) and fantasy point per game (right) for each position in the 2018/19 season

Chapter 4

Player Ranking

In this chapter, we examine network-based methods for football players ranking. Until now, previous work has only focused on ranking players by their individual performances, such as the number of goals or passes. The main limitation is the lack of a performance metric that is unbiased for players in different positions; this makes it hard to compare players between different positions. Moreover, most methods do not take into account information regarding opponent players in each match. For example, when a player played well against star players such as Mohamed Salah or Kevin De Bruyne, that player should gain more credit compared to playing well against moderate players. Networks approaches focus on the structure of a system and possess the potential to address these issues. In fact, there already are works in national football teams ranking [30], [31]. We model players as nodes in a directed graph, and if a player A performs better than a player B (in the opponent team), we draw a direct edge from the weaker player (B) to the stronger player (A). We explore two approaches to player ranking, PageRank [21] and a ranking method based on indirect wins [29].

4.1 PageRank

First, we begin with the PageRank approach. We give a short introduction to random-walk on networks and the PageRank algorithm below. The definitions and theorems are based on C5.4 Network course [23]. Random-walk on networks is a dynamical process that examines the network structure by randomly exploring the network. We initialize a walker at a node from initial distribution. At each step, the walker jumps to an adjacent node with a certain probability that we choose (in general, uniformly). We denote $p_i(t)$ is the probability that the walker visits the ith node after t steps. We can see this random-walk as a Markov Chain, and the probability $p_i(t)$ may converge to a stationary distribution p which, if existed, tell us about how often the walker visits each node in the long run. The stationary distribution could be seen as a centrality measure that if the walker visits a node frequently, that node is important. The benefit of this method is that it could take into account the global structure of the network, e.g., The importance of the ith node depends on a walker that travels across the network while the importance from other methods such as degree centrality only takes into account nodes which adjacent to the ith node. However, one of the major drawbacks is that the probability p may not converge to a unique stationary distribution p if there are multiple absorbing states. The stationary state is unique if and only if the network is strongly connected [23].

The PageRank algorithm manages to get around this multiple stationary distributions problem by randomly teleporting the walker to any node in the network regardless of their connectivity which leads to a strongly connected Markov chain. The algorithm was created originally for websites ranking. The main idea in websites ranking context is "a website is important if it received many hyperlinks from other websites, and we give credit to a website that receives a link from an important website. In random-walk, we know that a walker visits a node A that has many incoming edges frequently. As a consequence, the walker also visits nodes that are adjacent to A more often, but this depends on the number of outgoing edges of A as well. We model websites by a directed network where each website is a node, and each hyperlink is a directed edge between nodes. Let A be an adjacency matrix of this network, and the PageRank algorithm is defined as follows. First, we define the transition matrix T with

$$T_{ij} = \frac{A_{ij}}{k_i^{out}}$$

where $k_i^{out} = \sum_i A_{ij}$ is the sum of the weight of the outgoing edges of the ith node. T_{ij} is the probability of jumping from the ith node to jth node and is defined as the proportion of the weight from i to j, A_{ij}, to the total outgoing weight k_i^{out} from i. For an initial distribution $p(0) = (p_1(0), \ldots, p_n(0))$ of the walker, at each step we update

$$p_i(t+1) = \alpha \sum_{j=1}^{N} p_j(t) T_{ji} + (1-\alpha) u_i \tag{4.1}$$

Let α be the probability that the walker does not teleport at each step. The first term in this equation is the probability of visiting the ith at time $t+1$ without teleporting. $p_j(t)T_{ji}$ is the probability that a walker jumps from the jth node at time t to the ith node at a time $t+1$. The second term is the probability of visiting the ith node by teleporting and u_i is the probability of teleporting to the ith node so we must have $\sum_{i=1}^{n} u_i = 1$. As mentioned earlier, $p(t)$ converges to a unique stationary distribution p, and we can effectively compute p by using the power method where we iteratively apply the equation (4.1) to $p(t)$. The resulted stationary distribution p is called the *PageRank coefficient*.

In the same analogue, we can extend this concept to ranking in competitive sports. We can regard hyperlinks from websites by win-lose relations. For example, if a team A win over a team B, there is a directed edge from B to A. In football context, a good football team won many matches (there are many incoming edges to the node), and if a team beat a strong team, it should gain more credit (since the random-walker visits nodes that represent strong teams frequently, it also jumps to nodes that beats strong teams more often and these nodes, therefore, gain more credit). In fact, [4], [30] performed the PageRank algorithm on win-lose networks to rank national football teams. In this paper, we want to extend the idea of PageRank to rank football players. In [5],[38], the authors have already investigated applications of PageRank to analyze football players but largely limited to players within the same team. [5] looks at passing networks of a football team to identify key players. [38] proposed a framework to rank players' importance in a single match based on the impact they made and the PageRank algorithm.

A possible limitation of previous works is that their methods focus on players ranking in a single match which could lead to an information loss if we want to rank players for the whole season. For example, it is not clear how to compare rankings from two different matches. Also, it is hard to give more credit to players that play well against star players as we need to specify who are star players. For this reason, we chose to construct a network of players' performance for the whole Premier League season and consequently applied the PageRank algorithm. Doing so will automatically handle who are star players for us. Moreover, our work is largely based on limited data (the Premier League Fantasy data, which summarise players' performance for each match, rather than minute by minute actions or passing data). The aim of the research was, therefore, to focus on the big picture of players in the entire season.

The availability of the Fantasy Premier league data give us the performance of players in

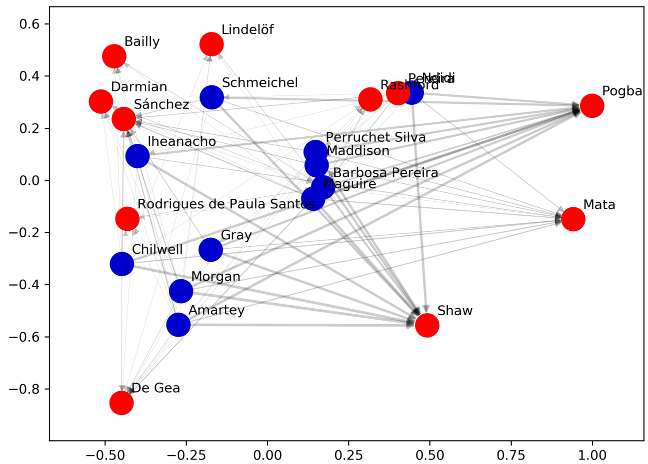

Figure 4.1: A directed graph generated from a match between Man United - Leceister on 11 Aug 2018 which Man United won 2-1, and Paul Pogba was the man of the match

each match via fantasy points. The construction of a network is as follows. We modelled players as nodes and for each match, if a player A has a higher performance point than a player B, we draw a directed edge from B to A with a weight on edge as the average difference between their scores (in case they played together more than once). We chose only players that played more than 60 minutes in each match to make sure substitutions get no disadvantages. Figure 4.1 is a directed graph generated from a match between Man United - Leceister on 11 Aug 2018 which Man United won 2-1. A thicker edge implies more weight on that edge. Note that Paul Pogba was the man of the match according to BBC [39] and we can see that Pogba's node has many thick incoming edges compared to other players in the team. Subsequently, we combined the network for each match into a single network and PageRank algorithm is then applied to this network. The ten players with the highest PageRank coefficient in the 2018/2019 season is shown in the table 4.1 below. The result includes famous players such as Sterling, Salah and Hazard, which were expected from football fans. However, the ranking also includes players from the middle/bottom of the league teams such as Bournemouth, Cardiff City. This might be a good indication that the PageRank algorithm does not prefer only players from top teams. In fact, Lucas Digne from Everton performed quite well in the 2018/2019 and was named the club player of the season. Surprisingly, the only goalkeeper in the list is not Liverpool's Alisson Becker or Manchester City's Ederson but Cardiff city's Neil Etheridge. Manchester City and Liverpool finished first and second respectively in the 2018/19 season,

Name	Team	Position	Points	BPS	match	BPS/ game	Points/ game
Eden Hazard	Chelsea	3.0	223	802	31	25.87	7.19
Raheem Sterling	Man City	3.0	226	722	31	23.29	7.29
Mohamed Salah	Liverpool	3.0	255	673	37	18.19	6.89
Sadio Mané	Liverpool	3.0	230	698	35	19.94	6.57
Paul Pogba	Man Utd	3.0	178	642	34	18.88	5.24
Neil Etheridge	Cardiff	1.0	154	775	38	20.39	4.05
Andrew Robertson	Liverpool	2.0	213	906	36	25.17	5.92
Lucas Digne	Everton	2.0	159	739	33	22.39	4.82
Ryan Fraser	Bournemouth	3.0	176	729	36	20.25	4.89
David Luiz	Chelsea	2.0	164	794	36	22.06	4.56

Table 4.1: Ten players with the highest PageRank coefficient in the 2018/2019 season

and their goalkeepers were selected to be the best goalkeeper of the league from many critics. In addition, Alisson signed for Liverpool in July for an initial fee of £56 million with an option to rise to £66.8 million; a world-record fee for a goalkeeper at the time [40]. The reason for this rather contradictory result is still not entirely clear, but one explanation is that the PageRank algorithm over-reacted to an event when a player performs really well in some matches. The table 4.2 shows five goalkeepers with the highest points in the 2018/19 season. Alisson was ranked first follows by Ederson, Pickford and Etheridge. The points differences between Alisson and Etheridge is 22 points while Etheridge has the highest BPS in the entire league with the value of 775 compared to 771 for Alisson.

Name	Team	Points	bonus	BPS	match
Alisson_Ramses Becker	Liverpool	176	9	771	38
Ederson_Santana de Moraes	Man City	169	6	700	38
Jordan_Pickford	Everton	161	13	706	38
Neil_Etheridge	Cardiff	154	18	775	38
Hugo_Lloris	Tottenham	145	10	691	33

Table 4.2: Five goalkeepers with the highest points in the 2018/19 season

We explore further by looking at histograms of points for each game (left) and BPS for each game (right) for Alisson, Ederson and Etheridge in the 2018/19 season in figure 4.2.

It is clear that there are matches that Etheridge performed extremely well. But in term of consistency, Alisson and Ederson have fewer matches with low points which we perhaps are a more favourable trait for a goalkeeper. We hypothesize that very high points from some matches have a significant effect on ranking result and therefore, great care must be taken assigning weights to directed edges. However, it is undeniable that Etheridge's performance was one of the best in the 2018/2019 English Premier League.

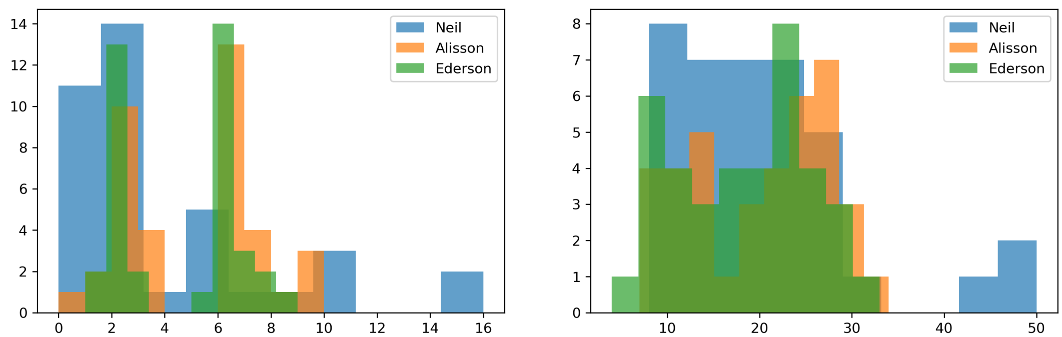

Figure 4.2: Histograms of points for each game (left) and BPS for each game (right) for Alisson, Ederson and Etheridge in the 2018/19 season

We refined the method used earlier by capping differences between points to be no more than six points and no less than two. If a difference is higher than six, we set it equals to six, and if a difference is lower than two, then we set it equals to zero. The number six was chosen because it is the 75th percentile of the score differences between players as shown in the figure 4.3. We also decided that it is difficult to say that a player A played well against a player B if their score difference is only one. The result from applying the PageRank algorithm to this network is shown in the table 4.3. Contrary to expectations, we did not find a significant difference between this ranking and the one from the PageRank with no scoring cap as in the table 4.1. Now, Jordan Pickford is the best goalkeeper in the league, follows by Etheridge.

Now, we compare five players with the highest PageRank coefficient for each position with five best players ranked by FourFourTwo, which is a famous football magazine. We can see that the result from PageRank does not align fully with one from the critic. Apart from Midfielder/ Winger, at most 2 out of 5 players from the PageRank and from the FourFourTwo magazine aligned. Moreover, contrary to expectations, Aleksandar Mitrovic, who scored 11 goals from 37 games, was ranked by PageRank to be the best striker of the season, compared to Sergio Aguero who scored 21 goals from 33 games, the best striker from the magazine. This suggests

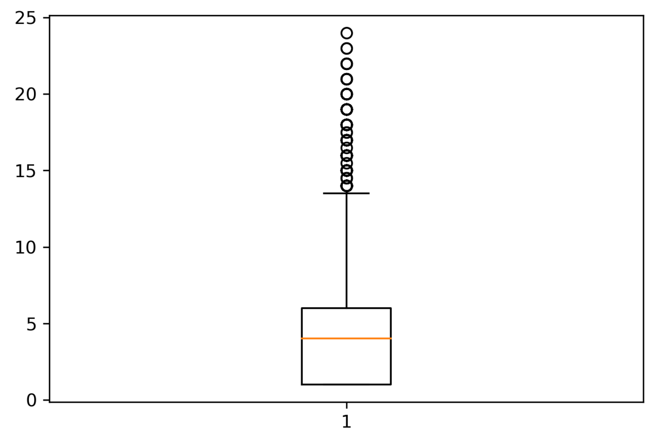

Figure 4.3: A boxplot of score differences in the 2018/19 season

Name	**Team**	**Position**	**Points**	**BPS**	**Match**	**BPS/ game**	**point/ game**
Mohamed Salah	Liverpool	3.00	255	673	37	18.19	6.89
Eden Hazard	Chelsea	3.00	223	802	31	25.87	7.19
Andrew Robertson	Liverpool	2.00	213	906	36	25.17	5.92
Raheem Sterling	Man City	3.00	226	722	31	23.29	7.29
Neil Etheridge	Cardiff	1.00	154	775	38	20.39	4.05
Nathan Aké	Bournemouth	2.00	120	716	38	18.84	3.16
Jordan Pickford	Everton	1.00	161	706	38	18.58	4.24
Ederson Santana de Moraes	Man City	1.00	169	700	38	18.42	4.45
Lucas Digne	Everton	2.00	159	739	33	22.39	4.82
Sadio Mané	Liverpool	3.00	230	698	35	19.94	6.57

Table 4.3: Ten players with the highest PageRank coefficient with capped score differences in the 2018/2019 season

Goalkeeper	**Defender**	**Midfielder/ Winger**	**Striker**
Jordan Pickford	Andrew Robertson	Mohamed Salah	Aleksandar Mitrovic
Neil Etheridge	Lucas Digne	Eden Hazard	Raúl Jiménez
Ederson Santana de Moraes	Nathan Aké	Raheem Sterling	Pierre-Emerick Aubameyang
Lukasz Fabianski	César Azpilicueta	Sadio Mané	Jamie Vardy
Alisson Ramses Becker	David Luiz	Paul Pogba	Callum Wilson

Table 4.4: Five players with the highest PageRank coefficient with capped score differences for each position in the 2018/19 season

Goalkeeper	**Defender**	**Midfielder/ Winger**	**Striker**
Martin Dubravka	Virgil van Dijk	Bernardo Silva	Sergio Aguero
Alisson Becker	Aymeric Laporte	Raheem Sterling	Harry Kane
Ben Foster	Andrew Robertson	Eden Hazard	Pierre-Emerick Aubameyang
Lukasz Fabianski	Ben Chilwell	Sadio Mané	Alexandre Lacazette
David de Gea	Aaron Wan-Bissaka	Mohamed Salah	Raul Jimenez

Table 4.5: Five players with for each position in the 2018/19 season from FourFourTwo [41]

that PageRank may not be a good candidate for player ranking. However, we believe that the PageRank algorithm could point out interesting players to investigate, but careful attention must be paid.

4.2 Indirect wins

An alternative solution is an indirect wins approach. In [8] the authors extended the idea that we often hear from sports fans that "Although my team A didn't play your team C this season, it did beat B who in turn beat C. Therefore A is better than C and would have won had they played a game" into a sports team ranking algorithm, in particular, American football in their paper. There are key advantages to this method. Firstly, it is interpretable as the main idea came from a classic argument in sports. Secondly, it also takes into account the global structure of the network that is a player score depends on interactions between other players in the league. The author in [31] examined the application of this ranking method to rank national football teams, and we seek to rank football players in this paper. We modelled players as nodes, and for each match, if a player A has a higher performance point than a player B, we draw a directed edge from B to A with a weight on edge as the average difference between their scores (same as in PageRank). Let A be the adjacency matrix of the network. The direct wins are naturally defined by

$$\text{direct wins for player i} = \sum_{j} A_{ji}$$

Note that direct win for player i is just an in-degree of the ith node. We say that a player A has an indirect win over a player C if there is a player B that A win over B and B win over C. We called this indirect wins at a distance two and we could also consider high-order indirect wins (A beats B beats C beats D). We defined indirect wins at a distance 2 by

$$\text{indirect wins at a distance 2 for player i} = \sum_{j,k} A_{kj} A_{ji}$$

We discount the effect of indirect wins by a constant factor α for every level of indirection. For example, the effect of indirect wins at a distance 3 is discounted by α^2. We define the total win score w_i of a player i as the sum of direct win and indirect wins at all distances,

$$
\begin{aligned}
w_i &= \sum_j A_{ji} + \alpha \sum_{j,k} A_{kj} A_{ji} + \alpha^2 \sum_{h,j,k} A_{hk} A_{kj} A_{ji} + \dots \\
&= \sum_j (1 + \alpha \sum_k A_{kj} + \alpha^2 \sum_{h,k} A_{hk} A_{kj} + \dots) A_{ji} \\
&= \sum_j (1 + \alpha w_j) A_{ji} \\
&= k_i^{\text{in}} + \alpha \sum_j A_{ij}^T w_j
\end{aligned} \tag{4.2}
$$

When $k_i^{\text{in}} = \sum_j A_{ji}$ is the total weights of edges pointing toward the vertex i. Similarly, the loss score l_i is defined as

$$
\begin{aligned}
l_i &= \sum_j A_{ij} + \alpha \sum_{j,k} A_{ij} A_{jk} + \alpha^2 \sum_{h,j,k} A_{ij} A_{jk} A_{kh} + \dots \\
&= \sum_j A_{ij} (1 + \alpha \sum_k A_{jk} + \alpha^2 \sum_{h,k} A_{jk} A_{kh} + \dots) \\
&= \sum_j A_{ij} (1 + \alpha l_j) \\
&= k_i^{\text{out}} + \alpha \sum_j A_{ij} l_j
\end{aligned} \tag{4.3}
$$

when $k_i^{\text{out}} = \sum_j A_{ij}$ is the out-degree of the ith node. We define the total score of a team to be the difference of the win and loss scores

$$s_i = w_i - l_i$$

,and we rank team based on this score. Let $\mathbf{w} = (w_1, w_2, \dots)$, $\mathbf{l} = (l_1, l_2, \dots)$, $\mathbf{k}^{\text{out}} = (k_1^{\text{out}}, k_2^{\text{out}}, \dots)$ and $\mathbf{k}^{\text{in}} = (k_1^{\text{in}}, k_2^{\text{in}}, \dots)$ we can arrange equations 4.2, 4.3 to

$$\mathbf{w} = \mathbf{k}^{\text{in}} + \alpha A^T w \text{ , } \mathbf{l} = \mathbf{k}^{\text{in}} + \alpha A l$$

We can rearrange to

$$\mathbf{w} = (I - \alpha A^T)^{-1} \mathbf{k}^{\text{in}} \text{ , } \mathbf{l} = (I - \alpha A)^{-1} \mathbf{k}^{\text{out}}$$

which is regarded as a generalisation of the Katz centrality [8]. Not every value of α makes **w, l** converge and it is shown in [31] that **w, l** converge if $\alpha < \lambda_{\max}^{-1}$ when $\lambda_{\max}$ is the largest

Name	Team	Pos	Points	BPS	BPS/ game	Points /game	Score
Raheem Sterling	Man City	3.00	226	722	23.29	7.29	7,129.90
Mohamed Salah	Liverpool	3.00	255	673	18.19	6.89	6,332.49
Sadio Mané	Liverpool	3.00	230	698	19.94	6.57	6,274.56
Sergio Agüero	Man City	4.00	176	635	22.68	6.29	5,481.92
Eden Hazard	Chelsea	3.00	223	802	25.87	7.19	5,450.15
Andrew Robertson	Liverpool	2.00	213	906	25.17	5.92	5,408.66
Trent Alexander-Arnold	Liverpool	2.00	180	734	27.19	6.67	5,266.74
Virgil van Dijk	Liverpool	2.00	207	917	24.78	5.59	5,078.87
Heung-Min Son	Tottenham	3.00	147	446	20.27	6.68	4,798.83
Leroy Sané	Man City	3.00	135	461	25.61	7.50	4,654.20

Table 4.6: Ten players with the highest ranks from indirect wins ranking with $alpha = 0.8\alpha_{\max}$ in the 2018/2019 season

eigenvalue of the adjacency matrix A. In [8] the authors assessed the performance of α by calculating the fraction of games won by higher-ranked team and suggested that the best result of α is around $0.8\lambda_{\max}^{-1}$. We denote $\alpha_{\max} = \lambda_{\max}^{-1}$ as the maximum value that α can take. In an extreme case, $\alpha = 0$ means we only consider the effect of direct wins and loses. Table 4.6 shows ten players with the highest rank from the indirect win method with $\alpha = 0.80\alpha_{\max}$. We note that this ranking includes many well-known players that football fans would expect to see. Moreover, the ranking includes Liverpool's centre back, Van Dijk. Van dijk was named PFA (Professional Footballers' Association) Player of the Year and Premier League Player of the Season for the 2018/19 season but his name was not even listed among top ten players in every ranking from PageRank (see tables 4.1, 4.3). This result suggests that there is a good agreement with the indirect wins ranking method and result from the real world (see table 4.5).

In our view, great care must be taken when choosing the value of α to make sure that the value is plausible. We propose an alternative method for choosing the value of α by looking at the proportion of total score that was explained by direct wins and loses. We believe that a good α value should give a high enough proportion of score from direct wins and loses. Figure

4.4 illustrates the boxplots of proportions of total scores explain by direct wins and loses for each value of α. The proportion is calculated by

$$\frac{\text{direct wins for player i} - \text{direct loses for player i}}{w_i - l_i}$$

for each node i. We can see that when $\alpha = 0.8\alpha_{\max}$, only 20 percents of the total score came from direct wins and direct loses which in our opinion, is slightly low as we should give more credit to direct wins and loses rather than indirect ones which based on our assumption. Interestingly, the median proportions of total scores explained by direct wins and loses have a

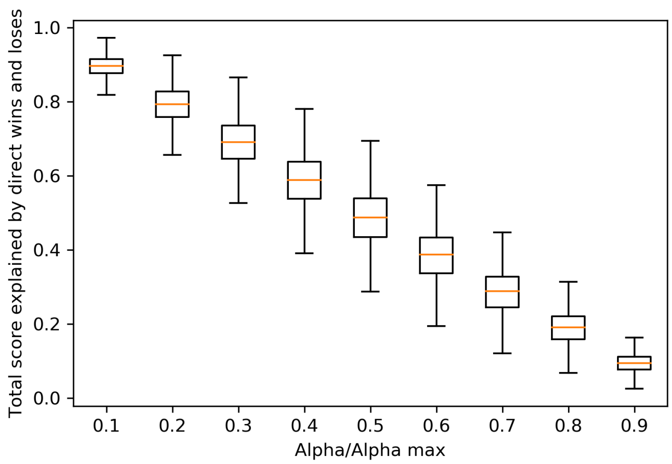

Figure 4.4: Boxplots of proportions of total scores explained by direct wins and loses for each value of α

linear relationship with α. We chose $\alpha = 0.5\alpha_{\max}$ to make sure players' scores were explained by direct wins and loses 50 percents, on average. The result is shown in table 4.7

There were no significant differences between table 4.7 and 4.6 in terms of players, but their scores have different magnitude. The players' scores allow us to make a quantitative comparison between players, regardless of their positions. Now, we look at goalkeepers ranking to compare with one from the PageRank algorithm. Table 4.8 shows five goalkeepers with the highest scores from indirect wins method with $\alpha = 0.5\alpha_{\max}$. We see that Alisson and Ederson achieved an overwhelming victory against other goalkeepers with their score around two times of the goalkeeper in the third rank. This also aligns with 90min's opinion that they are the second and third best goalkeepers in the 2018/19 season [42] (90min.com is a London-based football news platform). However, 90min also suggests that Man United's David de Gea, is the best goalkeeper of that season while he ranked 9th in the indirect win method with the score

Name	Team	Pos	Points	BPS	BPS/ game	Points /game	Score
Raheem Sterling	Man City	3.00	226	722	23.29	7.29	2,862.34
Mohamed Salah	Liverpool	3.00	255	673	18.19	6.89	2,634.95
Sadio Mané	Liverpool	3.00	230	698	19.94	6.57	2,483.93
Andrew Robertson	Liverpool	2.00	213	906	25.17	5.92	2,190.75
Eden Hazard	Chelsea	3.00	223	802	25.87	7.19	2,170.64
Sergio Agüero	Man City	4.00	176	635	22.68	6.29	2,147.69
Trent Alexander-Arnold	Liverpool	2.00	180	734	27.19	6.67	2,104.84
Virgil van Dijk	Liverpool	2.00	207	917	24.78	5.59	1,995.17
Leroy Sané	Man City	3.00	135	461	25.61	7.50	1,858.94

Table 4.7: Ten players with the highest ranks from indirect wins ranking with $\alpha = 0.5\alpha_{\max}$ in the 2018/2019 season

of 66.05. This may indicate that De Gea's performance was being overestimated by the critic in the 2018/19 season. In addition, Neil Etheridge, who was ranked first from the PageRank algorithm, was ranked 14th with the score of -129.29 by the indirect wins method. A possible explanation for this difference may be that the indirect win method's score comprises of a win score and a lose score. Therefore, there is a penalty when players did not perform well. For example, we can see from the figure 4.2 that Etheridge had more than 10 matches with the points less than or equal to one while Alisson and Ederson did have at most two such matches. This would appear to indicate that the indirect win methods could capture consistency of players for the players ranking task better than the PageRank.

Now, we also look at five players with the highest indirect wins ranking, for each position in table 4.9. We can see that players in this ranking agree more with players ranking from FourFourTwo. For example, Sergio Aguero is ranked first in both tables. The indirect win approach shows a clear advantage over the PageRank algorithm in term of consistency with critics. We aware that our research may have some limitations. The first is that results from critics are not perfect standards for player ranking. There are always some disagreements among sports critics and fans concerning who is the best players in the league. Even results from FourFourTwo [41] and 90min [42] that we mentioned in this paper are not fully aligned. Secondly, Premier League Fantasy data involves only a certain aspect of players, and the data

Name	Team	Pos	Points	BPS	BPS/ game	Points /game	Score
Alisson Ramses Becker	Liverpool	1.00	176	771	20.29	4.63	1,487.67
Ederson Santana de Moraes	Man City	1.00	169	700	18.42	4.45	1,487.50
Hugo Lloris	Tottenham	1.00	145	691	20.94	4.39	819.19
Jordan Pickford	Everton	1.00	161	706	18.58	4.24	632.02
Kepa Arrizabalaga	Chelsea	1.00	142	685	19.03	3.94	419.38

Table 4.8: Five goalkeepers with the highest scores from indirect wins method with $\alpha = 0.5\alpha_{\max}$

Goalkeeper	**Defender**	**Midfielder/ Winger**	**Striker**
Alisson Ramses Becker	Andrew Robertson	Raheem Sterling	Sergio Agüero
Ederson Santana de Moraes	Trent Alexander-Arnold	Mohamed Salah	Roberto Firmino
Hugo Lloris	Virgil van Dijk	Sadio Mané	Harry Kane
Jordan Pickford	Aymeric Laporte	Eden Hazard	Pierre-Emerick Aubameyang
Kepa Arrizabalaga	Kyle Walker	Leroy Sané	Raúl Jiménez

Table 4.9: Five players with the highest indirect wins ranking for each position in the 2018/2019 season

is available to us in term of the summation of the score for each match. Our method is based on an assumption that a player perform well against another players if he has a higher fantasy point. Having a more complex data between a pair of players, such as the number of successful tackle a defender made against a striker, the number of successful dribbles that a striker made when facing a defender, would yield a better result. Our work clearly has some limitations. Nevertheless, we believe that our ranking method probably is usefully employed in the Football industry. For football enthusiasts, we also provide the result for the 2019/20 season from both PageRank and Indirect win method in the appendix. However, it is important to note that at as of April 2020, the Premier League has been suspended due to the Covid-19 and each team played only 29 matches.

Chapter 5

Player Chemistry

The aim of our work in this chapter was to broaden the current knowledge of football players chemistry. Players chemistry measures how well players played together, which can be in the form of assists or high passing rates, for example. Currently, there has been little discussion on finding players chemistry in the literature, perhaps because it is not clear how to define chemistry between players. A more recent work [34] was one of the first to propose a methodology to calculate players chemistry directly from on-the-ball action sequences data as in [33]. The data is very detailed; it contains second-by-second players' actions, as we can see in figure 1.2. We will give a quick review of their method. First, a performance metric for a player's action is defined based on its impact of that action on the player's team chances of scoring and conceding a goal. Let a_i^p be the ith action in the match which made by a player p, and $I(a_i^p)$ be the impact score of the action a_i^p. To calculate mutual chemistry between players p and q, they look for consecutive actions between p and q of the form (a_i^p, a_{i+1}^q) or (a_i^q, a_{i+1}^p) and define impact $I(a_i^p, a_{i+1}^q) = I(a_i^p) + I(a_{i+1}^q)$ as the sum of impact of each action. Now, joint offensive impact between players p, q in a match m is defined by

$$\mathrm{JOI}_m(p, q) = \sum_i I(a_i^p, a_{i+1}^q) + \sum_j I(a_j^q, a_{j+1}^p)$$

and a normalised joint offensive impact in a season is

$$\mathrm{JOI90}(p, q) = (\sum_m \mathrm{JOI}_m(p, q)) * \frac{90}{\text{number of minutes that } p \text{ and } q \text{ spent together}}$$

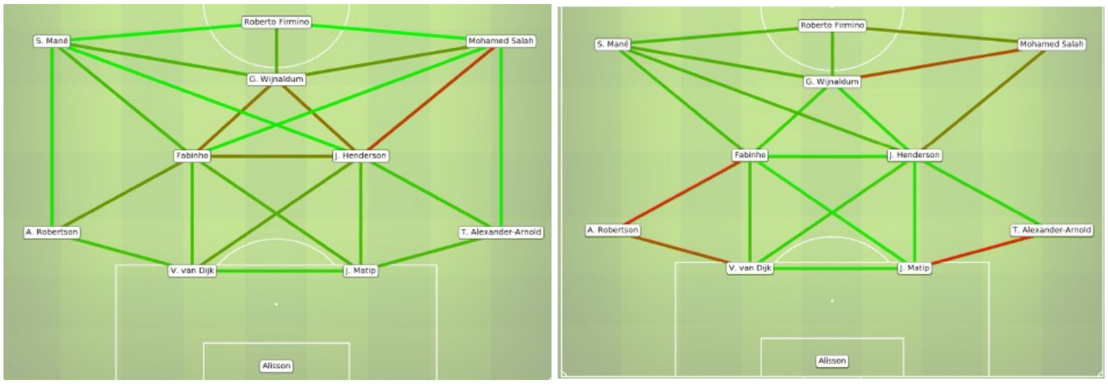

Figure 5.1: The mutual offensive chemistry (left) and the mutual defensive chemistry (right) for Liverpool players in the 2018/19 season

We assume that a pair of players with a higher JOI90 has higher offensive chemistry between them. The defensive chemistry is defined differently in the paper as the match event data only contains actions that happen, not actions that were prevented which is a key aspect of defending, but we will leave it for interested readers to look in the paper. Figure 5.1 shows the mutual offensive chemistry (left) and the mutual defensive chemistry (right) for Liverpool players in the 2018/19 season from [34]. We will compare our result with this chemistry which is to our knowledge the only available chemistry approach. Since our study is based on a limited dataset which is a Premier League Fantasy dataset, we will not be able to conduct research on a similar level to one in [34]. However, we seek to find an alternative way to derive players chemistry from only matches data in this paper.

5.1 Correlation method

First, we propose a correlation method on the player's scores. Since BPS is calculated based on various players' actions in each match (more actions than fantasy points), we may prefer the BPS over fantasy points when investigating players chemistry. In a football match, we define a good sequence of actions as a sequence of actions that made a positive impact on the player's team. For example, a sequence of actions that lead to a goal or a shot on target. Every player that involve in a good sequence of actions will gain credit in terms of BPS points. For instance, a goal scorer, a player who assisted, players who made a successful open-play cross(if existed), players who made key passes. We assume that when players with high chemistry play together in a match, there will be more good sequences of actions among them in the match. Although each player with different role gains different BPS from a good sequence of actions, there is still an upward trend between their score. Therefore, we hypothesise that two players with high chemistry will have a high correlation between their BPS score. We calculated the correlation of BPS score for any two Liverpool's player (only the matches that they played together) in the 2018/19 season and the result is shown in figure 5.2. It is crucial to note that with our data, we will not be able to derive offensive and defensive chemistry separately, and our results will be in the form of general chemistry. To assess our result, we will compare chemistry between forwards and midfielders with the offensive chemistry in figure 5.1 and the chemistry between midfielders and defenders with the defensive chemistry in figure 5.1.

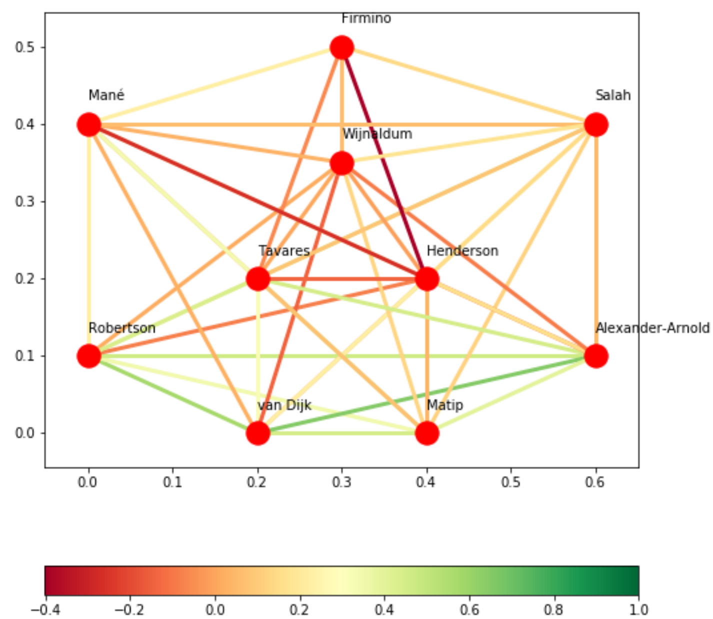

Figure 5.2: Liverpool players score correlation in the 2018/19 season

We can see that the result from our method struggles from the scaling problem where some correlations take negative values which were not expected for players in the same team. In addition, correlations are high among defenders, while the correlation between midfielders and strikers are much lower. Also, correlations between forwards are quite different from one in figure 5.1 (offensive). For example, the correlation between the Wijnaldum (midfielder) and Salah (winger) is higher than the correlation between Firmino (forward) and Salah (winger) which should be the other way round.

Alternatively, we can also associate the opponent teams' performance with our calculation. Note that, we must use fantasy points than the BPS when comparing players from different positions. Instead of calculating a correlation of BPS between 2 players A, B. We record fantasy points differences between A, B and players from the opponent team and calculate the correlation of that score differences instead. This method is very similar to calculating the correlation of BPS for each match, but we do it at players level. Figure 5.3 illustrates our process.

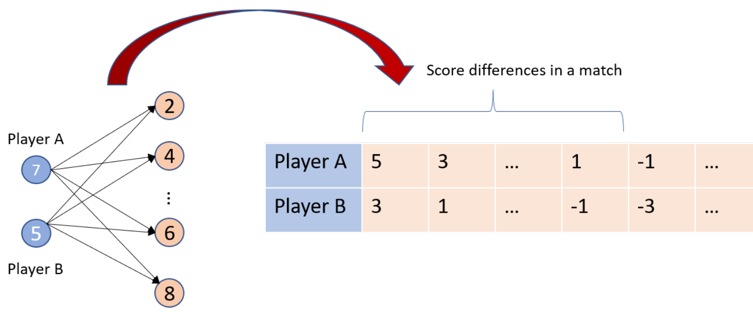

Figure 5.3

Our reason behind this approach is that for a pair of players A, B with high chemistry. If there is a player from an opponent team C that performs well against A. For example, A, B are strikers facing a defender C. If C defend well against A, there will be less good actions sequence that involves A. But A, B have high chemistry, so we assume that there will be less good actions sequence that involve B as well, which decrease both fantasy points of A and B. We assume that a score difference between A and C measures the performance of A over C. Therefore, we assume that high chemistry implies high correlation in score differences. One benefit of calculating correlations at players level is that it increases the number of data points from 1 data point per match to 11 data points per match for each player. This perhaps makes the calculation more robust. We are aware that BPS/fantasy points reward the same amount of point to goalkeepers and defenders if the team manage to get a clean sheet in a match. This might lead to a higher correlation between defenders and goalkeepers. Figure 5.4 illustrates a boxplot of correlation between players in each position for every team in the league. We denote G, D, M, S as goalkeeper, defender, midfielder/winger, striker respectively. It is clear that correlations between defenders are higher on average compare to other positions. Therefore, we normalised a correlation between two players with a position P_1, P_2 by adding

$$\text{the median of correlation } DD - \text{the median of correlation } P_1P_2$$

to their current correlation. For instance, if A is a striker and B is a midfielder, we increase their correlation by median DD − median MS.

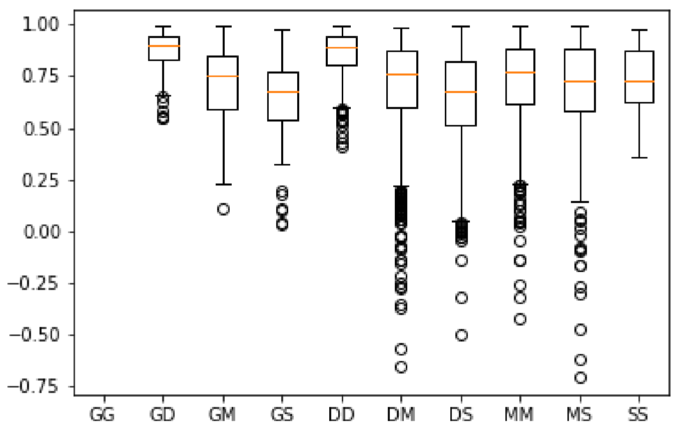

Figure 5.4: Boxplots of correlations between players in different positions in the 2018/19 season

The result after normalising is shown in figure 5.5. We can see that this method yield better scaling but may not reach a satisfactory level. Liverpool's forward, Mane, Salah and Firmino's chemistry is only around half of the chemistry between midfielders Wijnaldum, Tavares and Henderson. In addition, the most notable result in figure 5.1(offensive) is that chemistry between Salah and Henderson is the lowest in the team. It seems that our correlation method cannot capture this property as the chemistry between Salah and Henderson is quite high, as shown in figure 5.5. Another distinguishing feature in figure 5.1 (offensive) is chemistry among Liverpool's forwards, Mane, Firmino, Salah and the midfielder Wijnaldum. We can see higher chemistry between Mane, Firmino, Salah than the chemistry between these three players with Wijnaldum. Unfortunately, the correlation method has failed to address this feature as the correlation between 4 players are about the same as shown in figure 5.5. To sum up, the performances of both correlation methods were a little disappointing. The performances suffered from the scaling problem and low chemistry between forwards, which were expected to be higher. This is not unexpected as the current study was limited by the availability of data as well as many assumptions. More data would definitely help us to achieve a better result.

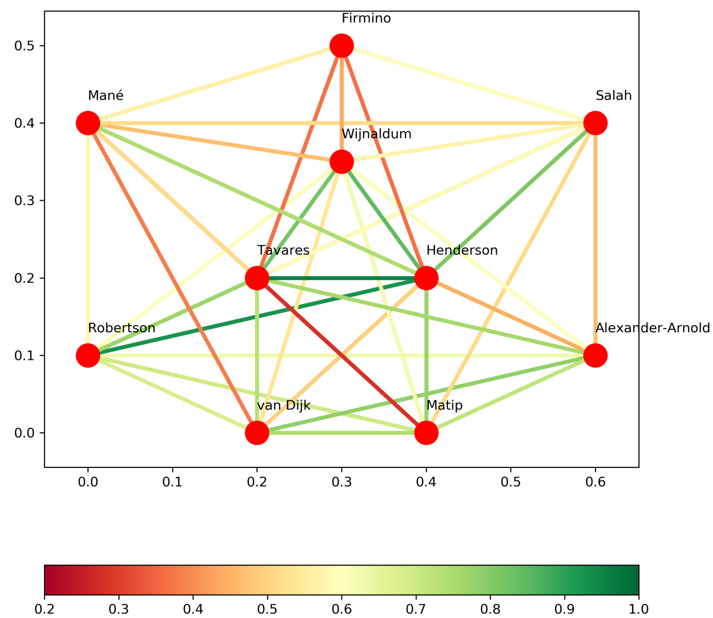

Figure 5.5: Liverpool players score correlation in the 2018/19 season after normalisation

5.2 Role-based similarity

In this section, we explore an application of role-based similarity in directed networks [36] to derive football players chemistry. First, we give a brief idea behind the role-based similarity and explain why it might be useful for deriving football players chemistry. For a network model, we generally regard a group of nodes with many connections within the group and fewer connections to external nodes as a community. There is a vast amount of literature on community detection based on this idea, such as Spectral clustering [43], Modularity method [44]. However, the authors in [36] argued that this idea could not be extended to some class of network such as directed networks where the direction of edges might contain important information. They introduced an alternative method for grouping nodes in a directed network based on their role in the network. By role, they meant the pattern of incoming and outgoing flows, and we cluster nodes with similar roles into the same group. For example, figure 5.6 from [36] give an example of a world trade network of manufacture of metals. Countries in the network are grouped based on their role into a core, a semi-periphery and a periphery country.

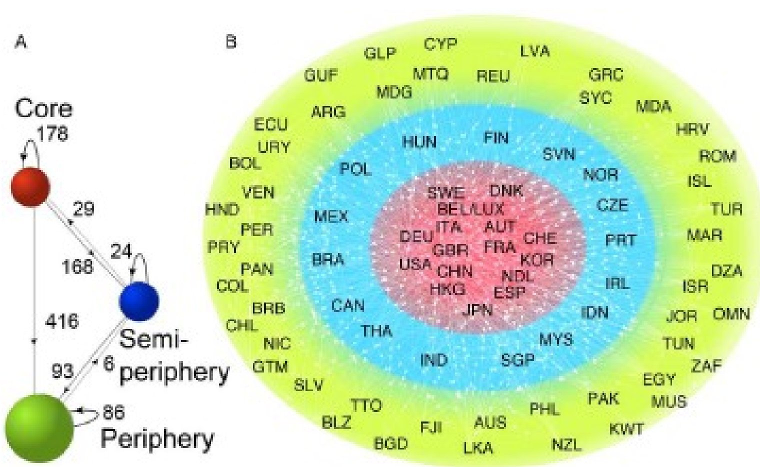

Figure 5.6: World trade network of manufacture of metals

Given that our works are based on a directed network of football players where there is no edge between players in the same team, it is impossible to apply the former idea that a community contains tightly connected nodes. The role-based similarity method might be a more suitable approach to our problem. Similar to the correlation method above, we assume that there are more good action sequences involving players with high chemistry. Therefore, players with high chemistry tend to have similar incoming and outgoing flow patterns. We construct a weighted directed network the same way we did in the player ranking chapter.

For a directed graph with n nodes and adjacency matrix A. We define the number of incoming paths and outgoing path of length k to a the ith node as $\mathrm{In}(i, k)$ and $\mathrm{Out}(i, k)$ respectively. These number is actually the number of indirect wins and loses at a distance k in the indirect wins method. For example, the number of incoming paths of length 2 to the ith node is

$$\begin{aligned}\mathrm{In}(i,2) &= \sum_{j,k} A_{kj}A_{ji} \\ &= \text{indirect wins at distance 2 for player i} \\ &= i\text{th index of } (A^T)^2 1\end{aligned}$$

when 1 is an $n \times 1$ matrix with entries 1. We can use induction to show that

$$\mathrm{In}(i,k) = i\text{th index of } (A^T)^k 1$$

and similarly

$$\mathrm{Out}(i,k) = i\text{th index of } (A)^k 1$$

Let $k_{\max}$ be the maximum distance of paths that we want to consider, construct a $n \times 2k_{\max}$ matrix as follows.

$$X = \left[\begin{array}{cccc|cccc} \alpha\mathrm{In}(1,1) & \alpha^2\mathrm{In}(1,2) & \dots & \alpha^{k_{\max}}\mathrm{In}(1,k_{\max}) & \alpha\mathrm{Out}(1,1) & \alpha^2\mathrm{Out}(1,2) & \dots & \alpha^{k_{\max}}\mathrm{Out}(1,k_{\max}) \\ \alpha\mathrm{In}(2,1) & \alpha^2\mathrm{In}(2,2) & \dots & \alpha^{k_{\max}}\mathrm{In}(2,k_{\max}) & \alpha\mathrm{Out}(2,1) & \alpha^2\mathrm{Out}(2,2) & \dots & \alpha^{k_{\max}}\mathrm{Out}(2,k_{\max}) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha\mathrm{In}(n,1) & \alpha^2\mathrm{In}(n,2) & \dots & \alpha^{k_{\max}}\mathrm{In}(n,k_{\max}) & \alpha\mathrm{Out}(n,1) & \alpha^2\mathrm{Out}(n,2) & \dots & \alpha^{k_{\max}}\mathrm{Out}(n,k_{\max}) \end{array}\right]$$

Where $0 < \alpha < \lambda_{\max}^{-1}$ when $\lambda_{\max}$ is the highest eigenvalue of A . We denote the ith row of X as X_i which represents the incoming and outgoing flow of the ith node, up to the distance $k_{\max}$. We choose a cosine similarity metric and the construct a corresponding similarity matrix Y:

$$Y_{ij} = \frac{X_i X_j^T}{||X_i|| \cdot ||X_j||}$$

The original paper used spectral clustering as in [43] to cluster nodes based on the similarity matrix Y but we will use Y as a chemistry matrix where Y_{ij} denote the chemistry between ith and jth player. The result from the role-based similarity with $k_{\max} = 20, \alpha = 0.8\lambda_{\max}^{-1}$ is shown in figure 5.7.

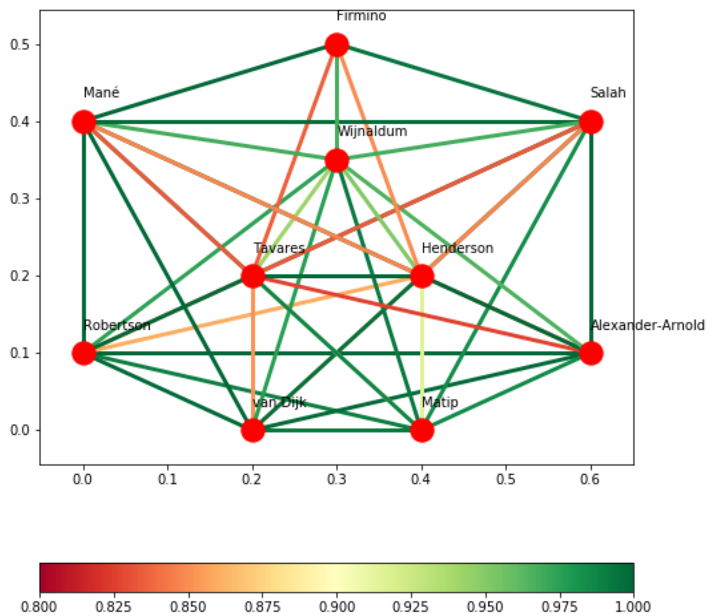

Figure 5.7: Liverpool players chemistry from the role-based similarity in the 2018/19 season

We can see that the resulted chemistry is better scaled with values between $0.8 - 1.0$. Chemistry between the Liverpool's forwards are on the same scale as the chemistry between Liverpool's defenders. Moreover, it is clear from the figure that adjacent players tend to have higher mutual chemistry which was expected in reality. The result can capture higher chemistry among Mane, Firmino, Salah than the chemistry between these three players with Wijnaldum which, was observed in figure 5.1(offensive). Also, the chemistry between Salah and Henderson is lower than the chemistry between Salah and other players, which aligned with the result from in figure 5.1(offensive). The evidence from this study suggests that the role-based similarity could give comparable chemistry to chemistry from [34] using much fewer data. Nevertheless, the role-based model suggested low chemistry between Henderson and Mane and between Fabinho(Tavares) and Salah, which was high in figure 5.1(offensive). All in all, the results point to the likelihood that a role-based method can provide a plausible players chemistry at least for Liverpool in the 2018/19 season. Since, Liverpool's chemistry in the 2018/19 season is the only result available up to this point (April 2020), to confirm that the role-based method is a suitable approach for deriving players chemistry, further data collection is required.

Chapter 6

Appendix

The following are ranking results for the Premier League season 2019/20 from the PageRank and Indirect wins method.

Goal keeper	**Defender**	**Midfielder/ Winger**	**Striker**
Nick Pope	James Tarkowski	Kevin De Bruyne	Jamie Vardy
Ben Foster	John Lundstram	Mohamed Salah	Pierre-Emerick Aubameyang
David de Gea	Trent Alexander-Arnold	Sadio Mané	Tammy Abraham
Martin Dubravka	Ricardo Pereira	Heung-Min Son	Marcus Rashford
Vicente Guaita	Ben Mee	Richarlison de Andrade	Teemu Pukki

Table 6.1: Five players with the highest PageRank coefficient with capped score differences for each position in the 2019/20

Goal keeper	**Defender**	**Midfielder/ Winger**	**Striker**
Alisson Ramses Becker	Trent Alexander-Arnold	Mohamed Salah	Jamie Vardy
Dean Henderson	Virgil van Dijk	Sadio Mané	Marcus Rashford
Kasper Schmeichel	Andrew Robertson	Kevin De Bruyne	Sergio Agüero
Hugo Lloris	Marcos Alonso	Riyad Mahrez	Roberto Firmino
Rui Patrício	Joseph Gomez	Heung-Min Son	Pierre-Emerick Aubameyang

Table 6.2: Five players with the highest indirect wins ranking for each position in the 2019/2020 season

The following are codes for this dissertation which is orgainsed as follows, Downloading data, Functions, PageRank, Indirect wins approach, BPS correlation, Score difference correlation and Role-based similarity.

6.1 Download data and functions

```
################## DOWNLOAD DATA AND FUNCTION ###################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import copy
import networkx as nx
from scipy.stats import pearsonr
import os
pd.options.mode.chained_assignment = None  # default='warn'
from sklearn.cluster import SpectralClustering

#Download player data
player_raw = pd.read_csv(os.getcwd()+'/data/2018-19/players_raw.csv', encoding =
     "ISO-8859-1")
player_index = ['first_name','second_name','team','id',
                'total_points', 'points_per_game',
                'form', 'influence','creativity','threat','ict_index','
    element_type']

#Dictionary
position = {player_raw[player_index]['id'][i]: player_raw[player_index]['
    element_type'][i] for i in range(len(player_raw[player_index]))}

teamid_to_team = {1: 'Arsenal',
 2: 'Bournemouth',
 3: 'Brighton',
 4: 'Burnley',
 5: 'Cardiff',
 6: 'Chelsea',
 7: 'Crystal Palace',
 8: 'Everton',
 9: 'Fulham',
 10: 'Huddersfield',
```

```
 11: 'Leceicester',
 12: 'Liverpool',
 13: 'Man City',
 14: 'Man Utd',
 15: 'Newcastle Utd',
 16: 'Southampton',
 17: 'Tottenham',
 18: 'Watford',
 19: 'West Ham',
 20: 'Wolves'}

playerid_to_team = {player_raw['id'][i]:teamid_to_team[player_raw['team'][i]]
                   for i in range(len(player_raw))}

#Fixture
fixtures = pd.read_csv(os.getcwd()+'/data/2018-19/fixtures.csv')
fixtures_column = ['id','team_a','team_h', 'kickoff_time', 'finished',
                   'team_a_score','team_h_score','team_a_difficulty','
    team_h_difficulty']
fixtures = fixtures[fixtures_column]
fixture_list = []
for team_id in range(21):
    fixture_home = fixtures[fixtures['team_h'] == team_id]
    fixture_away = fixtures[fixtures['team_a'] == team_id]
    length_a = len(fixture_away)
    length_h = len(fixture_home)

    #home or away if home 0, away 1
    fixture_home.loc[:,'away'] = ([0]*length_h)
    fixture_away.loc[:,'away'] = ([1]*length_a)
    fixtures_i = pd.concat([fixture_home, fixture_away]).sort_values(by=['
    kickoff_time'])
    #opponent id for team i
    fixtures_i['opponent_id'] = fixtures_i['away']*fixtures_i['team_h'] + (1-
    fixtures_i['away'])*fixtures_i['team_a']
    # difficulty team i face
    fixtures_i['team i difficulty'] = (1-fixtures_i['away'])*fixtures_i['
    team_h_difficulty'] + (fixtures_i['away'])*fixtures_i['team_a_difficulty']

    fixtures_i = fixtures_i.reset_index(drop = True)
    fixture_list.append(fixtures_i)
```

```
#Raw datafor each week
gw_matrix = []
for i in range(1,39):
    gameweek_str = os.getcwd()+'/data/2018-19/gws/'+'gw' + str(i) +'.csv'
    gw = pd.read_csv(gameweek_str,encoding = "ISO-8859-1")
    gw_matrix.append(gw)

#Function
# adding player id column into the data frame
def extract_index(gw):
    id_list = []
    surname_list = []
    for i in range(len(gw)):
        player = gw['name'][i]
        player_split = player.split('_')
        id_list.append(player_split[2])
        surname_list.append(player_split[1])
    gw['player_id'] = id_list
    gw['surname'] = surname_list
    gw['position'] = [position[int(gw['player_id'][j])] for j in range(len(gw))]
    return gw

def concat_df(gw_list):
    concat_list = gw_list[0]
    for i in range(1, len(gw_list)):
        concat_list = pd.concat([concat_list, gw_list[i]])
    concat_list = concat_list.reset_index(drop = True)
    return concat_list

def getdata_team(gw_matrix, team_code, gw_column = ['name', 'minutes', 'total_points',  'bonus', 'bps']):
    #Look at Man United fixtures
    fixture_i = fixture_list[team_code]
    gw_team_list = []

    #Run through each gameweek
    for i in range(len(gw_matrix)):
        gw_i = gw_matrix[i]


```

```python
        #select players from the opponent team index
        opponent_team = fixture_i['opponent_id'][i]
        gw_team = gw_i[gw_i['opponent_team'] == opponent_team]

        #select columns we are interested in
        gw_team = gw_team[gw_column]
        gw_team = gw_team.reset_index(drop = True)
        #extract player id
        gw_team = extract_index(gw_team)

        gw_team_list.append(gw_team)
    return gw_team_list


#Pick only players that played more than 60mins
gw_column = ['name', 'minutes', 'total_points',  'bonus', 'bps','position', 'player_id','surname','fixture','opponent_team']
gw_list_60 = []
mins = 60
for i in range(len(gw_matrix)):
    gw_i = gw_matrix[i]
    gw_i = gw_i.reset_index(drop = True)
    #extract player id
    gw_i = extract_index(gw_i)
    #select column
    gw_i = gw_i[gw_column]
    gw_i['match'] = 1
    #only choose players that played
    gw_i = gw_i[gw_i['minutes'] > mins]
    gw_list_60.append(gw_i)

#Concatenate data
all_gw60 = concat_df(gw_list_60)
#reset index again
all_gw60 = all_gw60.reset_index(drop = True)

all_gw60['team'] = 0
all_gw60['player_id'] = all_gw60['player_id'].astype(int)
for i in range(len(all_gw60)):
    all_gw60['team'][i] = playerid_to_team[all_gw60['player_id'][i]]

```

```python

#Dictionary
id_to_player = {all_gw60['player_id'][i]: all_gw60['name'][i] for i in range(len(all_gw60))}
id_to_player_short = {all_gw60['player_id'][i]: all_gw60['name'][i].split('_')[1] for i in range(len(all_gw60))}

#Generate Adjacency matrix
def gen_array(all_gw_mat = all_gw60, method = 'total_points', low_to_high = True, unweighted = False):


    id_list = list(all_gw_mat['player_id'].unique())
    id_list.sort()

    #id_to_location
    id_to_loc = {id_list[i]:i for i in range(len(id_list))}
    loc_to_id = {i: id_list[i] for i in range(len(id_list))}
    #create an array for player
    n = len(id_list)
    player_array = np.zeros((n,n))
    number_array = np.zeros((n,n))

    #update by using a for loop in fixtures
    for ii in range(len(fixtures)):
        player_in_match = all_gw_mat[all_gw_mat['fixture'] == fixtures['id'][ii]]
        team_a = player_in_match[all_gw_mat['opponent_team'] == fixtures['team_h'][ii]].reset_index(drop = True)
        team_h = player_in_match[all_gw_mat['opponent_team'] == fixtures['team_a'][ii]].reset_index(drop = True)

        if(low_to_high == True):
            for i in range(len(team_a)):
                for j in range(len(team_h)):

                    score = team_a[method][i] - team_h[method][j]
                    index_i = id_to_loc[team_a['player_id'][i]]
                    index_j = id_to_loc[team_h['player_id'][j]]

                    if(score > 0):
```

```
                        if(unweighted == True):
                            player_array[index_j,index_i] += 1
                            number_array[index_j,index_i] += 1
                        else:
                            player_array[index_j,index_i] += score
                            number_array[index_j,index_i] += 1

                    elif(score < 0):
                        if(unweighted == True):
                            player_array[index_i,index_j] += 1
                            number_array[index_i,index_j] += 1
                        else:
                            player_array[index_i,index_j] += -score
                            number_array[index_i,index_j] += 1

        else:
            for i in range(len(team_a)):
                for j in range(len(team_h)):

                    score = team_a[method][i] - team_h[method][j]
                    index_i = id_to_loc[team_a['player_id'][i]]
                    index_j = id_to_loc[team_h['player_id'][j]]

                    if(score > 0):
                        if(unweighted == True):
                            player_array[index_i,index_j] += 1
                            number_array[index_i,index_j] += 1
                        else:
                            player_array[index_i,index_j] += score
                            number_array[index_i,index_j] += 1
        #                 print('yay')
                    elif(score < 0):
                        if(unweighted == True):
                            player_array[index_j,index_i] += 1
                            number_array[index_j,index_i] += 1
                        else:
                            player_array[index_j,index_i] += -score
                            number_array[index_j,index_i] += 1

    for ii in range(n):
        for jj in range(n):
```

```
            if(number_array[ii,jj] > 0):
                player_array[ii,jj] = player_array[ii,jj]/number_array[ii,jj]


    return player_array, loc_to_id
```

6.2 Players Ranking

```
###################### PLAYER RANKING #####################
#Summarise matrix
all_gw_sum = all_gw60.groupby(['name'], as_index = False).sum()
all_gw_sum['BPS/game'] = all_gw_sum['bps']/all_gw_sum['match']
all_gw_sum['point/game'] = all_gw_sum['total_points']/all_gw_sum['match']
all_gw_sum['position'] = all_gw_sum['position']/all_gw_sum['match']
all_gw_sum['BPS/min'] = all_gw_sum['bps']/all_gw_sum['minutes']
all_gw_sum['point/min'] = all_gw_sum['total_points']/all_gw_sum['minutes']

all_gw_sum['team'] = 0
for i in range(len(all_gw_sum)):
    all_gw_sum['team'][i] = playerid_to_team[all_gw_sum['player_id'][i]/
    all_gw_sum['match'][i]]

############### PageRank
player_array, loc_to_id = gen_array(all_gw_mat = all_gw60, method = '
    total_points', low_to_high = True)
G = nx.Graph(player_array)
rank_G = nx.pagerank(G)

rank_G_sorted = {k: v for k, v in sorted(rank_G.items(), key=lambda item: item
    [1], reverse = True)}
rank_G_list = [[id_to_player[loc_to_id[k]],v] for k,v in rank_G_sorted.items()]
all_gw_sum['PR_point_w'] =  0
for elem in rank_G_list:
    all_gw_sum['PR_point_w'][all_gw_sum['name'] == elem[0]] = elem[1]

######Indirect win
def indirect_rank(A = player_array, alpha_ratio = 0.85):
    k_in = np.sum(A, axis = 1)
    k_out = np.sum(A, axis = 0)
    n = len(player_array)

```

```
    eigval, eigvec =np.linalg.eig(A)
    l_max = max(eigval)
    alpha_max = 1/l_max
    alpha = alpha_ratio*alpha_max

    #score
    win = np.matmul(np.linalg.inv(np.identity(n) - alpha*np.transpose(A)), k_out
    )
    lose = np.matmul(np.linalg.inv(np.identity(n) - alpha*A), k_in)

    return (win-lose).astype(float)

#Indirect win with 0.8alpha_max
player_array, loc_to_id = gen_array(all_gw_mat = all_gw60, method = '
    total_points', low_to_high = True, unweighted = False)
ind_rank = indirect_rank(player_array, alpha_ratio = 0.8)
ind_rank_list = [[id_to_player[loc_to_id[i]], ind_rank[i] ] for i in range(len(
    ind_rank))]
all_gw_sum['indirect_win_points_w_08'] =  0
for elem in ind_rank_list:
    all_gw_sum['indirect_win_points_w_08'][all_gw_sum['name'] == elem[0]] = elem
    [1]
```

6.3 Players chemistry

```
########## PLAYER CHEMISTRY #######
#### Visualisation
#visulaise function
def visualise_community(correlation_newway, name, community =
    [1,1,1,1,1,1,1,1,1,1], vmin = 0.8, vmax = 1):
    A = np.array(correlation_newway)

    G = nx.Graph(A)
       # Remove edge to match a pic
    edge_to_remove = [(9,0),(9,1),(9,2),(9,3)]
    for edge in edge_to_remove:
        G.remove_edge(edge[0],edge[1])
    pos = {2: [0.4,0],
     3: [0.6, 0.1],
     1: [0.2,0],
     0: [0, 0.1],
```

```
     5: [0.4, 0.2],
     6: [0,0.4],
     7: [0.3,0.35],
     8: [0.6,0.4],
     4: [0.2, 0.2],
     9: [0.3,0.5]}
    edges,weights = zip(*nx.get_edge_attributes(G,'weight').items())
    fig = plt.figure(figsize = (8,9))
    nodes = nx.draw_networkx_nodes(G,pos,node_color='r', with_labels=False)
    #change rom node_color = 'r' to node_color = community if we want node
    colors based on community
    edges = nx.draw_networkx_edges(G, pos, edge_color = weights, width = 3,
    edge_cmap = plt.cm.RdYlGn, edge_vmin = vmin,edge_vmax= vmax)
    #name
    cur_namedict = {i: name[i] for i in range(len(name))}
    for cur_pos in pos:
        x,y = pos[cur_pos]
        plt.text(x,y+0.03, cur_namedict[cur_pos], fontsize = 10)

    plt.colorbar(edges,orientation='horizontal')
    plt.show()
###### BPS correlation
# normal correlation from BPS
gw_liv_list = getdata_team(gw_matrix, team_code = 12)

liv_weekly_score = pd.DataFrame()
#concat all liv u data
all_liv_game = concat_df(gw_liv_list)
liv_weekly_score['name'] = all_liv_game['name'].unique()
liv_weekly_score = extract_index(liv_weekly_score)
for i in range(len(gw_liv_list)):
    this_week = gw_liv_list[i].reset_index(drop = True)
    liv_weekly_score['week'+ str(i+1)] = np.nan
    for j in range(len(this_week)):
        cur_bps = this_week['bps'][j]
        cur_id = this_week['player_id'][j]
        liv_weekly_score['week'+ str(i+1)][liv_weekly_score['player_id'] ==
    cur_id] = cur_bps

#drop players that play less than 3 matches
index = (np.sum(liv_weekly_score.iloc[:,4:].notnull(), axis =1) >=3)
```

```
liv_weekly_score = liv_weekly_score[index].sort_values(by=['player_id']).reset_index(drop = True)

index = (np.sum(liv_weekly_score.iloc[:,3:].notnull(), axis =1) >2)
liv_weekly_score = liv_weekly_score[index].sort_values(by=['player_id']).reset_index(drop = True)
liv_weekly_score_T = liv_weekly_score.transpose()
liv_weekly_score_T = liv_weekly_score_T.iloc[3:,:]
liv_weekly_score_T.columns = liv_weekly_score['surname']

l = len(liv_weekly_score)
modify_corr = pd.DataFrame(0.0, index = liv_weekly_score['surname'], columns = liv_weekly_score['surname'])
for i in range(l):
    for j in range(l):
        score_i = liv_weekly_score_T.iloc[:,i]
        score_j = liv_weekly_score_T.iloc[:,j]
        index = score_i.notnull()*score_j.notnull()
        score_i = score_i[index]
        score_j = score_j[index]
        #If there are no interception, we can't find a correlation and set it equals to zero
        if(len(score_i) >2 ):
            correlation = pearsonr(score_i, score_j)[0]

        else:
            correlation = 0
        modify_corr.iloc[i,j] = correlation
        modify_corr.iloc[j,i] = correlation

col = ['Robertson',
 'van Dijk',
 'Matip',
 'Alexander-Arnold',
 'Tavares',
 'Henderson',
 'Man ',
 'Wijnaldum',
 'Salah',
 'Firmino']
modify_corr = modify_corr.loc[:,col]
```

```
modify_corr = modify_corr.loc[col,:]
plt.figure(figsize=(10,8))
visualise_community(modify_corr, name = modify_corr.columns, vmin = -0.3, vmax =
     0.8)
plt.show()

####### Score difference
# fixture of team A
def gen_array_for_corr(team_code):
    method = 'total_points'
    match_len = 38
    team_A = team_code
    fixture_A = fixture_list[team_A]
    fixture_A = fixture_A[fixture_A['finished'] == True]
    fixture_A = fixture_A.iloc[ : match_len]


    #matrix that contain all players from team A
    score_matrixA = pd.DataFrame()
    gw_teamA = getdata_team(gw_matrix, team_code = team_A)
    all_teamA = concat_df(gw_teamA)
    score_matrixA['name'] = all_teamA['name'].unique()


    #extract index
    id_list = []
    surname_list = []
    for i in range(score_matrixA.shape[0]):
        player = score_matrixA['name'][i]
        player_split = player.split('_')
        id_list.append(player_split[2])
        surname_list.append(player_split[1])
    score_matrixA['player_id'] = id_list
    score_matrixA['surname'] = surname_list
    score_matrixA['position'] = [position[int(score_matrixA['player_id'][j])]
    for j in range(len(score_matrixA))]


    #iterate over number of matches we are interested in
    for i in range(match_len):
        #match id
```

```
        match_id = fixture_A['id'][i]
        team_list = [fixture_A.iloc[i,j] for j in [1,2]]
        team_list.remove(team_A)
        opponent_id = team_list[0]
        my_id = team_A



        #identify players from team A, team B
        gw_i = gw_matrix[i]
        team_A_player = gw_i[gw_i['fixture'] == match_id][gw_i['opponent_team'] == opponent_id].reset_index(drop = True)
        team_B_player = gw_i[gw_i['fixture'] == match_id][gw_i['opponent_team'] == team_A].reset_index(drop = True)

        #drop players that play less than 60 mins
        team_A_player = team_A_player[team_A_player['minutes'] > 60].reset_index(drop = True)
        team_B_player = team_B_player[team_B_player['minutes'] > 60].reset_index(drop = True)

        for ii in range(team_B_player.shape[0]):
            score_B = team_B_player[method][ii]
            player_name = str(team_B_player['name'][ii]+' ' +str(i))
#             print(player_name)
            score_matrixA[player_name]  = np.nan
            for jj in range(team_A_player.shape[0]):
                score_A = team_A_player[method][jj]
                player_nameA= team_A_player['name'][jj]
                score_matrixA[player_name][score_matrixA['name'] == player_nameA] = score_A -score_B

#         print(team_A_player)
#         print(team_B_player)

        #update matrix !!!


score_matrixA = score_matrixA.sort_values(by = ['player_id'])
#drop player who play less than 3 matches
index = (np.sum(score_matrixA.iloc[:,1:].notnull(), axis =1) >30)
```

```
    score_matrixA = score_matrixA[index].reset_index(drop = True)
    return score_matrixA

def cal_corr_nonan(matrix):
    l = matrix.shape[0]
    modify_corr = pd.DataFrame(0.0, index = matrix['surname'], columns = matrix['surname'])
    position_matrix = pd.DataFrame(0.0, index = matrix['surname'], columns = matrix['surname'])

    #     print(l)
    for i in range(l):
    #         print(i)
        for j in range(l):
    #             print(j)
            score_i = matrix.iloc[i,4:]
            score_j = matrix.iloc[j,4:]
            #calculate not null index
            index = score_i.notnull()&score_j.notnull()
            score_i = score_i[index]
            score_j = score_j[index]
    #             print(len(score_i))
            correlation = np.nan
            if(len(score_i) >20):
                correlation = pearsonr(score_i, score_j)[0]
                modify_corr.iloc[i,j] = correlation
                modify_corr.iloc[j,i] = correlation
            else:
                modify_corr.iloc[i,j] = correlation
                modify_corr.iloc[j,i] = correlation

            position_matrix.iloc[i,j] = str(matrix.iloc[i,3])+'_'+str(matrix.iloc[j,3])
            position_matrix.iloc[j,i] = str(matrix.iloc[j,3])+'_'+str(matrix.iloc[i,3])
    return modify_corr, position_matrix

corr_mat_list = []
position_mat_list = []
for i in range(1,21):
    #gen matrix
```

```
    print(i)
    score_mat_tem = gen_array_for_corr(team_code = i)
    corr_mat_tem, position_mat_tem = cal_corr_nonan(score_mat_tem)

    corr_mat_list.append(corr_mat_tem)
    position_mat_list.append(position_mat_tem)
corr_11 = []
corr_12 = []
corr_13 = []
corr_14 = []
corr_22 = []
corr_23 = []
corr_24 = []
corr_33 = []
corr_34 = []
corr_44 = []
for ii in range(20):
    position_mat = position_mat_list[ii]
    corr_mat = corr_mat_list[ii]
    for i in range(len(position_mat)):
        for j in range(i,len(position_mat)):
            #not nan
            if(corr_mat.iloc[i,j] == corr_mat.iloc[i,j]):
                if(corr_mat.iloc[i,j] < 0.99):
                    if(position_mat.iloc[i,j] == '1_1'):
                        corr_11.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '1_2' or position_mat.iloc[i,
    j] == '2_1'):
                        corr_12.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '1_3' or position_mat.iloc[i,
    j] == '3_1'):
                        corr_13.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '1_4' or position_mat.iloc[i,
    j] == '4_1'):
                        corr_14.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '2_2'):
                        corr_22.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '2_3' or position_mat.iloc[i,
    j] == '3_2'):
                        corr_23.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '2_4' or position_mat.iloc[i,
```

```
    j] == '4_2'):
                        corr_24.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '3_3'):
                        corr_33.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '3_4' or position_mat.iloc[i,j] == '4_3'):
                        corr_34.append(corr_mat.iloc[i,j])
                    elif(position_mat.iloc[i,j] == '4_4'):
                        corr_44.append(corr_mat.iloc[i,j])
factor_11 = 0
factor_22 = 0
factor_33 = np.median(corr_22)-np.median(corr_33)
factor_44 = np.median(corr_22)-np.median(corr_44)
factor_12 = np.median(corr_22)-np.median(corr_12)
factor_23 = np.median(corr_22)-np.median(corr_23)
factor_34 = np.median(corr_22)-np.median(corr_34)
factor_13 = np.median(corr_22)-np.median(corr_13)
factor_24 = np.median(corr_22)-np.median(corr_24)
factor_14 = np.median(corr_22)-np.median(corr_14)

position_mat = position_mat_list[11]
corr_mat =  corr_mat_list[11]
for i in range(len(position_mat)):
    for j in range(i,len(position_mat)):
        #not nan
        if(corr_mat.iloc[i,j] == corr_mat.iloc[i,j]):
             if(corr_mat.iloc[i,j] < 0.98):
                if(position_mat.iloc[i,j] == '1_1'):
                    corr_mat.iloc[i,j] += factor_11
                    corr_mat.iloc[j,i] += factor_11
                elif(position_mat.iloc[i,j] == '1_2' or position_mat.iloc[i,j] == '2_1'):
                    corr_mat.iloc[i,j] += factor_12
                    corr_mat.iloc[j,i] += factor_12
                elif(position_mat.iloc[i,j] == '1_3' or position_mat.iloc[i,j] == '3_1'):
                    corr_mat.iloc[i,j] += factor_13
                    corr_mat.iloc[j,i] += factor_13
                elif(position_mat.iloc[i,j] == '1_4' or position_mat.iloc[i,j] == '4_1'):
                    corr_mat.iloc[i,j] += factor_14
```

```
                    corr_mat.iloc[j,i] += factor_14
                elif(position_mat.iloc[i,j] == '2_2'):
                    corr_mat.iloc[i,j] += factor_22
                    corr_mat.iloc[j,i] += factor_22
                elif(position_mat.iloc[i,j] == '2_3' or position_mat.iloc[i,j] == '3_2'):
                    corr_mat.iloc[i,j] += factor_23
                    corr_mat.iloc[j,i] += factor_23
                elif(position_mat.iloc[i,j] == '2_4' or position_mat.iloc[i,j] == '4_2'):
                    corr_mat.iloc[i,j] += factor_24
                    corr_mat.iloc[j,i] += factor_24
                elif(position_mat.iloc[i,j] == '3_3'):
                    corr_mat.iloc[i,j] += factor_33
                    corr_mat.iloc[j,i] += factor_33
                elif(position_mat.iloc[i,j] == '3_4' or position_mat.iloc[i,j] == '4_3'):
                    corr_mat.iloc[i,j] += factor_24
                    corr_mat.iloc[j,i] += factor_24
                elif(position_mat.iloc[i,j] == '4_4'):
                    corr_mat.iloc[i,j] += factor_44
                    corr_mat.iloc[j,i] += factor_44

row = ['Robertson', 'van Dijk', 'Matip','Alexander-Arnold','Tavares','Henderson', 'Man ', 'Wijnaldum', 'Salah', 'Firmino']
liverpool_start = corr_mat[row].loc[row, :]
plt.figure(figsize = (10,8))
visualise_community(liverpool_start, name = liverpool_start.columns, vmin = 0.2, vmax = 1.0)
plt.show()

##### Role-based similarity
# role based correlation
from sklearn.metrics.pairwise import cosine_similarity as cosine_sim
player_array, loc_to_id = gen_array(all_gw_mat = all_gw60, method = 'total_points', low_to_high = True, unweighted = False)
A = player_array
A_T = np.transpose(player_array)

#calculate eigenvalue
eigval, eigvec =np.linalg.eig(A)
```

```
l_max = max(eigval)
alpha_rat = 0.8
alpha_max = 1/l_max
alpha = alpha_rat*alpha_max

B = A.copy()
k_max = 20

n = len(B)
one = np.ones((n,1))
X = np.zeros((n,1))
B_k = alpha*B.copy()
for i in range(k_max):
    In_k = np.matmul(B_k,one)
    B_k =  np.matmul(B_k, B)*alpha
    X = np.hstack((X,In_k))


B_T = np.transpose(B)
B_T_k = alpha*B_T.copy()
for i in range(k_max):
    Out_k = np.matmul(B_T_k,one)
    B_T_k = np.matmul(B_T_k, B_T)*alpha
    X = np.hstack((X,Out_k))

X = X[:,1:].astype(float)
Y = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        Y[i,j] = cosine_sim([X[i,:]],[X[j,:]])
#calculate similarity matrix Y

#visualise role based similarity for liverpool
id_to_loc = {v:u for u,v in loc_to_id.items()}
liv_id = [247,246,243,245,255,249,251,252,253,257]
liv_loc = [id_to_loc[i] for i in liv_id]
liv_name = [id_to_player_short[i] for i in liv_id]
Y_liverpool = Y[liv_loc,:][:,liv_loc]
clustering = SpectralClustering(n_clusters = 3, affinity = 'precomputed').fit(
    Y_liverpool)
label = clustering.labels_
```

```
visualise_community(Y_liverpool, name = liv_name)
```

Bibliography

[1] S. Motegi and N. Masuda, "A network-based dynamical ranking system for competitive sports," *Scientific reports*, vol. 2, p. 904, 2012.

[2] X. Wei, P. Lucey, S. Morgan, and S. Sridharan, "Predicting shot locations in tennis using spatiotemporal data," in *2013 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–8, IEEE, 2013.

[3] J. Hucaljuk and A. Rakipović, "Predicting football scores using machine learning techniques," in *2011 Proceedings of the 34th International Convention MIPRO*, pp. 1623–1627, IEEE, 2011.

[4] V. Lazova and L. Basnarkov, "Pagerank approach to ranking national football teams," *arXiv preprint arXiv:1503.01331*, 2015.

[5] J. L. Pena and H. Touchette, "A network theory analysis of football strategies," *arXiv preprint arXiv:1206.6904*, 2012.

[6] C. K. Leung and K. W. Joseph, "Sports data mining: predicting results for the college football games," *Procedia Computer Science*, vol. 35, pp. 710–719, 2014.

[7] T. Callaghan, P. J. Mucha, and M. A. Porter, "Random walker ranking for ncaa division ia football," *The American Mathematical Monthly*, vol. 114, no. 9, pp. 761–777, 2007.

[8] J. Park and M. E. Newman, "A network-based ranking system for us college football," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 10, p. P10014, 2005.

[9] S. C. Albright, "A statistical analysis of hitting streaks in baseball," *Journal of the american statistical association*, vol. 88, no. 424, pp. 1175–1183, 1993.

[10] M. Marchi and J. Albert, *Analyzing baseball data with R.* CRC Press, 2013.

[11] M. Lewis, *Moneyball: The art of winning an unfair game.* WW Norton & Company, 2004.

[12] J. Wu, "Diana ma, this laker's data scientist is nba's best kept secret," *Forbes*, Feb 2020.

[13] "Training ground guru - manchester city create new first team data science role," 2019.

[14] S. Wasserman, K. Faust, *et al.*, *Social network analysis: Methods and applications*, vol. 8. Cambridge university press, 1994.

[15] P. J. Carrington, J. Scott, and S. Wasserman, *Models and methods in social network analysis*, vol. 28. Cambridge university press, 2005.

[16] N. P. Hummon and P. Dereian, "Connectivity in a citation network: The development of dna theory," *Social networks*, vol. 11, no. 1, pp. 39–63, 1989.

[17] S. A. Greenberg, "How citation distortions create unfounded authority: analysis of a citation network," *Bmj*, vol. 339, p. b2680, 2009.

[18] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[19] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in social media," *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 515–554, 2012.

[20] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports*, vol. 533, no. 4, pp. 95–142, 2013.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," tech. rep., Stanford InfoLab, 1999.

[22] X. REN *et al.*, "Review of ranking nodes in complex networks," *Chinese Science Bulletin*, vol. 59, no. 13, pp. 1175–1197, 2014.

[23] R. Lambiotte, "C5.4 networks." `https://courses.maths.ox.ac.uk/node/view_material/47273`, 2020.

[24] R. Sharan, I. Ulitsky, and R. Shamir, "Network-based prediction of protein function," *Molecular systems biology*, vol. 3, no. 1, 2007.

[25] J. Yan, S. L. Risacher, L. Shen, and A. J. Saykin, "Network approaches to systems biology analysis of complex disease: integrative methods for multi-omics data," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1370–1381, 2018.

[26] A. Marin and B. Wellman, "Social network analysis: An introduction," *The SAGE handbook of social network analysis*, vol. 11, 2011.

[27] P. Cintia, S. Rinzivillo, and L. Pappalardo, "A network-based approach to evaluate the performance of football teams," in *Machine learning and data mining for sports analytics workshop, Porto, Portugal*, 2015.

[28] B. Gonçalves, D. Coutinho, S. Santos, C. Lago-Penas, S. Jiménez, and J. Sampaio, "Exploring team passing networks and player movement dynamics in youth association football," *PloS one*, vol. 12, no. 1, 2017.

[29] M. Newman, *Networks*. Oxford university press, 2018.

[30] B. Wang and Z. Luo, "Pagerank approach to ranking football teams' network," in *Proceedings of the 10th EAI International Conference on Simulation Tools and Techniques*, pp. 136–140, 2017.

[31] S. Abernethy, "Dynamic network 3 – 0 fifa rankings: Replacing an inaccurate, biased, and exploitable ranking system," 2018.

[32] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[33] T. Decroos, L. Bransen, J. Van Haaren, and J. Davis, "Actions speak louder than goals: Valuing player actions in soccer," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1851–1861, 2019.

[34] L. Bransen and J. V. Haaren, "Player chemistry: Striving for a perfectly balanced soccer team," 2020.

[35] W. Contributors, "Fantasy football (association)," Feb 2020.

[36] K. Cooper and M. Barahona, "Role-based similarity in directed networks," *arXiv preprint arXiv:1012.2726*, 2010.

[37] vaastav, "vaastav/fantasy-premier-league," Feb 2020. Available at `https://github.com/vaastav/Fantasy-Premier-League`.

[38] S. Brown, "A pagerank model for player performance assessment in basketball, soccer and hockey," *arXiv preprint arXiv:1704.00583*, 2017.

[39] E. Begley, "Paul pogba scores as man utd beat leicester 2-1 - jose mourinho praises captain," *BBC Sport*, Aug 2018. Available at https://www.bbc.com/sport/football/45053886.

[40] J. Pearce, "Confirmed - alisson becker is a liverpool player," Jul 2018. Available at https://www.liverpoolecho.co.uk/sport/football/transfer-news/liverpool-confirm-signing-alisson-becker-14930903.

[41] FourFourTwo, "Ranked! the 50 best players in the premier league this season," Apr 2019. Available at https://www.fourfourtwo.com/features/best-players-premier-league-201819.

[42] T. Cudworth, "Premier league goalkeepers: Ranking the 20 current first choice stoppers," 2019. Available at https://www.90min.com/posts/6267824-premier-league-goalkeepers-ranking-the-20-current-first-choice-stoppers.

[43] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[44] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.