Problem Chosen	2020 MCM/ICM Summary Sheet	Team Control Number
D		2000708

Best team, best team performance

Summary

Researches on team success are always eye-catching and heated-discussed. In this paper, we state our points on team strategies and success through several models.

To solve the first problem, we establish a multivariate weight evaluation complex network model (MWEN) by setting up a weight matrix through the distance between players, the number of passes, and positions. Combining the match time and players' position, MWEN model is visualized, which can directly reflect the relationship between different players on the field. Then, we calculate the degree centrality, betweenness centrality, closeness, clustering coefficient and other indicators. These parameters describe the relationship between a pair as well as the team from different perspectives in multiple scales. We also use label propagation community detection algorithm to identify the configurations of the network.

In the second problem, we define the player conversion rate, possession time, and action contribution, together with betweenness as four indicators to evaluate players' performance. These indicators can describe the players' flexibility, the ability to control the ball, the cooperation with the players and so on. Using AHP to calculate the weight of new indicators, we get the comprehensive performance which can fairly reflect the contribution level of players. According to it, we get the rankings of Huskies players and different teams respectively. In addition, using the empirical distribution function of two situations under three outcomes, we draw the conclusion that the opponent's counter-strategy has greater influence on the outcome than the home team strategy.

For the third problem, we consider the goalkeeper, defender, midfielder and forward as the object of formulating structural strategy. Through cross analysis of various indicators of different positions, we get customized suggestions for each position. Through the sensitivity analysis of our model we find that "betweenness" is comparatively sensitive to the score, so we can get several structural strategies to improve the team's comprehensive performance.

To solve the last problem, we extend MWEN model to general team network by analogy analysis and case study. We analyze the subjective and objective factors of team performance through a case of virtual social network. Furthermore, we explain the quantitative methods of indicators and the construction methods of similar network models.

Keywords: Team performance; Group dynamics; Performance evaluation; Community detection

Contents

1 Introduction

1.1 Problem background

With the increasingly complex structure and problems of human society, the team composed of elites in various fields is more sought after by people than the big name alone. Also, the importance of the team is valued by an increasing number of people. Therefore, how the team use the best strategy to solve the problem quickly becomes a new exploration direction for some scholars. One of the most informative settings to explore team processes is in competitive team sports. In this paper, we have the Huskies to be our home soccer team with their all 38 games against their 19 opponents.

- To understand the team's dynamics, we are asked to create network for the passes.
- Then, player's performance plays a vital role in exploring the reasons for team success. Our home soccer team and the opponents should all be taken into consideration.
- For practical significance, we will work out the strategies for the Huskies.
- For generalized use, not just in a controlled setting team, we should consider group dynamics and performance in various fields.

1.2 Our work

To further present our solutions, we arrange our paper as follow:

- In section 2 and 3, we give out the reliable assumptions to simplify the model, then the symbols listed are used to describe our model.
- In section 4, to solve problem A, we take each player as a node, and build a complex multivariate network model to identify network patterns and properties.
- In section 5, to solve problem B, after identify the player's and team's performance indicators, we use analytic hierarchy process to quantify the weight of indicators. Then, we do regression analysis of the Huskies' and their opponents' performance which is the comprehensive performance that can fairly reflect players' contribution, thus drawing the conclusion of team strategies.
- In section 6, to solve problem C, we make use of Stacked Bar Chart to compare various indicators of different positions, thus get customized suggestions for each position.What's more, we do sensitivity analysis on each indicator, and then get several structrual strategies to improve team performance.
- In section 7, to solve problem D, we generalize our model to solve team issues in various fields.
- At last, we discuss the strenghths and weaknesses of our model in detail.

2 Assumptions

To simplify our problems, we make the following basic assumptions:

- Accurate and true data provied by "2020 Problem D DATA".
- If the outgoer and receiver of two adjacent passes are the same player, the time difference between the two passes is approximately the ball possession time of the player (ignoring the ball rolling time).
- When a player passes, the player who receives the ball is in the destination in the data we use.
- If a shot is followed by "Reflexes" of the other side's goalkeeper, it means that the player on his own side has scored and the other side's goalkeeper has lost the ball. If not, it will be regarded as a goal failure.
- The ID of regular players and substitute players in the same position can not be identified by playing time, and the distinction between them should be inferred in the figures.
- All fields are the same size and shape.

3 Nomenclature

In this paper we use the nomenclature in Table1 to describe our model. Other symbols that are used only once will be described later.

Symbol	Significance
W_{ij}	Weight between nodes in adjacency matrix
d_{ij}	Distance between nodes in adjacency matrix
$C_D^{w\alpha}(i)$	Improved weighted Degree of player i
$C_{closeness}(i)$	Measurement of the difficulty of reaching code i
$C_{betweenness}(i)$	Measurement of the indispensability of code i to a network
$C_{total}(G)$	Measurement of the clustering of the whole network
$C_{performance}$	Comprehensive performance score of player i
$C_{goal}(i)$	Direct contribution score of player i
$C_{con}(i)$	Conversion rate score of player i
$C_{possessionTime}(i)$	Total possession time score of player i
$C_{action}(i)$	Individual action contribution score of player i

Table 1: Nomenclature

4 Solution to Problem A

In this section, we discuss all the details about our model used for problem A, followed by the solutions and problem-solving process.

4.1 Data pre-processing

We use the "passingevent.csv" file provided by the topic to build a two-dimensional coordinate system.

We know from the data provided by the topic:

- There are 4 kinds of players: 'F' forward, 'D' defender, 'M' midfielder, 'G' goalkeeper;
- A unqiue identifier for each match played during the season, and reflects the order of the match in the season;
- MatchPeriod
  The half in which the event took place. '1H': first half, '2H': second half;
- EventTime
  The time in seconds during the MatchPeriod (1st or 2nd half) at which the event took place.

By abstracting the player as a node, the position of the player, i.e., the coordinate, is determined. We assume that there are players i and j of the Huskies:

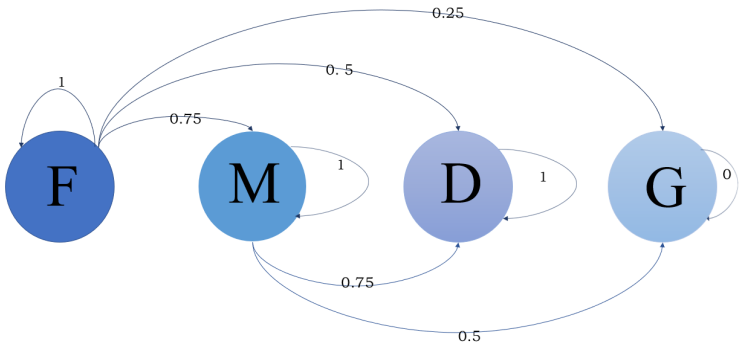
- We calculate the distance of each Huskies' player at different time and normalize it by the position of the players' coordinates.
- The number of passes from player i to player j in the table is calculated and normalized.
- Distinguish the passing categories of different players.

4.2 MWEN model

We construct the MWEN model according to the weighted (adjacency) matrix:

According to Javier et al. (2012), this basic visual analysis can be more quantified by calculating global network invariants (describing the overall characteristics of a team) or local invariants (providing insight into individual participants). They all depend on the adjacency matrix W, where W_{ij} in the matrix W has a weight from player i to player j.
It's not hard to know that the further away players are from each other on the pitch, the less connected they are. Similarly, the number of passes between players reflects the closeness of their relationship.

For the type of passes, consider the connection between different players' positions. For example, the same position usually passes more passes, while the connection between different positions is usually weakened according to the farther position they usually have in the court. In addition, the model visualization we implemented finally forms a directed graph, and the possibility of player i passing to player j is different from that of player j passing to player i.



Through the above rules, the following standards are established and nomalized.

Let the (normalized) distance be L_{ij}, the number of passes be n_{ij}, quantitative passes category index be f_{ij}, where $i, j \in \{ F, M, D, G\}$

$$f_{ji} = \begin{cases} 1 - f_{ij} & f_{ij} > 0.5 \\ f_{ij} & f_{ij} = 0.5 \\ \frac{1}{2} \times f_{ij} & f_{ij} < 0.5 \end{cases} \tag{1}$$

(2)

$$W_{ij} = \alpha \cdot \frac{1}{L_{ij}} + \beta \cdot n_{ij} + \gamma \cdot f_{ij} \tag{3}$$

According to the importance, we give 0.2, 0.4 and 0.4 weights to α, β and γ respectively. Finally, we get the weight adjacency matrix W of i to j from the above formula:

$$W = \begin{bmatrix} ('F',' F') & ('F',' M') & ('F',' D') & ('F',' G') \\ ('D',' D') & ('D',' M') & ('D',' G') & ('D',' F') \\ ('M',' F') & ('M',' M') & ('M',' D') & ('M',' G') \\ ('G',' F') & ('G',' M') & ('G',' D') & ('G',' G') \end{bmatrix} = \begin{bmatrix} 1 & 0.75 & 0.5 & 0.25 \\ 1 & 0.75 & 0.5 & 0.5 \\ 0.25 & 1 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.5 & 0 \end{bmatrix} \tag{4}$$

4.3 Dynamic model visualization

Weight will be used to measure the length and thickness of different nodes in the network, and to measure the proximity between two players.

We have mapped the visual tracks of players in the first half and the second half of all data in passingevent.csv. Space and dynamics are shown on the graph by intercepting the average positions of players in different time periods. Because it is not sure whether the player is a substitute in a match, whether he or she will play midway or not, and we use all- matches data for having more accurate results, according to the degree of correlation between the midpoint and the point in the figure, that is, the closer the same type of players get together, the more likely they are substitutes for each other, and the shape of the match can also be seen.

Although we have the data in the range [0, 100], we reshape the following figures to the actual size of a soccer field.

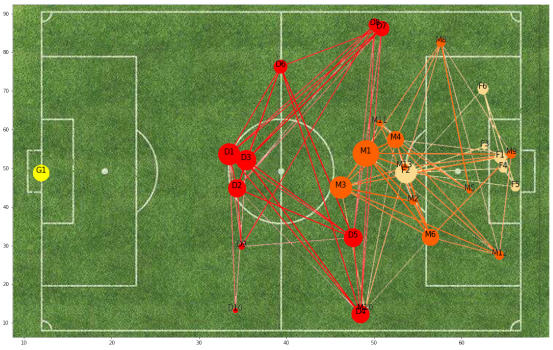


Figure 1: Passing networks for the Huskies in the first half in proportion to the actual size of the soccer field, using the passingEvents data

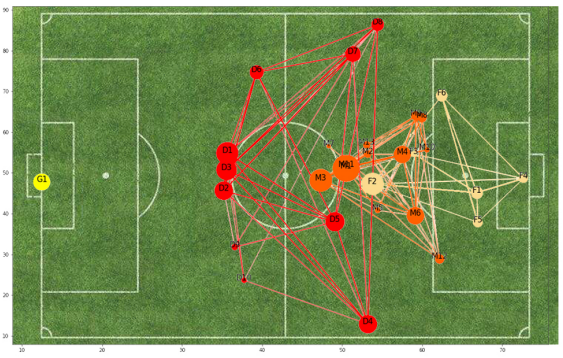


Figure 2: Passing networks for the Huskies in the second half in proportion to the actual size of the soccer field, using the passingEvents data

The position of the node represents the average location of the player on the field, the size of the point represents the touch times of the player, and the color of the point represents the position of the player who passes the ball: yellow is the goalkeeper G, red is the forward F, orange is the midfielder M, and light yellow is the defender D. The point label represents the player's name. The thickness of the line represents the weight, and the color of the line represents the position of the player passing the ball.

4.4 Identify network patterns and configurations

- Network indicator

Through several local network invariants of the ball-passing network, we measure the individual contribution of a player in a team.

- Improved weighted Degree

Improved weighted Degree is a measure of the importance of players. Improved weighted Degree is the weight of quantity and branch chain weight, using Degree Centrality Measure proposed by Tore Opsahl et al.(2010) for reference:

$$C_{D-out}^{w\alpha} = k_i^{out} \times \left(\frac{s_i^{out}}{k_i^{out}}\right)^{\alpha} \tag{5}$$

$$C_{D-in}^{w\alpha} = k_i^{in} \times \left(\frac{s_i^{in}}{k_i^{in}}\right)^{\alpha} \tag{6}$$

$$C_{D}^{w\alpha} = \frac{C_{D-in}^{w\alpha} + C_{D-out}^{w\alpha}}{2} \tag{7}$$

where α is a positive tuning parameter that can set according to the research setting and data.We setα0.8 here referring to literature. If this parameter is between 0 and 1,

then having a high degree is taken as favorable, whereas if it is set above 1, a low degree is favorable. k^{out} represented the activity of a node, that is, the number of ties that originate from a node. k^{in} shows the number of ties that are directed towards a node, a proxy of its popularity. s^{out} and s^{in} are defined as the total weight attached to the outgoing and incoming ties, respectively. C_D is the degree of one node.

- Closeness

Direct measurement of the difficulty of reaching a specific player in a team(using Python package):

$$C_{closeness}(i) = \frac{1}{\sum_j d_{ij}} \tag{8}$$

where d_{ij} is defined as the distance between two nodes:

$$d_{ij} = \frac{1}{w_{ij}} \tag{9}$$

where w_{ij} is the weight between node i and j.

- Betweenness

$$C_{betweenness}(i) = \sum_{j \neq k \neq i} \frac{n_{jk}}{g_{jk}} \tag{10}$$

where n_{jk} is the number of nodes on the shortest path through node i, and g_{jk} is the number of nodes on the path between j and k through node i.

- Global clustering coefficient

In graph theory, clustering coefficient is a measure of the degree to which points in a graph tend to cluster together. The evidence shows that in most real networks and special social networks, nodes have a strong tendency to form a group, which is characterized by a relatively close connection.

The global clustering coefficient measures the clustering of the whole network, based on the triple of nodes. A triple is one of three nodes with two (open triples) or three (closed triples) undirected edge connections. A triangle consists of three closed triples, which are concentrated on each node. The global aggregation coefficient is the number of closed triples in all triples (both open and closed).

$$C_{total}(G) = \frac{3 \times G_\Delta}{3 \times G_\Delta + G_\wedge} \tag{11}$$

where G_Δ represents the number of closed triples in the path; $G_\wedge$ represents the number of open triples in the path.

- Local clustering coefficient

$$C_i = \frac{2|\{e_{jk}\}|}{u_i(u_i-1)} \tag{12}$$

where $|\{e_{jk}\}|$represents the **the number of connections** between its adjacent nodes, and $\frac{u_i(u_i-1)}{2}$represents **the number of all possible connections** between adjacent nodes.

The clustering coefficient of the whole network $\overline{C} = \sum_i C_i$ can also be defined by Watts and Strogatz as the mean value of the local clustering coefficient of all nodes n.

4.4.1 Network configuration

Network dyadic and triadic configurations is a popular community detection problem. Here, Through "asyn lpa communities" algorithms in Python (ASYN), we repeatedly set the label of a node to be the label that appears most frequently among that nodes neighbors until each node has the label that appears most frequently among its neighbors. Results are as follows:

Community Detection Results of Huskies Team		
M12	D2,D4,D5,D6,D7,D8, F1,F2,F3,F4,F5,F6, M1,M10,M11,M13,M2,M3,M4, M5,M6,M7,M8,M9	G1 D1,D10, D3,D9

Table 2: Community Detection Results of Huskies Team

Because of the randomness of the community detection algorithm results, it can be seen from the representative community results (Table 2) that four defenders and a goalkeeper are a community, which has set up the huskies team's back line and become a more reasonable community, while the forward and the midfield do not constitute a dyadic or triadic configuration, which shows that the whole team cooperates tacitly. So the network is team configuration.

It is worth mentioning that we found that the results of the community based on the self-defined weight matrix are not good. The results shown above are only based on the number of passes as the weight index. Although it looks unitary, from another perspective, it shows that a single index considering only the number of passes is conducive to the simplest and direct reflection of the community structure. But, this does not mean that our weight algorithm is redundant. Compared with ASYN algorithm only considering the number of communities, in fact, MWEN model takes more indicators into consideration, which can comprehensively reflect the performance of team members and is relatively more stable. Therefore, we will continue to use MWEN model in following parts.

4.5 Structural indicators and network properties

By calculating the Improved weighted Degree, Closeness, Betweenness, Global clustering coefficient and Local clustering coefficient of different types of players, we

can get the values of the pair's indicators and the four types of indicators of the whole team in one game and the whole season, and figures.

Then, we consider the differences of Degree, Closeness, Betweenness, and Clustering between two players. Using 350s of a game as a time interval to include enough passes, we select the two players with the highest betweenness value in an intense game, and continuously observe the changes of betweenness value of the two players in different time intervals. Because there are not enough data, the index is almost zero in different periods of time. Therefore, in a single game, we set the index value as $index_i$ for the pair ,and set $index_{total} = \frac{\sum_i index_i}{n}$ for the whole team, which is the average value of all nodes.

We multiply different indicators by coefficients, so that they fall roughly in the same two-level range, which is convenient for observation and analysis at the same time.

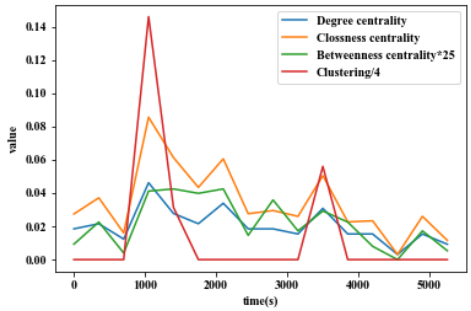


Figure 3: Two players in one game

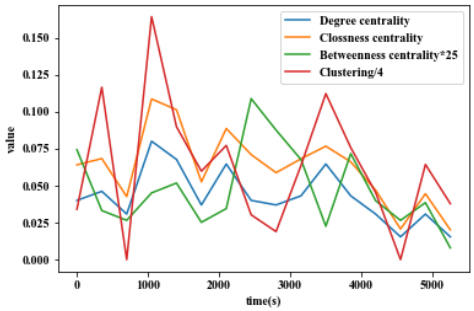


Figure 4: Whole team in one game

As shown in Figure 3, for 2 players, Clustering fluctuates greatly with time in one game and has lots of 0 values, while Betweenness, Closeness and Degree tend to be more stable in a certain range, where Closeness has the largest average value, followed by Degree and Betweenness, showing that this pair are close to each other in the game. What's more, players are more energetic in the first half of the game because the value of four indicators goes down over time. In Figure 4, for the whole team, Betweenness and Clustering fluctuate greatly, which indicates players often change their formation. Compared with them, the other two indicators have similar vibration amplitudes and respective average values, showing the team's close connection.

For the whole season, we use the same method to segment each game, and draw the changes of different indicators of the fixed two nodes in different games, that is, the changes of different indicators of the whole team with the game number. The figures are as follows.

As shown in Figure 5, for 2 players, Clustering fluctuates frequently and greatly over the season, while Betweenness, Closeness and Degree tend to be more stable in a certain range, where Closeness has the largest average value, followed by Degree, then by Betweenness, which illustrates that players tend to act similarly in each game, except their formation. At the same time, in Figure 6, for the whole team, Betweenness fluctuates greatly. Compared with it, the other three have similar vibration amplitudes and respective average values,showing the similar formation.

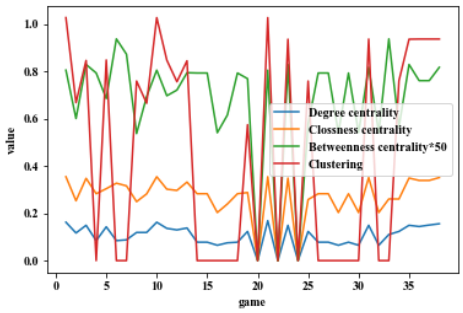


Figure 5: Two players in one season

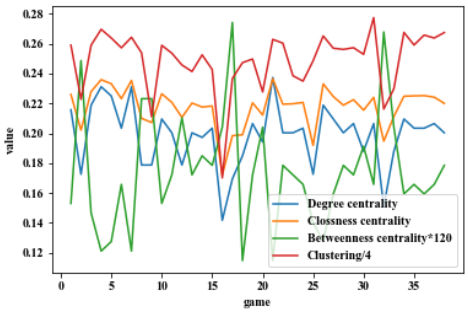


Figure 6: Whole team in one season

5 Solution to Problem B

The performance of players has a significant influence on many aspects, such as team strategy, player rankings, and player compensation. How to describe players' performance quantitatively has become more and more vital for coaches. A player's performance is related to many factors, such as, ratio of goals per shot, possession in genral, possession characteristics, smart passes and so on.

5.1 Data pre-processing

The "fullevents.csv" data of players with opponents adds information such as soccer moves and shots. Using these added information, we can not only identify the football movement,thus grading each player's performance, but also identify the substitute's playing time. Moreover, the time when the team scored the goal and other information can be clarified as well. It is conducive to our better analysis of player's performance with these information.

5.2 Quantify performance index

$$C_{performance} = C_{direct} + C_{indirect} \tag{13}$$

Because player's personal performance is not only related to whether he scores goals thus leading to the direct result of the game, but also closely related to whether he reasonably cooperates with his teammates, responds to cover and helps teammates score. Therefore, we divide player's contribution into direct and indirect score contribution.

5.2.1 Direct score contribution

Designate direct score contribution as C_{goal}.

$$C_{goal} = f(R) = e^{a_0 \cdot R + b_0} \tag{14}$$

Goal proportion is:

$$R = \frac{n_{goal}}{n_{shot}}, \tag{15}$$

where n_{shot} is the total number of shots a player has in a game. Goal scoring can be measured by the number of shot actions of the player, minus the number of defensive failures of the other team, i.e., the number of reflexes of the other goalkeeper. It should be noted that only forwards and centers may have direct score contribution.

5.2.2 Indirect score contribution

Indirect score contribution can be defined as follows:

$$C_{indirect} = \alpha 2\beta 2 \cdot C_{con} + \alpha 3\beta 3 \cdot C_{possessionTime} + \alpha 4\beta 4 \cdot C_{Betweenness} + \alpha 5\beta 5 \cdot C_{action} \tag{16}$$

where α_i is weight, β_i is dimensional correction factor.

We use AHP to determine the above parameters. Consistent judgement matrix Denote $\{x_1, x_2, x_3, x_4\}\, as\, \{C_{betweenness}, C_{action}, C_{possesion}, C_{possessionTime}\}$.From the matrix proposed by Saaty, we get the following pairwise comparison matrix M :

$$M = \begin{bmatrix} 1 & 5 & 3 & 7 \\ 1/5 & 1 & 3/5 & 3/7 \\ 1/3 & 5/3 & 1 & 7/5 \\ 1/7 & 7/3 & 7 & 1 \end{bmatrix} \tag{17}$$

After calculation, we get CI= 0.04502561010707584, CR= 0.04020143759560343<0.1

Compared with matrix W , we pass the consistence test and get the weight vector $(\alpha_2, \alpha_3, \alpha_4, \alpha_5) = (0.6052347, 0.08969842, 0.17162574, 0.13344114)$

- Player conversion rate score (Possesion charge score)

The calculation of player conversion rate score is that each time the Huskies get the ball from their team member, pass it to the next player on their own side and get one point, until it reaches the player in the other team, then two points will be reduced. At this time, a conversion rate scoring period ended, and the next conversion rate scoring period starts from the next time the Huskies get the ball.Tthe total conversion rate of players is the cumulative score of all scoring periods.

$$C_{con} = \sum_{i}^{n} C_{con}^{i} \tag{18}$$

- Total possession time score

The calculation method is from the time when the Huskies get the ball as the destiny player to the time when the Huskies pass it to their opponents. This time period is the

scoring period of one ball control time. Then the total possession time is the sum of the one ball control time.

$$C_{possessionTime} = \sum_{i}^{n} C^{i}_{possessionTime} \tag{19}$$

- Betweenness score

As mentioned in the previous chapter, being an important parameter from the network model, betweenness is defined as the central media ability of the player and other players, which represents the indispensable degree of the player in the whole team i.e., if the player is substituted, the team's passing chain will be disconnected a lot or completely. Therefore, this indicator can also be used as a standard to measure indirect contribution.

- Individual (event)action contribution score

The action of players also has a great impact on the game. A player's good action, such as a smart pass and a clearance, can help the team to control the ball and win a score, while a bad action, such as a foul, may lead to his being sent off, thus making the team lose its original stable formation. Therefore, according to the knowledge of football, we can define the individual action contribution score of players through their actions. The formula is as follows:

$$\begin{aligned} C_{action} = {} & 4 \cdot Groundattackingduel + 4 \cdot Groundde fendingduel + 1 \cdot Groundlooseballduel \\ & +3 \cdot airduel + 1 \cdot launch + 3 \cdot clearance + 5 \cdot smartpass + 2 \cdot cross - 2 \cdot kickcross \\ & -7 \cdot fuel + 3 \cdot Acceleration + 1 \cdot Goalkeeperleavingline - 10 \cdot Outofgamefoul \\ & +3 \cdot Saveattempt - 8 \cdot otherfuel - 5 \cdot Reflexes \end{aligned} \tag{20}$$

where ground attacking duel, ground defending duel, ground loose ball duel, air duel, launch, clearance, smartpass, cross, acceleration, goalkeeper leaving line, save attempt are regarded as the action that contributes to the team's score, and its contribution rate weight is given according to relevant information and empirical evidence. However, fuel, kick cross, out of game foul are regarded as fouls of different degrees, which can reduce players' scores. For "the other fuel" given in the table, we always give the weight of "-8" for the sake of simplification. In addition, reflexes refers to defensive failure, which represents the negative contribution rate of the defensive players.

In addition, it should be noted that for different positions, there will be their own regular actions. For example, in the indicator calculation, the midfield usually takes the task of shot, while the goalkeeper rarely passes and takes more defensive actions. Therefore, for each player, when we consider all the action indicators, we can evaluate their performance comprehensively and fairly.

Finally, the comprehensive indicator of players' performance $C_{performance}$ is obtained by summing up each indicator.

$$C_{performance} = C_{direct} + C_{indirect} \\ = \alpha 1\beta 1 \cdot C_{goal} + \alpha 2\beta 2 \cdot C_{trans} + \alpha 3\beta 3 \cdot C_{controltime} + \alpha 4\beta 4 \cdot C_{Betweenness} \\ +\alpha 5\beta 5 \cdot C_{action} \tag{21}$$

After establishing all performance indicators above, we have the Huskies rankings and all teams' rankings, shown the performance of each team and every single player.

Player ID	Rank	Performance score	Player ID	Rank	Performance score
M1	1	81.786	G1	16	36.399
D5	2	75.771	M9	17	36.094
D3	3	60.006	M4	18	33.639
M3	4	58.832	F5	19	33.162
D7	5	58.670	F6	20	31.227
D8	6	58.031	F4	21	23.868
D1	7	57.162	M11	22	21.145
D6	8	54.617	M12	23	16.890
D4	9	54.226	M8	24	15.284
F2	10	53.122	M2	25	14.876
M13	11	44.851	M5	26	12.273
D9	12	44.009	F3	27	12.006
F1	13	38.573	M7	28	11.931
D2	14	38.339	M10	29	10.761
M6	15	37.952	D10	30	9.2145

Figure 7: The Huskies rankings

Team ID	Rank	Performance score	Team ID	Rank	Performance score
Opponent2	1	99.847	Opponent10	11	46.750
Opponent5	2	86.058	Opponent6	12	44.623
Opponent4	3	77.151	Opponent12	13	43.548
Opponent16	4	76.626	Opponent17	14	43.451
Opponent9	5	65.756	Opponent15	15	39.106
Opponent3	6	65.280	Opponent8	16	38.683
Opponent7	7	54.239	Opponent19	17	38.470
Huskies	8	50.318	Opponent1	18	35.196
Opponent14	9	47.699	Opponent18	19	34.075
Opponent11	10	46.872	Opponent13	20	31.607

Figure 8: Rankings of all teams

To analyze the influence of the performance indicators we proposed, on one hand, we pick out two players from the Huskies and their opponents, respectively. Then we compare the performance indicators of them and display them by radar chart.

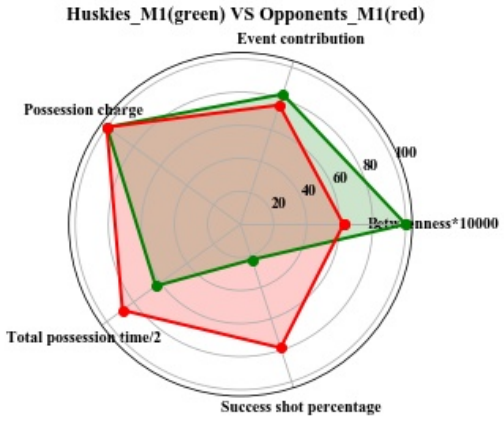


Figure 9: Huskies M1 VS Opponents M1

As is depicted in the chart, both player have a high conversion rate score i.e., possesion charge score, showing their close contact with their team members and being a great midfield. It is obvious that Huskies M1 is an integral part of the team, while Opponents M1 has weaker influence. However, Opponents M1's success shot percentage is much higher than Huskies M1, and total possession time is higher than Huskies M1, which means Opponents M1 is a briliant contributor to team scoring. Event contribution has rarely different effect on the two players' success.

On the other hand, the top 11 players who have the best scores under four indicators: Betweenness, Event contribution, Total possesion time and conversion rate were taken into consideration to analyze the Huskies and the change of the average index of

all opponents, i.e., the overall level of the specific indicators of European team members. For each indicator, we select the top 11 players in the team, that is to say, the ID of the players under each indicator is not necessarily the same. However, due to the ranking of indicators from high to low, this image shows the distribution level of different indicators in players in an ingenious way.

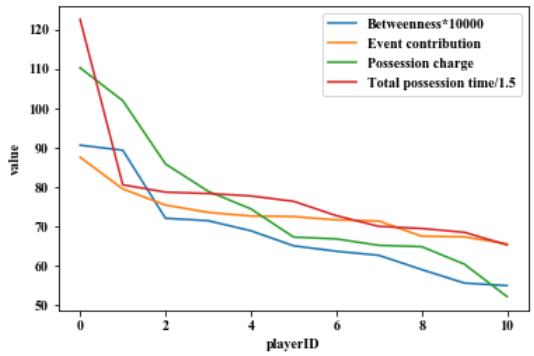


Figure 10: Top 11 players of The Huskies

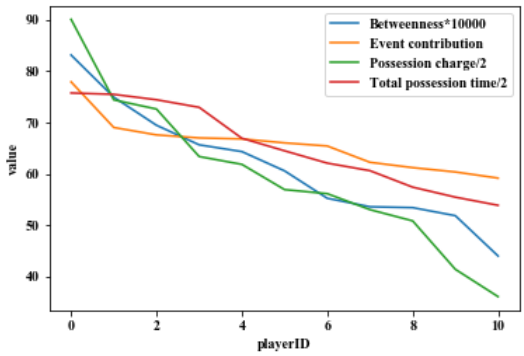


Figure 11: Top 11 players of the Opponents

As shown in Figure 8, the distribution of indicators shows a smooth exponential distribution, that is, most players' indicators are relatively concentrated, while the best few players' indicators are far ahead. The higher the ranking is, the greater the difference of the index value of the adjacent ranking is. Meanwhile, we can draw a conclusion from Figure 9 that the vast majority of the opponents' indicators are at average level, and few players have relatively high indicators.

In addition, we compare the Huskies and one of the opponents' performance.

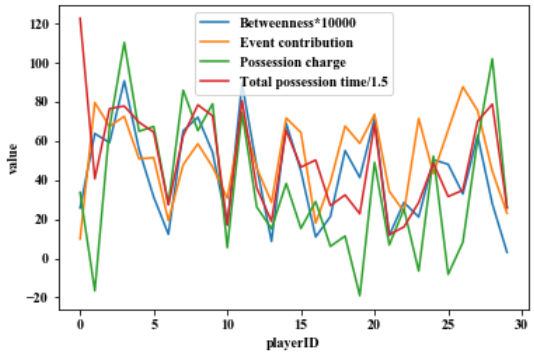


Figure 12: 30 players of The Huskies

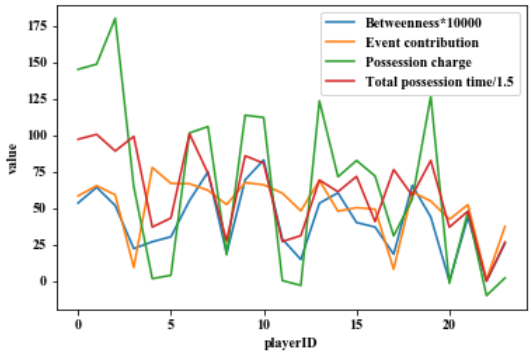


Figure 13: 24 players of the Opponents

In Figure 10, we can have a thorough view of the Huskies performance. It is shown that some of the players have high balanced indicators, such as, ID 12 and 20, while ID 5, 10 and 13 have relatively weak performance. In addition, most of the Huskies have their own strengths and weaknesses, so the team should develop in a balanced way. We can evaluate the opponents' performance through Figure 11, among which the ID 8 and 21 are relatively balanced but the overall value is low, most of the other players have different indicator values.

5.2.3 Strategy universality

A good decision can make the team score more goals in all competitions, but it can also depend on the capability of the enemy team. Meanwhile, we assume that a good strategy is to improve the team's own comprehensive strength, that is, a good strategy represents a higher team performance score.

Here, we have obtained the comprehensive performance scores of each team in each game. According to this, we consider the impact of our team's performance score on the number of goals of our team. In addition, we consider the impact of the difference between our team's performance score and the other team's performance score on goals, i.e., we also consider the other party's strength.

$$C^{diff}_{performance} = C^{Huskies}_{performance} - C^{opponent}_{performance}$$

We assume that when the team wins the game, the performance score of the team is generally higher, but when the team loses the game, the score is generally lower(Jordi Duch, 2010). Performance scores may present different distribution levels under different competition results. Therefore, we consider drawing the performance of the host team and the performance difference between the host team and other teams according to the results of the three kinds of competitions: loss, win and tie.

The experience distribution function of $C_{performance}$, $C^{diff}_{performance}$ under different competition results is shown in the figure. We find out that the shape of empirical distribution function is similar and separated from each other, and it has a very normal geometry.

We use the exponential function to adjust the parameter of $y = \frac{1}{(1+exp^{-c*(x-d)})}$,

and make regression analysis. The result shows that the regression is very good.

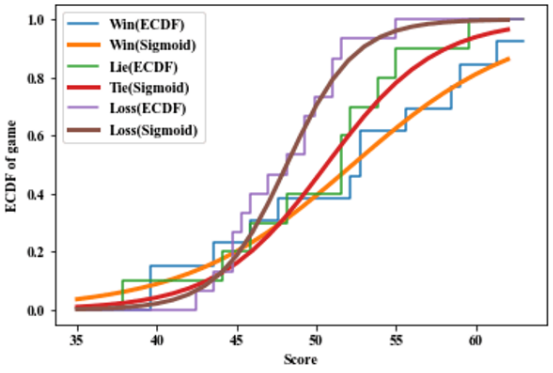


Figure 14: Empirical distribution function and fitting line of $C_{performance}$

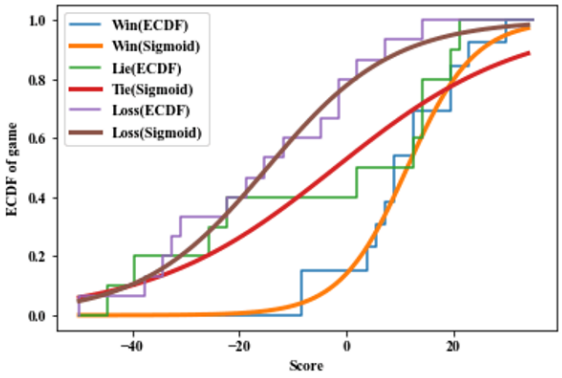


Figure 15: Empirical distribution function and fitting line of $C^{diff}_{performance}$

For the figure of $C_{performance}$, we look at the value of 50, and it can be seen that the probability of performance score on the right side of the value is 0.6, 0.52 and 0.3, respectively. Hence, there is a positive correlation between performance score and winning situation.

For the figure $C^{diff}_{performance}$, the distribution difference of the three curves is greater than above. Take point 0 as an example, the probability of $C^{diff}_{performance}$ on the right side of the value is 0.8, 0.5 and 0.2, respectively, which shows that the distribution difference of $C^{diff}_{performance}$ is more significant under different winning and losing levels. In other words, the value of $C^{diff}_{performance}$ is more significant for the outcome.

By comparison, we can see that the team's winning and losing results are positively correlated with the team's $C_{performance}$ and $C^{diff}_{performance}$, but the latter has a stronger impact on the outcome. Therefore, we come to the conclusion that the strategy has certain effect, but it depends on the opponent's counter-strategy more.

6 Solution to Problem C

6.1 Cross Analysis

In fact, it is difficult to study and formulate a team's structural strategy through the network model with low risk of uncertainty because the ball passing network of a team in a football match changes with time and space, and when one makes his own actions, others will make corresponding responses at the same time, i.e., the network has many uncertainty. Hence, we analyze the specific positions one by one, consider their improvement, and give the optimal strategy for each position in the network.

There are forward, midfielder, defender and goalkeeper in the team. They play different main roles, including goalkeeper opposing other team from scoring, defender tackling, midfielder assisting attack and shooting, forward mainly shooting, which forms a stable structure in the match field.

Based on the data from the previous calculation, we draw a stacked histogram of the players' indicators. Among them, we put players of the same profession adjacent to each other in order to learn the strengths and weaknesses of each position from the figure.

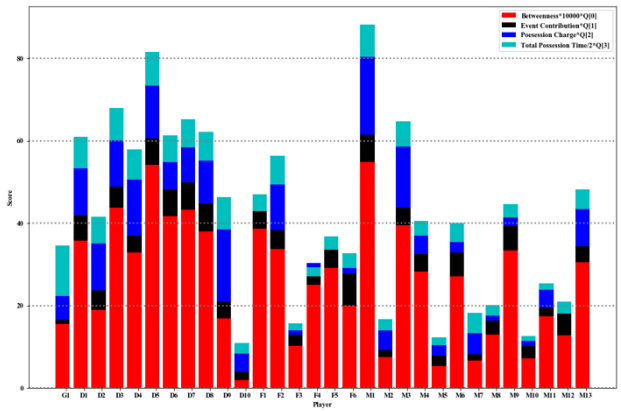


Figure 16: Four indicators' score of all players

The positions are goalkeeper, defense, forward, midfield from left to right.

- Goalkeeper

  As is depicted from the figure, goalkeeper's betweenness and possession charge scores are at average level, but has low event contribution score, which means he should improve the skill to prevent the opposing team from scoring and perform smart goal kicks. Compared with the average level of the whole team, the goalkeeper has more time to hold the ball, which shows that the goalkeeper's ability to control the ball is great.

- Defense

  The defenders' indicators are at a high level. More betweenness and possession time show that the defenders of the team have a strong ability to stabilize the rear, to control the field, and to tackle. A close relationship between the defenders is conducive to driving the whole team's attack and defense. The high conversion rate score indicates that the guard's passing is more flexible, and a good contribution rate of movement also shows that the rear tailgate is their good personal skill. All in all, almost all defenders' indicators have reached a quite satisfactory standard, except D10 defender who has low scores in all aspects.

- Forward

  The overall ability of forwards is not as good as that of midfielders, and their betweenness is at the average level. For possession charge, there are great differences between different players, which shows that the flexibility of the forward team is uneven, so it is necessary to enhance the passing training among the players.

- Midfield

  The midfielder accounts for nearly half of the team, however, their betweenness scores differ at relatively low level. Player M1, M3, M4, M6, M9, M13, have a quite great ability of passing ball, while others are far below average. It is speculated that these should be substitute players, which may be that they have not been well trained. Therefore, it is necessary to appropriately increase the opportunities for substitute players to be trained. Anyhow, the overall skill of midfielders still needs to be strengthened. We can consider learning relevant experience with the defenders to enhance the defense ability

Through the stacked histogram, we clearly show the indicators of different team members and even the indicators of the team. Through analysis above, we get the improvement strategies for each position, so as to effectively improve the overall performance level of the team.

6.2 Sensitivity Analysis

	Betweenness	Event contribution	Possession charge	Total possession time	Changing proportion	Average score	Deviation of score
0	0.60523	0.08970	0.17163	0.13344	0%	42.0180	0.00%
1	0.69602	0.08970	0.17163	0.13344	15%	45.9827	9.31%
2	0.51445	0.08970	0.17163	0.13344	-15%	38.1073	-9.31%
3	0.60523	0.10315	0.17163	0.13344	15%	42.7107	1.65%
4	0.60523	0.07624	0.17163	0.13344	-15%	41.3253	-1.65%
5	0.60523	0.08970	0.19737	0.13344	15%	42.9642	2.25%
6	0.60523	0.08970	0.14588	0.13344	-15%	41.0717	-2.25%
7	0.60523	0.08970	0.17163	0.15346	15%	42.7710	1.79%
8	0.60523	0.08970	0.17163	0.11342	-15%	41.2649	-1.79%

As for the advice on the coach, we consider our several performance scores the main factor contributing to team success, and access their impact on team success by sensitivity analysis. The table above showed our results. We set each indicator a 15% deviation and -15%deviation in turn and observe the final performance score's alteration. It's easy to know that 15% change in betweenness results in 9% change in total event contribution, while the same changing proportion of event contribution, possession charge, total possession time cause 1.6%,2.3%,1.8% change in final performance score respectively.

Therefore, we identify the betweenness has largest impact on team success. Next, we put forward several suggestions to improve players' betweenness. First, increase players training time properly and target at efficient coordination in passing patterns. Second, put more importance on players' assistance to each other to strengthen the bound. Thirdly, forcing players to interact with who have good possession of ball. Last but not least, promote everyone to participant in real match.

7 Solution to Problem D

After analyzing the group dynamics in a controlled setting of a team sports, generalizing the findings is also worth being emphasized. As indispensable as social network is, we explore the indicators that social network team performance has on the basis of comprehensive consideration. Through the analysis, we think there are the following points to consider. First of all, we divide the impact into subjective factors and objective factors. Subjective factors often represent more complex interpersonal relationships between people, while objective factors affect the performance level of the whole team from the physical function of people and the overall scale of the team.

7.1 Objective factors

1. Education level

   The education level of the team members determines the working efficiency of the team, to a great extent. Especially in the field of high technology, the academic certificate of team members shows their knowledge range and comprehensive quality. In addition, professional level and additional skills are also a bonus for team work.

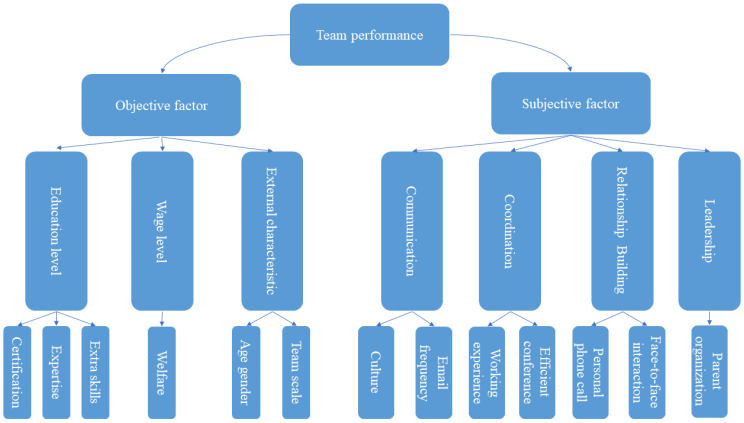


Figure 17: Team performance factors

2. Wage level

   Salary level, including the welfare and vacation of players, can be regarded as an incentive measure for employees. Well paid teams usually have high enthusiasm and subjective initiative for work, which is consistent with the evaluation indicators of our above-mentioned network model of ball passing.

3. External characteristic

   The external objective characteristics of the team, such as age and gender distribution of the team, usually play a crucial role in the team success. Young team members are often the fresh blood in the team, while the old are well-experienced. The appropriate gender ratio is powerful for the harmony and stability of the team.

7.2 Subjective factors

1. Communication

   Team members' communication ability is an important factor to maximize internal information. The number of communication between team members reflects the work intensity and cooperation density of the team, which is conducive to the overall performance improvement of the team.

2. Coordination

   The difference between cooperation and communication between team members is that the latter pays more attention to the idea exchange efficiency of specific tasks, while the former only summarizes the frequency of contact between players in general. The good cooperation of team members not only improves the tacit understanding and efficiency of the team, but also creates a comfortable working atmosphere.

3. Relationship building

   Intimacy reflects the cohesion of a team member and the consolidation of the relationship. The lack of trust between individuals in a team may lead to fierce

infighting, which will greatly reduce the team's performance level. Face to face interaction, personal phone, etc. can reflect the intimacy between team members.

4. Leadership

   The management personnel in the team can make an indispensable contribution to the overall planning of the whole team. The management ability of the leader can unify the team objectives, establish a hierarchical team structure, and thus improve the efficiency of the team. In addition, leadership is also conducive to improving the fairness of the team and solving the disputes among the team members.

7.3 Case analysis

Chong King Tan et al.(2018) had a quite thorough research of the social network. They considered several indicators influencing team performance in the virtual social network. Here, we can make an analogy to the virtual social network, using the complex multivariate network model of ball passing and other indicators.

Accordingly, the number of emails and other important information sent and received, and the positions of employees in the team reflect the important position of employees in the social network of the virtual company. We consider using the number of e-mails sent and received between team members and the average time spent on work communication on social platform to quantify communication ability indicators. In addition, the distribution of cultural sources of team members also affects the work efficiency of team members.

8 Strengths and weaknesses

Our complex multivariate network model has following strengths and weaknesses.

8.1 Strengths

- The model makes full use of the data given, and establishes a time-space dynamic network model of the ball passing, which can well reflect the internal mechanism of players' relation and realize the visualization of time and space.
- Combined with the actual situation of football field, the model analyzes the meaning and function of various actions, and defines some new indicators, such as the flexibility between players, the ability of possession, closeness, as well as the score of players' action contribution rate, which can reflect players' comprehensive indicators comprehensively.
- Owing to our clear logic, the model and figures are intuitive so that we can get all kinds of players' information, which is helpful for decision makers to make decisions.

- The model can be generalized, as is shown in the last problem, that is to say, how to apply the model to the evaluation of social networks is well illustrated.
- The model not only uses precise parameters, but also adds the impact of various factors, so that the stability of the model increases. Therefore, the calculation results obtained in use can well fit the actual situation.

8.2 Weaknesses

- In fact, in order to simplify the model, we still have some imperfections. For example, although we can infer the information of the substitute through the data, we do not clearly separate the player and the substitute in the image displayed, which is also because the data we process is relatively large.
- Although many researches have proved that the traditional clustering method can achieve community detection, the results we make by clustering have some unsatisfactory. Due to time constraints, we need to make further improvement according to the model.

9 Conclusions

In this paper, firstly, based on improved weighted matrices, we establish a complex multivariate network model of balling passes, taking each player as a node. Then, we define and calculate the node indicators in the network. Using traditional clustering methods for these indicators, we recognize the network pattern and its changes over time.

Secondly, we introduce several new indicators to evaluate the performance of soccer players, and use AHP to determine the importance weight of each new index for the performance of players, so as to get the comprehensive performance of each player and the team.

Thirdly, We make use of Stacked Bar Chart to get formulating structural strategy. We then change different indicators respectively, and make sensitivity analysis on the final score of the Huskies, thus providing the coach with team structual strategies to improve team success.

Finally, with the analogy idea, we extend the model to social network.Through case analysis, we provide the way of thinking on how to analyze network model indicators.

References

[1] Narizuka, Takuma, et al. "Statistical Properties of Position-Dependent Ball-Passing Networks in Football Games." Physica A: Statistical Mechanics and Its Applications, vol. 412, 2014, pp. 157–168., doi:10.1016/j.physa.2014.06.037.

[2] Opsahl, Tore, et al. "Node Centrality in Weighted Networks: Generalizing Degree and Shortest Paths." Social Networks, vol. 32, no. 3, 2010, pp. 245–251., doi:10.1016/j.socnet.2010.03.006.

[3] Javed, Muhammad Aqib, et al. "Community Detection in Networks: A Multidisciplinary Review." Journal of Network and Computer Applications, vol. 108, 2018, pp. 87–111., doi:10.1016/j.jnca.2018.02.011.

[4] Pe~na, Javier Lopez. "A Network Theory Analysis of Football Strategies." 2012.

[5] Duch, Jordi, et al. "Quantifying the Performance of Individual Players in a Team Activity." PLoS ONE, vol. 5, no. 6, 2010, doi:10.1371/journal.pone.0010937.

[6] Long, Hua, and Baoan Li. "Overlapping Community Identification Algorithm in Directed Network." Procedia Computer Science, vol. 107, 2017, pp. 527–532., doi:10.1016/j.procs.2017.03.105.

[7] Tan, Chong King, et al. "Factors Influencing Virtual Team Performance in Malaysia." Kybernetes, vol. 48, no. 9, 2019, pp. 2065–2092., doi:10.1108/k-01-2018-0031.

[8] Pappalardo, Luca, et al. "A Public Data Set of Spatio-Temporal Match Events in Soccer Competitions." Scientific Data, vol. 6, no. 1, 2019, doi:10.1038/s41597-019-0247-7.

[9] Raghavan, Usha Nandini, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks." Physical Review E 76.3 (2007): 036106.

Appendices

Appendix A Appendix

The solution to the four questions about the model

Question one:

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"#                          print


# In[225]:
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='Times New Roman'
matplotlib.rcParams['font.sans-serif']=['Times New Roman']


# #         weight         from anotheripynb

# In[3]:


import pandas as pd


# In[4]:


passing_copy_weight_O=pd.read_csv('passing_feature_weight_0.csv')


# In[5]:


map_player_id_O={'D1': 0,
 'D2': 1,
 'D3': 2,
 'D4': 3,
 'D5': 4,
 'D6': 5,
 'D7': 6,
 'F1': 7,
 'F2': 8,
 'F3': 9,
 'F4': 10,
 'F5': 11,
 'F6': 12,
 'G1': 13,
 'G2': 14,
 'M1': 15,
 'M2': 16,
 'M3': 17,
 'M4': 18,
 'M5': 19,
 'M6': 20,
 'M7': 21,
 'M8': 22}

# #         weight

# In[6]:


full=pd.read_csv("fullevents.csv")


# In[7]:
```

```
full_copy=full.copy()

# In[8]:

passing=pd.read_csv("passingevents.csv")

# In[9]:

passing_copy=passing.copy()

# In[10]:

full_copy.head(10)

# In[11]:

full_copy['Position']=full['OriginPlayerID'].apply(lambda x:x.split("_")[-1][0])

passing_copy['Position']=passing['OriginPlayerID'].apply(lambda x:x.split("_")[-1][0])

# In[15]:

passing_copy_H=passing_copy[passing_copy['TeamID']=='Huskies']

# In[17]:

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit_transform(passing_copy_H['OriginPlayerID'])

# In[19]:

map_player_id={'Huskies_D1': 0,
 'Huskies_D10': 1,
 'Huskies_D2': 2,
 'Huskies_D3': 3,
 'Huskies_D4': 4,
 'Huskies_D5': 5,
 'Huskies_D6': 6,
 'Huskies_D7': 7,
 'Huskies_D8': 8,
 'Huskies_D9': 9,
```

```
 'Huskies_F1': 10,
 'Huskies_F2': 11,
 'Huskies_F3': 12,
 'Huskies_F4': 13,
 'Huskies_F5': 14,
 'Huskies_F6': 15,
 'Huskies_G1': 16,
 'Huskies_M1': 17,
 'Huskies_M10': 18,
 'Huskies_M11': 19,
 'Huskies_M12': 20,
 'Huskies_M13': 21,
 'Huskies_M2': 22,
 'Huskies_M3': 23,
 'Huskies_M4': 24,
 'Huskies_M5': 25,
 'Huskies_M6': 26,
 'Huskies_M7': 27,
 'Huskies_M8': 28,
 'Huskies_M9': 29}


# In[20]:


passing_copy_H['OriginPlayerIDFit']=preprocessing.LabelEncoder().fit_transform(passing_c


# ###          fit_transformmap

# In[21]:


data = ['A', 'A', 'B', 'C', 'B', 'B'] # `y`

le = preprocessing.LabelEncoder()
mapped = le.fit_transform(data)

mapping = dict(zip(le.classes_, range(0, len(le.classes_)+1)))
mapped
print(mapping)
# {'A': 1, 'B': 2, 'C': 3}


# In[24]:


passing_copy_H['DestinationPlayerIDFit']=preprocessing.LabelEncoder().fit_transform(pass


# In[25]:


def getpair(data):
    left=data['OriginPlayerIDFit']
    right=data['DestinationPlayerIDFit']
    return (left,right)
```

```
passing_copy_H['PlayerIDFitPair']=passing_copy_H.apply(getpair,axis=1)
passing_copy_H['PassingBetweenPlayerCount']=[passing_copy_H.groupby('PlayerIDFitPair')['
```

```
# In[34]:
```

```
passing_copy_H['DestinationPosition']=passing_copy_H['DestinationPlayerID'].apply(lambda
```

```
# In[36]:
```

```
def getpair(data):
    left=data['Position']
    right=data['DestinationPosition']
    return (left,right)

passing_copy_H['PositionFitPair']=passing_copy_H.apply(getpair,axis=1)
```

```
# In[37]:
```

```
passing_copy_H['PassingBetweenPositionCount']=[passing_copy_H.groupby('PositionFitPair')
```

```
# In[39]:
```

```
passing_copy_H.to_csv('passing_feature.csv')
```

```
# In[40]:
```

```
PositionFitPairMap={('F','F'):1,('F','M'):0.75,('F','D'):0.5,('F','G'):0.25,('D','D'):1,
```

```
# In[41]:
```

```
alpha=1/3
beta=1/3
gamma=1/3
```

```
# In[42]:
```

```
passing_copy_H=pd.read_csv('passing_feature.csv')
```

```
# In[43]:
```

```
def dist(line):
    return pow(pow((line['EventOrigin_x']-line['EventDestination_x']),2)+pow((line['Even
```

```
# In[44]:


passing_copy_H['dist']=passing_copy_H.apply(dist,axis=1)


# In[46]:


0 in passing_copy_H['dist'].unique()


# In[47]:


# passing_copy_H['dist'].value_counts()
from collections import Counter

result = Counter(passing_copy_H['dist'])
print(result[25.9])


# In[48]:


passing_copy_H[passing_copy_H['dist']<1]


# In[49]:


#PositionFitPairMap,dist,PassingBetweenPlayerCount


# In[50]:


def PositionFitPairScoreMap(line):
    return PositionFitPairMap[eval(line['PositionFitPair'])]


# In[51]:


passing_copy_H['PositionFitPairScore']=passing_copy_H.apply(PositionFitPairScoreMap,axis


# In[53]:


passing_copy_H['OneDevideDist']=passing_copy_H.apply(lambda x:1.0/x['dist'] if x['dist']


# In[56]:
```

```
import numpy as np

# In[57]:

passing_copy_H['StdPassingBetweenPlayerCount']=(passing_copy_H['PassingBetweenPlayerCour

# In[59]:

def weight(line):
    return alpha*line['PositionFitPairScore']+beta*line['OneDevideDist']+gamma*line['Std

# In[60]:

passing_copy_H['weight']=passing_copy_H.apply(weight,axis=1)



# In[59]:

passing_copy_H.to_csv('passing_feature_weight.csv')

# In[62]:

passing_copy_weight=pd.read_csv('passing_feature_weight.csv')

# In[63]:

def get_avg(line):
    return passing_copy_weight.groupby('PlayerIDFitPair')['weight'].mean()[line['Player1

# In[64]:

passing_copy_weight['weight']=passing_copy_weight.apply(get_avg,axis=1)



# In[67]:

passing_copy_weight.to_csv('passing_feature_weight_mean.csv')

# In[68]:
```

```
passing_copy_weight.groupby('PlayerIDFitPair')['weight'].mean().to_csv('player_weight.cs

# In[69]:

passing_copy_weight2=pd.read_csv('player_weight.csv',header=None)

# In[70]:

def to_split(line):
    return eval(line[0])[0],eval(line[0])[1]

# In[71]:

passing_copy_weight2['left']=passing_copy_weight2.apply(to_split,axis=1).apply(lambda x:
passing_copy_weight2['right']=passing_copy_weight2.apply(to_split,axis=1).apply(lambda x

# In[72]:

passing_copy_weight2['weight']=passing_copy_weight2[1]

# In[73]:

passing_copy_weight2[['left','right','weight']].to_csv('player_weight_2.csv',index=None)

# In[75]:

def OutCount(line):
    return passing_copy_weight['OriginPlayerID'].value_counts()[line['OriginPlayerID']]

# In[76]:

passing_copy_weight['OutCount']=passing_copy_weight.apply(OutCount,axis=1)

# In[77]:

def InCount(line):
    return passing_copy_weight['DestinationPlayerID'].value_counts()[line['DestinationPl

# In[78]:
```

```
passing_copy_weight['InCount']=passing_copy_weight.apply(InCount,axis=1)


# In[87]:


first_half_data=passing_copy_weight[passing_copy_weight['MatchPeriod']=='1H']


# In[84]:


second_half_data=passing_copy_weight[passing_copy_weight['MatchPeriod']=='2H']

# In[223]:


first_half_data.to_csv('first_half_data.csv')

# In[85]:


def AvgPosition_x(line):
    return first_half_data.groupby('OriginPlayerID')['EventOrigin_x'].mean()[line['Origi


# In[86]:


#first_half_data=second_half_data

# In[88]:


first_half_data['AvgPosition_x']=first_half_data.apply(AvgPosition_x,axis=1)

# In[89]:


def AvgPosition_y(line):
    return first_half_data.groupby('OriginPlayerID')['EventOrigin_y'].mean()[line['Origi


# In[90]:


first_half_data['AvgPosition_y']=first_half_data.apply(AvgPosition_y,axis=1)

# In[91]:
```

```
def AvgWeight(line):
    return first_half_data.groupby('PlayerIDFitPair')['weight'].mean()[line['PlayerIDFit

# In[92]:

first_half_data['AvgWeight']=first_half_data.apply(AvgWeight,axis=1)

# ## OutAvgWeight,InAvgWeight

# In[93]:

def OutAvgWeight(line):
    return passing_copy_weight.groupby('OriginPlayerID')['weight'].mean()[line['OriginPl
def InAvgWeight(line):
    return passing_copy_weight.groupby('DestinationPlayerID')['weight'].mean()[line['Des

# In[94]:

passing_copy_weight['OutAvgWeight']=passing_copy_weight.apply(OutAvgWeight,axis=1)
passing_copy_weight['InAvgWeight']=passing_copy_weight.apply(InAvgWeight,axis=1)

# In[97]:

first_half_data_to_draw=first_half_data[['OriginPlayerID','DestinationPlayerID','OriginP

# In[93]:

first_half_data_to_draw.to_csv('first_half_data_to_draw.csv')

# In[95]:

'Huskies_M2'.split('_')[1]

# In[98]:

def cut_Huskies(line):
    return line['OriginPlayerID'].split('_')[1],line['DestinationPlayerID'].split('_')[1

# In[99]:

cut=first_half_data_to_draw.apply(cut_Huskies,axis=1)
```

```
first_half_data_to_draw['OriginPlayerID']=cut.apply(lambda x:x[0])
first_half_data_to_draw['DestinationPlayerID']=cut.apply(lambda x:x[1])


# In[101]:


positions=first_half_data_to_draw.drop_duplicates(subset=['OriginPlayerID'],keep='first'




# In[435]:


import matplotlib.pyplot as plt
import networkx as nx
import matplotlib.image as img

def minard_graph():
    data_F = first_half_data_to_draw[first_half_data_to_draw['Position']=='F']
    data_M = first_half_data_to_draw[first_half_data_to_draw['Position']=='M']
    data_D = first_half_data_to_draw[first_half_data_to_draw['Position']=='D']
    data_G = first_half_data_to_draw[first_half_data_to_draw['Position']=='G']
#24.0,54.9,340000,A,1
    #data_pos=first_half_data_to_draw[first_half_data_to_draw['Position']=='G']
    c = {}
#     for line in positions.split('\n'):
#         x, y, name = line.split(',')
#         c[name] = (float(x), float(y))
    for index,row in positions.iterrows():
        c[row['OriginPlayerID']]=(float(row['AvgPosition_x']),float(row['AvgPosition_y']
    g = []
    for data in [data_F, data_M, data_D, data_G]:
        G = nx.Graph()
        i = 0
        G.pos = {}  # location
        G.pop = {}  # size
        G.alpha = {}  # cuxi
        last=None

        for index,row in data.iterrows():
#OriginPlayerID DestinationPlayerID OriginPlayerIDFit   DestinationPlayerIDFit
#AvgWeight  AvgPosition_x   AvgPosition_y   PassingBetweenPlayerCount   Position
            G.pos[i] = (float(row['AvgPosition_x']), float(row['AvgPosition_y']))
            G.pop[i] = float(row['OutCount'])*2
            G.alpha[i]=row['AvgWeight']
            #G.add_edge(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit'])
            if last is None:
                last = i
            else:
                G.add_edge(i, last)
                last = i
            i = i + 1
        g.append(G)
    return g,c
if __name__ == "__main__":
```

```
    (g, player) = minard_graph()
#     imgP=plt.imread('court.jpg')
#     ax0.imshow(imgP)   #
#     import matplotlib.pyplot as plt
#     img = plt.imread("court.jpg")
#     fig, ax = plt.subplots()
#     ax.imshow(img,extent=[0,70,0,90])
    fig=plt.figure('court.jpg', figsize=(12*105/68.0,12))
#     plt.imshow(imgP)
#         fig = plt.figure(figsize=(16, 8))

    colors = ['#FBDB8E', '#FF6100', 'r','#FFFF00']
    for G in g:
        c = colors.pop(0)
        #print(G.pop)
        node_size = [G.pop[n] for n in G]
        width = [G.alpha[n] for n in G]
        width_next=width.pop(0)
        nx.draw_networkx_edges(G, G.pos, edge_color=c, width=2, alpha=width_next)
        nx.draw_networkx_nodes(G, G.pos, node_size=node_size, node_color=c, alpha=1)
        #nx.draw_networkx_nodes(G, G.pos, node_size=5, node_color='k')
    print(player)
    for c in player:
        x, y = player[c]
        plt.text(x, y , c,ha='center',color='#000000',fontsize=15)#city
    plt.show()
    plt.savefig("weight_first_half.png")

# In[325]:


from networkx.algorithms.community import kclique as kclique
klist = list(k_clique_communities(G,5))
print(klist)
nx.draw(G,pos =G.pos, with_labels=False)
nx.draw(G,pos = G.pos, nodelist = klist[0], node_color = 'b')
nx.draw(G,pos = G.pos, nodelist = klist[1], node_color = 'y')
plt.show()


# In[317]:


klist


# ###

# In[103]:


data_F = first_half_data_to_draw[first_half_data_to_draw['Position']=='F']
data_M = first_half_data_to_draw[first_half_data_to_draw['Position']=='M']
data_D = first_half_data_to_draw[first_half_data_to_draw['Position']=='D']
data_G = first_half_data_to_draw[first_half_data_to_draw['Position']=='G']
```

```
# In[104]:


kiin=passing_copy_weight['DestinationPlayerID'].value_counts()#kiin


# In[105]:


kiout=passing_copy_weight['OriginPlayerID'].value_counts()#kiout


# In[106]:


weightin=passing_copy_weight.groupby('OriginPlayerID')['InAvgWeight'].mean()#weightin
weightout=passing_copy_weight.groupby('OriginPlayerID')['OutAvgWeight'].mean()#weightout




# In[108]:


import math
math.isnan(weightin['Huskies_M7'])


# In[109]:


PlayerScore=[((kiout[i]*pow((weightout[i]/kiout[i]),0.8))+(kiin[i]*pow((weightin[i]/kiin


# In[110]:


from sklearn.cluster import KMeans
import numpy as np
X = np.array(PlayerScore).reshape(-1,1)
kmeans = KMeans(n_clusters=10, random_state=0).fit(X)
kmeans.labels_



# In[112]:


import sklearn.cluster as skc  #
dbscan=skc.DBSCAN(eps=0.1, min_samples=2).fit(X)
dbscan.labels_


# In[113]:


for i in range(len(passing_copy_weight['OriginPlayerID'].unique())):
```

```
    print(passing_copy_weight['OriginPlayerID'].unique()[i],dbscan.labels_[i])


# In[114]:


passing_copy_weight['OriginPlayerID'].unique(),kmeans.labels_


# In[115]:


for i in range(len(passing_copy_weight['OriginPlayerID'].unique())):
    print(passing_copy_weight['OriginPlayerID'].unique()[i],kmeans.labels_[i])


# In[ ]:




# ##        betweenness              shortestpath

# In[335]:



import networkx as nx
import pylab
import numpy as np
#
row=np.array([0,0,0,1,2,3,6])
col=np.array([1,2,3,4,5,6,7])
value=np.array([1,2,1,8,1,3,5])

print('                              ')
G=nx.DiGraph()
print('                              ...')
for i in range(0,np.size(col)+1):
    G.add_node(i)
print('                                    ...')
for i in range(np.size(row)):
    G.add_weighted_edges_from([(row[i],col[i],value[i])])

print('                       ...')
pos=nx.shell_layout(G)
print('                       ')
nx.draw(G,pos,with_labels=True, node_color='white', edge_color='red', node_size=400, alp
pylab.title('Self_Define Net',fontsize=15)
pylab.show()


'''
Shortest Path with dijkstra_path
'''
print('   d   i   j   k   s   t   r   a     ')
path=nx.dijkstra_path(G, source=0, target=7)
```

```
print('           07           ', path)
print('   d   i   j   k   s   t   r   a     ')
distance=nx.dijkstra_path_length(G, source=0, target=7)
print(distance)


# In[369]:



import networkx as nx
import pylab
import numpy as np
#
row=np.array([0,1])
col=np.array([1,2])
value=np.array([2,3])

print('                              ')
G=nx.DiGraph()
print('                              ...')
for i in range(29):
    G.add_node(i)
print('                                    ...')
for i in range(np.size(row)):
    G.add_weighted_edges_from([(row[i],col[i],value[i])])

print('                       ...')
pos=nx.shell_layout(G)
print('                       ')
nx.draw(G,pos,with_labels=True, node_color='white', edge_color='red', node_size=400, alp
pylab.title('Self_Define Net',fontsize=15)
pylab.show()


# In[458]:


import networkx as nx
import pylab
import numpy as np
#

print('                              ')
G=nx.Graph()
print('                              ...')
for i in range(29):
    G.add_node(i)
print('                                    ...')
weighted=[]
for index,row in passing_copy_weight2.iterrows():
    weighted.append(1/row['weight'])
    #G.weight[row['left']][row['right']]=1/row['weight']
    G.add_weighted_edges_from([(row['left'],row['right'],(1/row['weight'])/20)])
#     if(row['left']==20 and row['right']==19):
#         print(row['weight'],'-----------')
    G.edges[row['left'],row['right']]['weight']=(1/row['weight'])/20
```

```
print('                        ...')
pos=nx.shell_layout(G)
print('                        ')
nx.draw(G,pos,with_labels=True, node_color='white', edge_color='red', node_size=40, alph
pylab.title('DiGraph',fontsize=15)
pylab.show()
def shortest(row):
    return nx.dijkstra_path_length(G, source=row['left'], target=row['right'])
passing_copy_weight2['shortest_path']=passing_copy_weight2.apply(shortest,axis=1)
def onedevideweight(row):
    return 1/row['weight']
passing_copy_weight2['onedevideweight']=passing_copy_weight2.apply(onedevideweight,axis=


# In[469]:


import networkx as nx
import pylab
import numpy as np
#

print('                              ')
G=nx.Graph()
print('                              ...')
for i in range(29):
    G.add_node(i)
print('                                    ...')
weighted={}
for index,row in passing_copy_weight.iterrows():
    #G.weight[row['left']][row['right']]=1/row['weight']
                #G.add_weighted_edges_from([(row['left'],row['right'],1/row['weight'])])
#     if(row['left']==20 and row['right']==19):
#         print(row['weight'],'-----------')
    #G.add_edge(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit'])
    if (row['OriginPlayerIDFit'],row['DestinationPlayerIDFit']) not in weighted:
        weighted[(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit'])]=1
    else:
        weighted[(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit'])]+=1
   # G.edges[row['OriginPlayerIDFit'],row['DestinationPlayerIDFit']]['weight']+=1
for index,row in passing_copy_weight2.iterrows():
    #G.weight[row['left']][row['right']]=1/row['weight']
    if (row['left'],row['right']) in weighted:
        G.add_weighted_edges_from([(row['left'],row['right'],weighted[(row['left'],row['
        G.edges[row['left'],row['right']]['weight']=weighted[(row['left'],row['right'])]
    else:
        G.add_weighted_edges_from([(row['left'],row['right'],0)])
        G.edges[row['left'],row['right']]['weight']=0
#     if(row['left']==20 and row['right']==19):
#         print(row['weight'],'-----------')

print('                        ...')
pos=nx.shell_layout(G)
print('                        ')
nx.draw(G,pos,with_labels=True, node_color='white', edge_color='red', node_size=40, alph
pylab.title('DiGraph',fontsize=15)
pylab.show()
def shortest(row):
```

```
    return nx.dijkstra_path_length(G, source=row['left'], target=row['right'])
passing_copy_weight2['shortest_path']=passing_copy_weight2.apply(shortest,axis=1)
def onedevideweight(row):
    return 1/row['weight']
passing_copy_weight2['onedevideweight']=passing_copy_weight2.apply(onedevideweight,axis=
# for index,row in passing_copy_weight2.iterrows():
#     row['shortest_path']=nx.dijkstra_path_length(G, source=row['left'], target=row['ri

# '''
# Shortest Path with dijkstra_path
# '''
# print('   d   i   j   k   s   t   r   a     ')
# path=nx.dijkstra_path(G, source=0, target=12)
# print('           012          ', path)
# print('   d   i   j   k   s   t   r   a     ')
# distance=nx.dijkstra_path_length(G, source=0, target=12)
# print(distance)


# In[397]:


passing_copy_weight.columns


# In[345]:


from networkx.algorithms.community import kclique as kclique
klist = list(k_clique_communities(G,5))
print(klist)
nx.draw(G,pos =pos, with_labels=False)
nx.draw(G,pos = pos, nodelist = klist[0], node_color = 'b')
nx.draw(G,pos = pos, nodelist = klist[1], node_color = 'y')
plt.show()


# In[470]:


from networkx.algorithms.community import k_clique_communities
from networkx.algorithms.community.label_propagation import asyn_lpa_communities
from networkx.algorithms.community.kernighan_lin import kernighan_lin_bisection
from networkx.algorithms.community.label_propagation import label_propagation_communitie
#from networkx.algorithms.community import greedy_modularity_communities

list(asyn_lpa_communities(G,weight='weight'))
#list(greedy_modularity_communities(G,weight='weight'))


# In[482]:


thelist=[{0, 1, 3, 9, 16},
 {2,
  4,
  5,
  6,
```

```
  7,
  8,
  10,
  11,
  12,
  13,
  14,
  15,
  17,
  18,
  19,
  21,
  22,
  23,
  24,
  25,
  26,
  27,
  28,
  29},
 {20}]
for i in range(len(thelist)):
    thelist[i]=[reverse_map_player_id[j] for j in list(thelist[i])]
thelist


# In[116]:


x={0: 0.02040559998827244,
 1: 0.0,
 2: 0.014170749726328582,
 3: 0.016658963483015248,
 4: 0.014707541888736507,
 5: 0.009456737684730331,
 6: 0.008826661685196205,
 7: 0.006773420017006925,
 8: 0.0005562418554838753,
 9: 0.0,
 10: 0.014170749726328582,
 11: 0.011862914567508202,
 12: 0.0009432729767415365,
 13: 0.007476144386403045,
 14: 0.006381300251498059,
 15: 0.007884839385815455,
 16: 0.01768725579825337,
 17: 0.02040559998827244,
 18: 0.00013683634373289546,
 19: 0.0016961230953361382,
 20: 0.0025843257634386068,
 21: 0.00012963443090484831,
 22: 0.00044371196754563894,
 23: 0.015198977109235767,
 24: 0.013315973686823477,
 25: 0.0002988264271225732,
 26: 0.010834622252270075,
 27: 0.0,
 28: 0.0021043850785230096,
```

```
 29: 0.0039526298443431545}


# In[118]:


for i in list(map_player_id.keys()):
    print(i,x[map_player_id[i]])


# In[119]:


one_game=passing_copy_weight[['MatchID','MatchPeriod','EventTime','OriginPlayerIDFit','D
one_game=one_game[one_game['MatchID']==1]



# In[122]:


def H_event_time_deal(line):
    if line['MatchPeriod']=='2H':
        line['EventTime']+=2712.241
    return line['EventTime']


# In[123]:


one_game['EventTime']=one_game.apply(H_event_time_deal,axis=1)


# ## betweenness_centrality

# In[134]:


interval=350


# In[136]:


import networkx as nx
import pylab
import numpy as np

def centrality_level(passing_copy_weight2,denominator=11):
    #degree,degree_centrality,closeness_centrality,betweenness_centrality,transitivity,c
    G=nx.Graph()
    for i in range(30):
        G.add_node(i)
    for index,row in passing_copy_weight2.iterrows():
        G.add_weighted_edges_from([(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit
    pos=nx.shell_layout(G)
    return [sum(dict(nx.degree(G)).values())/denominator,                sum(dict(nx.degree_
sum(dict(nx.closeness_centrality(G)).values())/denominator,                sum(dict(nx.betwe
```

```
sum(dict(nx.clustering(G)).values())/denominator]


# In[137]:


degree=[]
degree_centrality=[]
clossness_centrality=[]
betweenness_centrality=[]
clustering=[]
betweenness_centrality_2_best=[]
for i in range(16):
    left=i*interval
    right=(i+1)*interval
    print(left,right)
    #between=one_game[(one_game['EventTime']>left) & (one_game['EventTime']<=right)]
    between=one_game[(one_game['EventTime']>left) & (one_game['EventTime']<=right)]


    between=between[((between['OriginPlayerIDFit']==3) | (between['OriginPlayerIDFit']==


    between_2_best=between[((between['OriginPlayerIDFit']==16) | (between['OriginPlayerI
    degree.append(centrality_level(between)[0])
    betweenness_centrality_2_best.append(centrality_level(between_2_best,2))
    degree_centrality.append(centrality_level(between)[1])
    clossness_centrality.append(centrality_level(between)[2])
    betweenness_centrality.append(centrality_level(between)[3])
#     transitivity.append(centrality_level(between)[4])
    clustering.append(centrality_level(between)[4])


# In[138]:


one_game[((one_game['OriginPlayerIDFit']==0) | (one_game['OriginPlayerIDFit']==2)) & ((o


# In[126]:


degree_centrality_season=[]
clossness_centrality_season=[]
betweenness_centrality_season=[]
clustering_season=[]
for i in range(1,39):
    #game=passing_copy_weight[(passing_copy_weight['MatchID']==i) & (((passing_copy_weig
    game=passing_copy_weight[(passing_copy_weight['MatchID']==i)]

    #between_2_best=between[((between['OriginPlayerIDFit']==16) | (between['OriginPlayer
    #betweenness_centrality_2_best.append(centrality_level(between_2_best,2))
    degree_centrality_season.append(centrality_level(game)[1])
    clossness_centrality_season.append(centrality_level(game)[2])
    betweenness_centrality_season.append(centrality_level(game)[3])
#     transitivity.append(centrality_level(between)[4])
    clustering_season.append(centrality_level(game)[4])
```

```
# In[ ]:
```

```
# In[127]:
```

```
one_game[((one_game['OriginPlayerIDFit']==0) | (one_game['OriginPlayerIDFit']==2)) & ((o
```

```
# In[128]:
```

```
between=one_game[(one_game['EventTime']>350) & (one_game['EventTime']<=700)]
between=between[((between['OriginPlayerIDFit']==0) | (between['OriginPlayerIDFit']==2))
between
```

```
# In[926]:
```

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                                SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,5555,350)
# degree_centrality=[]
# clossness_centrality=[]
# betweenness_centrality=[]
# clustering=[]
# betweenness_centrality_2_best=[]
# plt.title("value_of_graph_per_game_2_players")

# fontproperties           fontsize
plt.xlabel("time(s)")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,degree_centrality,label="degree_centrality")
plt.plot(x,clossness_centrality,label="clossness_centrality")
plt.plot(x,[25*i for i in betweenness_centrality],label="betweenness_centrality")
plt.plot(x,[i/4 for i in clustering],label="clustering")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25',"Clus
# plt.show()
plt.savefig("value_of_graph_per_game_2_players.png")
```

```
# In[921]:
```

```
import numpy as np
from matplotlib import pyplot as plt
```

```
import matplotlib

# fname                                                SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,5555,350)
# degree_centrality=[]
# clossness_centrality=[]
# betweenness_centrality=[]
# clustering=[]
# betweenness_centrality_2_best=[]
# plt.title("value_of_graph_per_game")

# fontproperties           fontsize
plt.xlabel("time(s)")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,degree_centrality,label="degree_centrality")
plt.plot(x,clossness_centrality,label="clossness_centrality")
plt.plot(x,[25*i for i in betweenness_centrality],label="betweenness_centrality")
plt.plot(x,[i/4 for i in clustering],label="clustering")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25',"Clus
# plt.show()
plt.savefig("value_of_graph_per_game.png")


# In[459]:


np.arange(0,5555,350)


# In[ ]:




# In[916]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(1,39)
#     degree_centrality_season.append(centrality_level(game)[1])
#     clossness_centrality_season.append(centrality_level(game)[2])
#     betweenness_centrality_season.append(centrality_level(game)[3])
# #     transitivity.append(centrality_level(between)[4])
#     clustering_season.append(centrality_level(game)[4])
# plt.title("value_of_graph_one_season")

plt.xlabel("game")
plt.ylabel("value")
```

```
# plt.plot(x,degree,label="degree")
plt.plot(x,degree_centrality_season,label="degree_centrality_season")
plt.plot(x,clossness_centrality_season,label="clossness_centrality_season")
plt.plot(x,[120*i for i in betweenness_centrality_season],label="betweenness_centrality_
plt.plot(x,[i/4 for i in clustering_season],label="clustering_season")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centralit
plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*120','Clu
# plt.show()
plt.savefig("value_of_graph_one_season.png")


# In[908]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                              SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(1,39)
#     degree_centrality_season.append(centrality_level(game)[1])
#     clossness_centrality_season.append(centrality_level(game)[2])
#     betweenness_centrality_season.append(centrality_level(game)[3])
# #     transitivity.append(centrality_level(between)[4])
#     clustering_season.append(centrality_level(game)[4])
# plt.title("value_of_graph_one_season_2_players")

# fontproperties           fontsize
plt.xlabel("game")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,degree_centrality_season,label="degree_centrality_season")
plt.plot(x,clossness_centrality_season,label="clossness_centrality_season")
plt.plot(x,[50*i for i in betweenness_centrality_season],label="betweenness_centrality_s
plt.plot(x,clustering_season,label="clustering_season")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centralit
plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*50',"Clus
# plt.show()
plt.savefig("value_of_graph_one_season_2_players.png")


# In[ ]:




# ##

# In[141]:


fullevents=pd.read_csv('fullevents.csv')


# In[144]:
```

```
fullevents[fullevents['TeamID']=='Huskies']['OriginPlayerID'].apply(lambda x:x.split('_'

# In[145]:

h_players=fullevents[fullevents['TeamID']=='Huskies']['OriginPlayerID'].apply(lambda x:x

# In[146]:

o_players=fullevents[fullevents['TeamID']!='Huskies']['OriginPlayerID'].apply(lambda x:x

# In[147]:

init={}
for i in h_players:
    init[i]=[]
init

# In[148]:

init2={}
for i in o_players:
    init2[i]=[]
init2

# In[149]:

init11={}
for i in h_players:
    init11[i]=0
init11
init12={}
for i in o_players:
    init12[i]=0
init12

# In[151]:

def centrality_level_per_person(passing_copy_weight2,denominator=11):
    #degree,degree_centrality,closeness_centrality,betweenness_centrality,transitivity,c
    G=nx.Graph()
    for i in range(30):
        G.add_node(i)
    for index,row in passing_copy_weight2.iterrows():
        G.add_weighted_edges_from([(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit
```

```
    pos=nx.shell_layout(G)
    return dict(nx.betweenness_centrality(G))


# In[152]:


def event_type_score(line):
    event_type=line['EventSubType']
    if event_type=='Ground attacking duel':
        return 4
    elif event_type=='Ground defending duel':
        return 4
    elif event_type=='Ground loose ball duel':
        return 1
    elif event_type=='Air duel':
        return 3
    elif event_type=='Launch':
        return 1
    elif event_type=='Clearance':
        return 3
    elif event_type=='Smart pass':
        return 5
    elif event_type=='Cross':
        return 2
    elif event_type=='Free kick cross':
        return -2
    elif event_type=='Foul':
        return -7
    elif event_type=='Acceleration':
        return 3
    elif event_type=='Goalkeeper leaving line':
        return 2
    elif event_type=='Out of game foul':
        return -10
    elif event_type=='Save attempt':
        return 4
    elif event_type=='Reflexes':
        return -5
    elif type(event_type)==float:
        return 0
    elif 'foul' in event_type:
        return -8
    else:
        return 0




# In[154]:


betweenness={'H':{'G1':[],'F1':[],'D1':[],'M1':[],'F2':[],'D2':[],'M2':[],'M3':[],'D3':[
for matchid in range(38):
    #betweenness_this_game={'H':{'G1':0,'F1':0,'D1':0,'M1':0,'F2':0,'D2':0,'M2':0,'M3':0
    this_match_H=passing_copy_weight[passing_copy_weight['MatchID']==(matchid+1)]
    this_match_O=passing_copy_weight_O[passing_copy_weight_O['MatchID']==(matchid+1)]
    y={}
```

```
    z={}
    centrality_level_value_H=centrality_level_per_person(this_match_H)
    centrality_level_value_O=centrality_level_per_person(this_match_O)
    for i in list(map_player_id.keys()):
        ii=i.split('_')[1]
        y[ii]=centrality_level_value_H[map_player_id[i]]
    for i in list(map_player_id_O.keys()):
        z[i]=centrality_level_value_O[map_player_id_O[i]]
    for i in list(y.keys()):
        betweenness['H'][i].append(y[i])
    for i in list(z.keys()):
        betweenness['O'][i].append(z[i])
# centrality_level_per_person(this_match)
# for i in list(map_player_id.keys()):
#     print(i,x[map_player_id[i]])
# #betweenness


# In[155]:


betweenness


# In[156]:


map_player_id


# In[481]:


reverse_map_player_id={value:key for key,value in map_player_id.items()}


# In[158]:


for i in list(reverse_map_player_id.keys()):
    reverse_map_player_id[i]=reverse_map_player_id[i].split('_')[1]


# In[160]:


def delete0(list_,zero_or_f1):
    a=list(filter(lambda number : number != zero_or_f1, list_))
    if a==[]:
        return [0]
    else:
        return a


# In[161]:


import copy
```

```
def get_average(dict,zero_or_f1=0):
    dict_copy=copy.deepcopy(dict)
    dicth=dict_copy['H']
    dicto=dict_copy['O']
    for i in list(dicth.keys()):
        dict_copy['H'][i]=np.mean(delete0(dict_copy['H'][i],zero_or_f1))
    for i in list(dicto.keys()):
        dict_copy['O'][i]=np.mean(delete0(dict_copy['O'][i],zero_or_f1))
    return dict_copy


# In[151]:


#betweenness,event_contribution,possession_charge,total_possession_time,success_guard_pe


# In[162]:


event_contribution={'H':{'G1':[],'F1':[],'D1':[],'M1':[],'F2':[],'D2':[],'M2':[],'M3':[]
#
possession_charge={'H':{'G1':[],'F1':[],'D1':[],'M1':[],'F2':[],'D2':[],'M2':[],'M3':[],
#                                        +1             -2
total_possession_time={'H':{'G1':[],'F1':[],'D1':[],'M1':[],'F2':[],'D2':[],'M2':[],'M3'
#


success_guard_percentage={'H':{'G1':[] },'O':{ 'G1':[],'G2':[]}}
success_shot_percentage={'H':{ 'F1':[], 'M1':[],'F2':[], 'M2':[],'M3':[], 'F3':[], 'M4':

for matchid in range(38):
    this_match=fullevents[fullevents['MatchID']==(matchid+1)]
    is_first=1
    #               2H
    half_time=0#         0
    is_1H=1

    charge_score={'H':{'G1':0,'F1':0,'D1':0,'M1':0,'F2':0,'D2':0,'M2':0,'M3':0,'D3':0,'D
    time={'H':{'G1':0,'F1':0,'D1':0,'M1':0,'F2':0,'D2':0,'M2':0,'M3':0,'D3':0,'D4':0,'F3
    event_score={'H':{'G1':0,'F1':0,'D1':0,'M1':0,'F2':0,'D2':0,'M2':0,'M3':0,'D3':0,'D4
    guard_count={'H':{'G1':0 },'O':{ 'G1':0,'G2':0}}
    success_guard_count={'H':{'G1':0 },'O':{ 'G1':0,'G2':0}}
    shot_count={'H':{ 'F1':0, 'M1':0,'F2':0, 'M2':0,'M3':0, 'F3':0, 'M4':0,'M5':0, 'M6':
    success_shot_count={'H':{ 'F1':0, 'M1':0,'F2':0, 'M2':0,'M3':0, 'F3':0, 'M4':0,'M5':
    for index,row in this_match.iterrows():
        if(is_first):
            last_teamid=row['TeamID'][0]#H or O
            last_time=row['EventTime']
            last_player=row['OriginPlayerID'].split('_')[1]
            last_event_type=row['EventType']
            before_turnover_player=[]
            is_first=0
        else:
            if(row['MatchPeriod']=='2H' and is_1H==1):
                half_time=last_time
                is_1H=0
            now_teamid=row['TeamID'][0]
```

```
            now_time=row['EventTime']+half_time
            now_player=row['OriginPlayerID'].split('_')[1]

            event_score[now_teamid][now_player]+=event_type_score(row)
            if(last_event_type=='Shot'):
                if last_player in shot_count[last_teamid]:
                    shot_count[last_teamid][last_player]+=1
                    if row['EventSubType']=='Reflexes':
                        success_shot_count[last_teamid][last_player]+=1
                if now_player[0]=='G':
                    if row['EventSubType']!='Reflexes':
                        success_guard_count[now_teamid][now_player]+=1
                    guard_count[now_teamid][now_player]+=1
            if (now_teamid==last_teamid):
                time[now_teamid][last_player]+=now_time-last_time
                before_turnover_player.append(last_player)
                for i in before_turnover_player:
                    charge_score[now_teamid][i]+=1
            else:
                charge_score[last_teamid][last_player]-=2
                before_turnover_player=[]
            last_teamid=row['TeamID'][0]
            last_time=row['EventTime']+half_time
            last_player=row['OriginPlayerID'].split('_')[1]
            last_event_type=row['EventType']
    for i in h_players:
        possession_charge['H'][i].append(charge_score['H'][i])
    for i in o_players:
        possession_charge['O'][i].append(charge_score['O'][i])
    for i in h_players:
        total_possession_time['H'][i].append(time['H'][i])
    for i in o_players:
        total_possession_time['O'][i].append(time['O'][i])
    for i in h_players:
        event_contribution['H'][i].append(event_score['H'][i])
    for i in o_players:
        event_contribution['O'][i].append(event_score['O'][i])

    for i in ['G1']:
        success_guard_percentage['H'][i].append(success_guard_count['H'][i]/guard_count[
    for i in ['G1','G2']:
        success_guard_percentage['O'][i].append(success_guard_count['O'][i]/guard_count[
    for i in list(shot_count['H'].keys()):
        success_shot_percentage['H'][i].append(success_shot_count['H'][i]/shot_count['H'
    for i in list(shot_count['O'].keys()):
        success_shot_percentage['O'][i].append(success_shot_count['O'][i]/shot_count['O'

# In[222]:


event_contribution


# In[154]:


import numpy as np
```

```
from matplotlib import pyplot as plt
import matplotlib

# fname                                              SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,11)
#     degree_centrality_season.append(centrality_level(game)[1])
#     clossness_centrality_season.append(centrality_level(game)[2])
#     betweenness_centrality_season.append(centrality_level(game)[3])
# #     transitivity.append(centrality_level(between)[4])
#     clustering_season.append(centrality_level(game)[4])
# plt.title("value_of_players_per_season_Huskies")

# fontproperties           fontsize
plt.xlabel("playerID")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,sorted([10000*v for k, v in get_average(betweenness)['H'].items()],reverse=Tr
plt.plot(x,sorted([v for k, v in get_average(event_contribution)['H'].items()],reverse=T
plt.plot(x,sorted([v for k, v in get_average(possession_charge)['H'].items()],reverse=Tr
plt.plot(x,sorted([v/1.5 for k, v in get_average(total_possession_time)['H'].items()],re
# plt.plot(x,[v for k, v in get_average(betweenness)['H'].items()],label="betweenness")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Betweenness*10000",'Event contribution','Possession charge','Total possessi
# plt.show()
plt.savefig("value_of_person_per_season_Huskies_____.png")


# In[1199]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                              SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,30)
#     degree_centrality_season.append(centrality_level(game)[1])
#     clossness_centrality_season.append(centrality_level(game)[2])
#     betweenness_centrality_season.append(centrality_level(game)[3])
# #     transitivity.append(centrality_level(between)[4])
#     clustering_season.append(centrality_level(game)[4])
# plt.title("value_of_players_per_season_Huskies")

# fontproperties           fontsize
plt.xlabel("playerID")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,[10000*v for k, v in get_average(betweenness)['H'].items()],label="betweennes
plt.plot(x,[v for k, v in get_average(event_contribution)['H'].items()],label="event_cor
plt.plot(x,[v for k, v in get_average(possession_charge)['H'].items()],label="possession
plt.plot(x,[v/1.5 for k, v in get_average(total_possession_time)['H'].items()],label="to
# plt.plot(x,[v for k, v in get_average(betweenness)['H'].items()],label="betweenness")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Betweenness*10000",'Event contribution','Possession charge','Total possessi
```

```
# plt.show()
plt.savefig("value_of_player_per_season_Huskies_30player.png")


# In[1198]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                          SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,24)
#     degree_centrality_season.append(centrality_level(game)[1])
#     clossness_centrality_season.append(centrality_level(game)[2])
#     betweenness_centrality_season.append(centrality_level(game)[3])
# #     transitivity.append(centrality_level(between)[4])
#     clustering_season.append(centrality_level(game)[4])
# plt.title("value_of_players_per_season_Huskies")

# fontproperties           fontsize
plt.xlabel("playerID")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,[10000*v for k, v in get_average(betweenness)['O'].items()],label="betweennes
plt.plot(x,[v for k, v in get_average(event_contribution)['O'].items()],label="event_cor
plt.plot(x,[v for k, v in get_average(possession_charge)['O'].items()],label="possession
plt.plot(x,[v/1.5 for k, v in get_average(total_possession_time)['O'].items()],label="to
# plt.plot(x,[v for k, v in get_average(betweenness)['H'].items()],label="betweenness")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Betweenness*10000",'Event contribution','Possession charge','Total possessi
# plt.show()
plt.savefig("value_of_player_per_season_Opponents_24player.png")



# In[895]:


import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='Times New Roman'
matplotlib.rcParams['font.sans-serif']=['Times New Roman']
labels=np.array(['Betweenness*10000','Event contribution','Possession charge','Total pos
nAttr=5
Python=np.array([get_average(betweenness)['H']['M1']*10000/sorted([10000*v for k, v in g
get_average(event_contribution)['H']['M1']/sorted([v for k, v in get_average(event_contr
get_average(possession_charge)['H']['M1']/sorted([v for k, v in get_average(possession_c
get_average(total_possession_time)['H']['M1']/2/sorted([v/2 for k, v in get_average(tota
get_average(success_shot_percentage,-1)['H']['M1']/sorted([v for k, v in get_average(suc
Python2=np.array([get_average(betweenness)['O']['M1']*10000/sorted([10000*v for k, v in
get_average(event_contribution)['O']['M1']/sorted([v for k, v in get_average(event_contr
get_average(possession_charge)['O']['M1']/sorted([v for k, v in get_average(possession_c
get_average(total_possession_time)['O']['M1']/2/sorted([v/2 for k, v in get_average(tota
get_average(success_shot_percentage,-1)['O']['M1']/sorted([v for k, v in get_average(suc
```

```
angles=np.linspace(0,2*np.pi,nAttr,endpoint=False)
Python=np.concatenate((Python,[Python[0]]))
Python2=np.concatenate((Python2,[Python2[0]]))
angles=np.concatenate((angles,[angles[0]]))
fig=plt.figure(facecolor="white")
plt.subplot(111,polar=True)
plt.plot(angles,Python,'bo-',color='g',linewidth=2)
plt.fill(angles,Python,facecolor='g',alpha=0.2)
plt.plot(angles,Python2,'bo-',color='r',linewidth=2)
plt.fill(angles,Python2,facecolor='r',alpha=0.2)
plt.thetagrids(angles*180/np.pi,labels)
plt.figtext(0.52,0.95,'Huskies_M1(green) VS Opponents_M1(red)',size=12,ha='center')
plt.grid(True)
plt.savefig('dota_radar.JPG')
plt.show()


# In[251]:


SHUFFLE=delete0([i if 'G' in i else 0 for i in [k for k,v in betweenness['H'].items()]],


# In[254]:


import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='Times New Roman'
matplotlib.rcParams['font.sans-serif']=['Times New Roman']
bt=[]
ec=[]
pc=[]
tpt=[]
for i in SHUFFLE:
    bt.append(get_average(betweenness)['H'][i]*10000*Q[0].real)
    ec.append(get_average(event_contribution)['H'][i]*Q[1].real)
    pc.append(get_average(possession_charge)['H'][i]*Q[2].real)
    tpt.append(get_average(total_possession_time)['H'][i]/2*Q[3].real)
x=range(len([k for k,v in betweenness['H'].items()]))
plt.figure(figsize=(15,10))
#
plt.bar(x, bt, color='r', label='Betweenness*10000*Q[0]')
plt.bar(x, ec, bottom=np.array(bt), color='#000000', label='Event Contribution*Q[1]')
plt.bar(x, pc, bottom=np.array(bt)+np.array(ec), color='b', label='Poesession Charge*Q[2
plt.bar(x, tpt, bottom=np.array(bt)+np.array(ec)+np.array(pc), color='c', label='Total P
# #
# plt.xlim(-2, 22)
# plt.ylim(0, 280)

#
plt.legend(loc='upper right')

plt.xticks([index + 0.2 for index in x], SHUFFLE)
plt.xlabel("Player")
plt.ylabel("Score")
```

```
plt.grid(axis='y', color='gray', linestyle=':', linewidth=2)
plt.savefig('player_score_bar.png')
plt.show()


# In[552]:


#bt,ec,pc,tpt
x0=np.array([i/Q[0].real for i in list(bt)])
x1=np.array([i/Q[1].real for i in list(ec)])
x2=np.array([i/Q[2].real for i in list(pc)])
x3=np.array([i/Q[3].real for i in list(tpt)])
def cal(list_):
    return np.mean(list_[0]*x0+list_[1]*x1+list_[2]*x2+list_[3]*x3)
table.iloc[0,0:4]=[Q[0].real,Q[1].real,Q[2].real,Q[3].real]
table.iloc[0,4]=0
table.iloc[0,5]=cal(table.iloc[0,0:4])
table.iloc[0,6]=0

table.iloc[1,0:4]=[Q[0].real*(1+0.15),Q[1].real,Q[2].real,Q[3].real]
table.iloc[1,4]=0.15
table.iloc[1,5]=cal(table.iloc[1,0:4])
table.iloc[1,6]=table.iloc[1,5]/table.iloc[0,5]-1
table.iloc[2,0:4]=[Q[0].real*(1-0.15),Q[1].real,Q[2].real,Q[3].real]
table.iloc[2,4]=-0.15
table.iloc[2,5]=cal(table.iloc[2,0:4])
table.iloc[2,6]=table.iloc[2,5]/table.iloc[0,5]-1

table.iloc[3,0:4]=[Q[0].real,Q[1].real*(1+0.15),Q[2].real,Q[3].real]
table.iloc[3,4]=0.15
table.iloc[3,5]=cal(table.iloc[3,0:4])
table.iloc[3,6]=table.iloc[3,5]/table.iloc[0,5]-1
table.iloc[4,0:4]=[Q[0].real,Q[1].real*(1-0.15),Q[2].real,Q[3].real]
table.iloc[4,4]=-0.15
table.iloc[4,5]=cal(table.iloc[4,0:4])
table.iloc[4,6]=table.iloc[4,5]/table.iloc[0,5]-1

table.iloc[5,0:4]=[Q[0].real,Q[1].real,Q[2].real*(1+0.15),Q[3].real]
table.iloc[5,4]=0.15
table.iloc[5,5]=cal(table.iloc[5,0:4])
table.iloc[5,6]=table.iloc[5,5]/table.iloc[0,5]-1
table.iloc[6,0:4]=[Q[0].real,Q[1].real,Q[2].real*(1-0.15),Q[3].real]
table.iloc[6,4]=-0.15
table.iloc[6,5]=cal(table.iloc[6,0:4])
table.iloc[6,6]=table.iloc[6,5]/table.iloc[0,5]-1

table.iloc[7,0:4]=[Q[0].real,Q[1].real,Q[2].real,Q[3].real*(1+0.15)]
table.iloc[7,4]=0.15
table.iloc[7,5]=cal(table.iloc[7,0:4])
table.iloc[7,6]=table.iloc[7,5]/table.iloc[0,5]-1
table.iloc[8,0:4]=[Q[0].real,Q[1].real,Q[2].real,Q[3].real*(1-0.15)]
table.iloc[8,4]=-0.15
table.iloc[8,5]=cal(table.iloc[8,0:4])
table.iloc[8,6]=table.iloc[8,5]/table.iloc[0,5]-1


# In[547]:
```

```
#table.iloc[[0,2],0:3]=[[0,0,0],[0,0,1]]
table.iloc[0,0:7]


# In[524]:


table['betweenness'][0]


# In[551]:


table=pd.DataFrame(np.zeros([9,7])).rename(columns={0:'betweenness',1:'event_contributio


# In[554]:


table.to_csv('mingganxing.csv')


# In[ ]:


import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='Times New Roman'
matplotlib.rcParams['font.sans-serif']=['Times New Roman']
labels=np.array(['Betweenness*10000','Event contribution','Possession charge','Total pos
nAttr=5
Python=np.array([get_average(betweenness)['H']['M1']*10000/sorted([10000*v for k, v in g
get_average(event_contribution)['H']['M1']/sorted([v for k, v in get_average(event_contr
get_average(possession_charge)['H']['M1']/sorted([v for k, v in get_average(possession_c
get_average(total_possession_time)['H']['M1']/2/sorted([v/2 for k, v in get_average(tota
get_average(success_shot_percentage,-1)['H']['M1']/sorted([v for k, v in get_average(suc
Python2=np.array([get_average(betweenness)['O']['M1']*10000/sorted([10000*v for k, v in
get_average(event_contribution)['O']['M1']/sorted([v for k, v in get_average(event_contr
get_average(possession_charge)['O']['M1']/sorted([v for k, v in get_average(possession_c
get_average(total_possession_time)['O']['M1']/2/sorted([v/2 for k, v in get_average(tota
get_average(success_shot_percentage,-1)['O']['M1']/sorted([v for k, v in get_average(suc

angles=np.linspace(0,2*np.pi,nAttr,endpoint=False)
Python=np.concatenate((Python,[Python[0]]))
Python2=np.concatenate((Python2,[Python2[0]]))
angles=np.concatenate((angles,[angles[0]]))
fig=plt.figure(facecolor="white")
plt.subplot(111,polar=True)
plt.plot(angles,Python,'bo-',color='g',linewidth=2)
plt.fill(angles,Python,facecolor='g',alpha=0.2)
plt.plot(angles,Python2,'bo-',color='r',linewidth=2)
plt.fill(angles,Python2,facecolor='r',alpha=0.2)
plt.thetagrids(angles*180/np.pi,labels)
plt.figtext(0.52,0.95,'Huskies_M1(green) VS Opponents_M1(red)',size=12,ha='center')
plt.grid(True)
```

```
plt.savefig('dota_radar.JPG')
plt.show()


# In[229]:


import matplotlib.pyplot as plt
import numpy as np

#
x = np.linspace(0, 20, 20)
y1 = np.random.randint(50, 100, 20)
y2 = np.random.randint(50, 100, 20)
y3 = np.random.randint(50, 100, 20)

#
plt.bar(x, y1, color='r', label='      ')
plt.bar(x, y2, bottom=y1, color='g', label='      ')
plt.bar(x, y3, bottom=y1+y2, color='c', label='      ')

#
plt.xlim(-2, 22)
plt.ylim(0, 280)

#
plt.legend(loc='upper right')
plt.grid(axis='y', color='gray', linestyle=':', linewidth=2)

plt.show()


# In[804]:


import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='SimHei'
matplotlib.rcParams['font.sans-serif']=['SimHei']
labels=np.array(['      ','         ','         ','         ','         '])
nAttr=5
Python=np.array([1,85,90,95,70])
angles=np.linspace(0,2*np.pi,nAttr,endpoint=False)
Python=np.concatenate((Python,[Python[0]]))
angles=np.concatenate((angles,[angles[0]]))
fig=plt.figure(facecolor="white")
plt.subplot(111,polar=True)
plt.plot(angles,Python,'bo-',color='g',linewidth=2)
plt.fill(angles,Python,facecolor='g',alpha=0.2)
plt.thetagrids(angles*180/np.pi,labels)
# plt.figtext(0.52,0.95,'  p  y  t  h  o  n    ',ha='center')
# plt.grid(True)
plt.savefig('dota_radar.JPG')
plt.show()


# In[907]:
```

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                              SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,11)

# fontproperties           fontsize
plt.xlabel("playerID")
plt.ylabel("value")
# plt.plot(x,degree,label="degree")
plt.plot(x,sorted([10000*v for k, v in get_average(betweenness)['O'].items()],reverse=Tr
plt.plot(x,sorted([v for k, v in get_average(event_contribution)['O'].items()],reverse=T
plt.plot(x,sorted([v/2 for k, v in get_average(possession_charge)['O'].items()],reverse=
plt.plot(x,sorted([v/2 for k, v in get_average(total_possession_time)['O'].items()],reve
# plt.plot(x,[v for k, v in get_average(betweenness)['H'].items()],label="betweenness")
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
plt.legend(["Betweenness*10000",'Event contribution','Possession charge/2','Total posses
# plt.show()
plt.savefig("value_of_person_per_season_Opponents.png")


# ## DBSCAN

# In[166]:


import sklearn.cluster as skc  #
dbscan=skc.DBSCAN(eps=1, min_samples=2).fit(StandardScaler().fit_transform( np.array([[v
dbscan.labels_


# In[ ]:


np.array(StandardScaler().fit_transform([[v*10000 for k, v in get_average(betweenness)['


# In[165]:


from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler


# ##############################################################################
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            random_state=0)

# X = np.array(sorted(np.array([[v for k, v in get_average(betweenness)['H'].items()],\
```

```
# #  [v for k, v in get_average(event_contribution)['H'].items()],\
# #  [v for k, v in get_average(possession_charge)['H'].items()],\
#  [v for k, v in get_average(total_possession_time)['H'].items()]]).reshape(30,2),key=l

X = np.array([[v for k, v in get_average(betweenness)['H'].items()], [v for k, v in get_


# X=np.array([float(i) for i in sorted(np.array(PlayerScore).reshape(-1,1),key=lambda x:


# In[174]:


db = DBSCAN(eps=40, min_samples=3).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

# Plot result
import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1] ,'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1],'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()


# In[1117]:


np.array([float(i) for i in sorted(np.array(PlayerScore).reshape(-1,1),key=lambda x:x[0]


# In[1087]:
```

```
np.array([[v for k, v in get_average(betweenness)['H'].items()],#  [v for k, v in get_av
#  [v for k, v in get_average(possession_charge)['H'].items()],\
 [v for k, v in get_average(total_possession_time)['H'].items()]]).reshape(30,2)


# In[1090]:


sorted(np.array([[v for k, v in get_average(betweenness)['H'].items()],#  [v for k, v in
#  [v for k, v in get_average(possession_charge)['H'].items()],\
 [v for k, v in get_average(total_possession_time)['H'].items()]]).reshape(30,2),key=lam


# In[1008]:


labels = db.labels_


# In[167]:


import numpy as np

A = np.array([[1, 5, 3,7],
   [1/5,1,3/5,3/7],
   [1/3,5/3,1,7/5],
    [1/7,7/3,5/7,1]])
m=len(A)                                    #
n=len(A[0])
RI=[0, 0, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.45, 1.49, 1.51]
R= np.linalg.matrix_rank(A)                 #
V,D=np.linalg.eig(A)                        #                                    VD
list1 = list(V)
B= np.max(list1)                            #
index = list1.index(B)
C = D[:, index]                             #
CI=(B-n)/(n-1)                              #             CI
CR=CI/RI[n]
if CR<0.10:
    print("CI=", CI)
    print("CR=", CR)
    print('                                AQ                                  ')
    sum=np.sum(C)

    Q=C/sum                                 #
    print(Q)                                #
else:
    print("                                   AA                                  ")


# In[1046]:


A = np.array([[1,6,0.5],
              [0.166,1,0.25],
              [2,4,1]])
SUMR = []
```

```
for i in range(0,3):
    tempsum=0
    for j in range(0,3):
        tempsum+=A[j][i]
    SUMR.append(tempsum)
A=np.row_stack((A,SUMR))
A


# In[169]:


Final_Score={'H':{'G1':[],'F1':[],'D1':[],'M1':[],'F2':[],'D2':[],'M2':[],'M3':[],'D3':[
for i in [k for k, v in betweenness['H'].items()]:
    a=0.9*(Q[0].real*10000*np.array(betweenness['H'][i])+
Q[1].real*np.array(event_contribution['H'][i])+                              Q[2].real*np.
Q[3].real*1/1.5*np.array(total_possession_time['H'][i]))
    if i in success_guard_percentage['H']:
        a+=0.05*100*(Q[0].real*np.array([j if j!=-1 else 0 for j in success_guard_percen
    if i in success_shot_percentage['H']:
        a+=0.05*100*(Q[0].real*np.array([j if j!=-1 else 0 for j in success_shot_percent
    Final_Score['H'][i]=list(a)
for i in [k for k, v in betweenness['O'].items()]:
    if betweenness['O'][i]==[]:
        a=0.9*(                              Q[1].real*np.array(event_contribution['O'][i]
Q[2].real*np.array(possession_charge['O'][i])+                               Q[3].real*1/1.
    else:
        a=0.9*(Q[0].real*10000*np.array(betweenness['O'][i])+
Q[1].real*np.array(event_contribution['O'][i])+                               Q[2].real*np.
Q[3].real*1/1.5*np.array(total_possession_time['O'][i]))
    if i in success_guard_percentage['O']:
        a+=0.05*100*(Q[0].real*np.array([j if j!=-1 else 0 for j in success_guard_percen
    if i in success_shot_percentage['O']:
        a+=0.05*100*(Q[0].real*np.array([j if j!=-1 else 0 for j in success_shot_percent
    Final_Score['O'][i]=list(a)
Final_Score


# In[181]:


AvgFinalScore={'H':{'G1':0,'F1':0,'D1':0,'M1':0,'F2':0,'D2':0,'M2':0,'M3':0,'D3':0,'D4':
for i in [k for k, v in Final_Score['H'].items()]:
    AvgFinalScore['H'][i]=np.mean(delete0(Final_Score['H'][i],0))
for i in [k for k, v in Final_Score['O'].items()]:
    AvgFinalScore['O'][i]=np.mean(delete0(Final_Score['O'][i],0))
AvgFinalScore


# In[1185]:


AvgFinalScore['H']=sorted(AvgFinalScore['H'].items(),key=lambda x:x[1],reverse=True)
AvgFinalScore['O']=sorted(AvgFinalScore['O'].items(),key=lambda x:x[1],reverse=True)


# #
```

```
# In[176]:


def pos(line):#G_count FM_count
    return line['OriginPlayerID'].split('_')[1]
def count_pos(df):
    g=0
    fm=0
    xx=df.apply(pos,axis=1).unique()
    for i in xx:
        if 'G' in i:
            g+=1
        if 'F' in i or 'M' in i:
            fm+=1
    return(g,fm)



# In[202]:


fullevents[fullevents['TeamID']=='Huskies'].ix[[5]]


# In[ ]:


#                          #betweenness_per_team



# In[173]:


def centrality_level_pc(passing_copy_weight2,denominator=11):
    #degree,degree_centrality,closeness_centrality,betweenness_centrality,transitivity,c
    G=nx.Graph()
    for i in range(30):
        G.add_node(i)
    for index,row in passing_copy_weight2.iterrows():
        G.add_weighted_edges_from([(row['OriginPlayerIDFit'],row['DestinationPlayerIDFit
    pos=nx.shell_layout(G)
    return np.mean(delete0([v for k,v in dict(nx.betweenness_centrality(G)).items()],0))


# In[177]:


fullevents.columns


# In[183]:


fullevents['DestinationPlayerID'].isnull()
```

```
# In[ ]:



# In[170]:


fullevents_D=fullevents[fullevents['DestinationPlayerID'].notnull()]


# In[171]:


fullevents_D['OriginPlayerIDFit']=preprocessing.LabelEncoder().fit_transform(fullevents_
fullevents_D['DestinationPlayerIDFit']=preprocessing.LabelEncoder().fit_transform(fullev


# In[174]:


betweenness_per_team={}
all_=pd.concat([passing_copy_weight,passing_copy_weight_O],axis=0)
for teamid in all_['TeamID'].unique():
    game_per_team=all_[all_['TeamID']==teamid]
    player_count=len(game_per_team['OriginPlayerID'].unique())
    aa=0
    tot=0
    for i in game_per_team['MatchID'].unique():
        gpt=game_per_team[game_per_team['MatchID']==i]
        tot+=centrality_level_pc(gpt)
        aa+=1
    betweenness_per_team[teamid]=tot/aa


# In[206]:


betweenness_per_team


# In[178]:


import warnings
warnings.filterwarnings('ignore')

event_contribution_per_team={}
possession_charge_per_team={}
total_possession_time_per_team={}
success_guard_percentage_per_team={}
success_shot_percentage_per_team={}
for teamid in fullevents['TeamID'].unique():
    game_per_team=fullevents[fullevents['TeamID']==teamid]
    player_count=len(game_per_team['OriginPlayerID'].unique())
```

```
G_count=count_pos(game_per_team)[0]
FM_count=count_pos(game_per_team)[1]
is_first=1
#               2H
half_time=0#                                        half_time
half_time2=0
charge_score=0#+1-2
time=0 #
event_score=0 #
guard_count=0 #
success_guard_count=0#
shot_count=0#
success_shot_count=0#

for index,row in game_per_team.iterrows():
    if(is_first):
        is_first=0
        before_turnover_player=0
        last_period=row['MatchPeriod']
    else:
        last_teamid=fullevents.ix[[index-1]]['TeamID'].iloc[0][0]
        last_period_to_cal=fullevents.ix[[index-1]]['MatchPeriod'].iloc[0]
        last_event_type=fullevents.ix[[index-1]]['EventType'].iloc[0]
        last_time_to_cal=fullevents.ix[[index-1]]['EventTime'].iloc[0]+half_time
        now_teamid=fullevents.ix[[index]]['TeamID'].iloc[0][0]

        now_player=row['OriginPlayerID'].split('_')[1]
        now_period=row['MatchPeriod']
        if(last_period!=now_period):
            half_time=last_time
        if(last_period_to_cal==now_period):
            last_time_to_cal=fullevents.ix[[index-1]]['EventTime'].iloc[0]+half_time
        now_time=row['EventTime']+half_time
        #print('---',now_time,last_time_to_cal,'---')
        event_score+=event_type_score(row)
        if(last_event_type=='Shot' and last_teamid!=teamid[0]):
            print(last_teamid,teamid,row['EventType'],now_teamid,row['EventSubType']
            if row['EventType']=='Save Attempt' and now_teamid==teamid[0]:
                guard_count+=1
                if row['EventSubType']!='Reflexes':
                    success_guard_count+=1
        if(last_event_type=='Shot' and last_teamid==teamid[0]):
            if row['EventType']=='Save Attempt' and now_teamid!=teamid[0]:
                shot_count+=1
                if row['EventSubType']=='Reflexes':
                    success_shot_count+=1
        if (now_teamid==last_teamid):
            #if(now_time<last_time_to_cal or now_time-last_time_to_cal>1000):
            #    print(now_time,last_time_to_cal,last_time,half_time)
            time+=now_time-last_time_to_cal
            before_turnover_player+=1
            charge_score+=before_turnover_player
        else:
            charge_score-=2
            before_turnover_player=0
        last_time=row['EventTime']+half_time
        last_period=row['MatchPeriod']
```

```
    event_contribution_per_team[teamid]=event_score/player_count/2
    possession_charge_per_team[teamid]=charge_score/player_count/2
    total_possession_time_per_team[teamid]=time/player_count/2
    success_guard_percentage_per_team[teamid]=success_guard_count/guard_count/G_count/2
    success_shot_percentage_per_team[teamid]=success_shot_count/shot_count/FM_count/2 if


# In[180]:


event_contribution_per_team['Huskies']/=19/2
possession_charge_per_team['Huskies']/=19/2
total_possession_time_per_team['Huskies']/=19/2
success_guard_percentage_per_team['Huskies']/=19/2
success_shot_percentage_per_team['Huskies']/=19/2


# In[210]:


team_score={}
for i in [k for k, v in total_possession_time_per_team.items()]:
    team_score[i]=(Q[0].real*10000*betweenness_per_team[i]+
Q[1].real*event_contribution_per_team[i]+                              Q[2].real*possessio
Q[3].real*1/1.5*total_possession_time_per_team[i])


#betweenness


# In[185]:


len_=0
H_score_per_game=np.zeros((38,))
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in betweenness['H'].items()]):
    zero+=i
    len_+=1
H_score_per_game+=zero/len_*10000*Q[0].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in event_contribution['H'].items()]):
    zero+=i
    len_+=1
H_score_per_game+=zero/len_*Q[1].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in possession_charge['H'].items()]):
    zero+=i
    len_+=1
H_score_per_game+=zero/len_*Q[2].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in total_possession_time['H'].items()]):
    zero+=i
    len_+=1
H_score_per_game+=zero/len_/1.5*Q[3].real
```

```
H_score_per_game=H_score_per_game/22.837916545013048*50.31779008693441
H_score_per_game


# In[403]:


len_=0
O_score_per_game=np.zeros((38,))
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in betweenness['O'].items()]):
    if i!=[]:
        zero+=i
        len_+=1
O_score_per_game+=zero/len_*10000*Q[0].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in event_contribution['O'].items()]):
    zero+=i
    len_+=1
O_score_per_game+=zero/len_*Q[1].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in possession_charge['O'].items()]):
    zero+=i
    len_+=1
O_score_per_game+=zero/len_*Q[2].real
len_=0
zero=np.zeros((38,))
for i in ([np.array(v) for k,v in total_possession_time['O'].items()]):
    zero+=i
    len_+=1
O_score_per_game+=zero/len_/1.5*Q[3].real
O_score_per_game=O_score_per_game/22.837916545013048*50.31779008693441
O_score_per_game


# In[186]:


betweenness['O']['D1']


# In[415]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                          SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,38)
# x=[k for k,v in match_score_pair_Huskies.items()]
# degree_centrality=[]
# clossness_centrality=[]
# betweenness_centrality=[]
```

```
# clustering=[]
# betweenness_centrality_2_best=[]
# plt.title("value_of_graph_per_game_2_players")

# fontproperties           fontsize
plt.xlabel("match")
plt.ylabel("Score of Huskies")
# plt.plot(x,degree,label="degree")
plt.plot(x,sorted(H_score_per_game),label="degree_centrality")
plt.plot(x,[list_goal_num[i] for i in [k for k,v in match_score_pair_Huskies.items()]],l

#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
#plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25','Clu
# plt.show()
plt.savefig("Score of Huskies.png")


# In[187]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                          SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = H_score_per_game
# x=[k for k,v in match_score_pair_Huskies.items()]
# degree_centrality=[]
# clossness_centrality=[]
# betweenness_centrality=[]
# clustering=[]
# betweenness_centrality_2_best=[]
# plt.title("value_of_graph_per_game_2_players")

# fontproperties           fontsize
plt.xlabel("Score of Huskies")
plt.ylabel("Actual Goal")
# plt.plot(x,degree,label="degree")
plt.scatter(x,[v for k,v in list_goal_diff.items()],label="Total Goal")
plt.scatter(x,[v for k,v in list_goal_num.items()],label="Total Goal")

#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
#plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25','Clu
# plt.show()
plt.savefig("Score of Huskies___.png")


# In[413]:


print(sort_dict(match_score_pair_Huskies,'key'))
a=[1,2,3]
list(reversed([v for k,v in sort_dict(match_score_pair_Huskies,'key').items()]))


# In[412]:
```

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                                SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = np.arange(0,38)

plt.xlabel("match")
plt.ylabel("Score")
# plt.plot(x,degree,label="degree")
plt.plot(x,[v/10 for k,v in score_H_minus_O_sort_dict.items()],label="Huskies-Opponents
plt.plot(x,[list_goal_diff[i] for i in [k for k,v in match_score_pair_Huskies.items()]],
plt.plot(x,[list_goal_num[i] for i in [k for k,v in match_score_pair_Huskies.items()]],l
plt.legend(["Huskies-Opponents Score/10","Goal Difference",'Total Goal'])
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
#plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25',"Clu
# plt.show()
plt.savefig("Relationship_between_scorediff&DG.png")


# In[314]:


[v for k,v in .items()]


# In[199]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                                SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = score_H_minus_O

plt.xlabel("score_H_minus_O")
plt.ylabel('DG/TG')
# plt.plot(x,degree,label="degree")
# plt.plot(x,[v/10 for k,v in score_H_minus_O_sort_dict.items()],label="Huskies-Opponent
# plt.legend()
plt.scatter(x,[v for k,v in list_goal_diff.items()],label="Goal difference")
plt.scatter(x,[v for k,v in list_goal_num.items()],label="Total goal")
plt.legend(["Goal difference",'Total Goal'])
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
#plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25',"Clu
# plt.show()
plt.savefig("Relationship_between_scorediff&DG_xy.png")


# In[423]:
```

```
#
from sklearn.linear_model import LogisticRegression
#
model=LogisticRegression()
#
model.fit(np.array(score_H_minus_O).reshape(-1,1),[v for k,v in list_goal_num.items()])


# In[427]:


model.predict([[-1000],[2],[3]])


# In[ ]:


#
a=model.intercept_
#
b=model.coef_
#
def y_pred(x,y):
    #
    z=a+b[0,0]*x+b[0,1]*y
    #
    return 1/(1+np.exp(-z))


# In[429]:


model.intercept_


# In[ ]:


#
a=model.intercept_
#
b=model.coef_
#
def y_pred(x,y):
    #
    z=a+b[0,0]*x+b[0,1]*y
    #
    return 1/(1+np.exp(-z))


# In[223]:


from scipy.optimize import curve_fit

def sigmoid2(x, a,c, d):
    #y = a/ (1 + np.exp(-1*(x-d)))
```

```
    y=np.log(1+np.exp(-c*x+d))
    return y

xdata = score_H_minus_O
ydata = [v for k,v in list_goal_num.items()]

popt, pcov = curve_fit(sigmoid2, xdata, ydata)
print(popt)

x = np.arange(-38,38)#np.linspace(-50, 50, 50)
y = sigmoid2(x, *popt)

pylab.plot(xdata, ydata, 'o', label='data')
pylab.plot(x,y, label='fit')
#pylab.ylim(0, 1.05)
pylab.legend(loc='best')
pylab.show()


# In[226]:


import matplotlib.pyplot as plt
import numpy as np

#
x = np.linspace(0, 20, 20)
y1 = np.random.randint(50, 100, 20)
y2 = np.random.randint(50, 100, 20)
y3 = np.random.randint(50, 100, 20)

#
plt.bar(x, y1, color='r', label='      ')
plt.bar(x, y2, bottom=y1, color='g', label='      ')
plt.bar(x, y3, bottom=y1+y2, color='c', label='      ')

#
plt.xlim(-2, 22)
plt.ylim(0, 280)

#
plt.legend(loc='upper right')
plt.grid(axis='y', color='gray', linestyle=':', linewidth=2)

plt.show()


# In[342]:


import numpy as np
from matplotlib import pyplot as plt
import matplotlib

# fname                                          SimHei.ttf
# zhfont1 = matplotlib.font_manager.FontProperties(fname="SimHei.ttf")

x = list(df_3.index)
```

```
# x=[k for k,v in match_score_pair_Huskies.items()]
# degree_centrality=[]
# clossness_centrality=[]
# betweenness_centrality=[]
# clustering=[]
# betweenness_centrality_2_best=[]
# plt.title("value_of_graph_per_game_2_players")

# fontproperties           fontsize
plt.xlabel("score_H_minus_O")
plt.ylabel('DG/TG')
# plt.plot(x,degree,label="degree")
# plt.plot(x,[v/10 for k,v in score_H_minus_O_sort_dict.items()],label="Huskies-Opponent
# plt.legend()
plt.scatter(x,np.array(df_3[1]),label="Goal difference")
plt.scatter(x,np.array(df_3[2]),label="Total goal")
plt.legend(["Goal difference",'Total Goal'])
#plt.legend(["degree","degree_centrality","clossness_centrality",'betweenness_centrality
#plt.legend(["Degree centrality","Clossness centrality",'Betweenness centrality*25',"Clu
# plt.show()
plt.savefig("Relationship_between_scorediff&DG_xy.png")


# In[323]:


df_3=pd.DataFrame([score_H_minus_O,[v for k,v in list_goal_diff.items()],[v for k,v in l


# In[324]:


df_3.head(5)


# In[336]:


df_3=df_3.groupby(0).mean()
df_3


# In[198]:


score_H_minus_O_sort_dict=sort_dict({i+1:score_H_minus_O[i] for i in range(0,38)})
score_H_minus_O_sort_dict



# In[540]:


win=list(H_score_per_game[[list(matches[matches['Outcome']=='win'].index)]])
tie=list(H_score_per_game[[list(matches[matches['Outcome']=='tie'].index)]])
loss=list(H_score_per_game[[list(matches[matches['Outcome']=='loss'].index)]])
avgwin=np.mean(win)
avgtie=np.mean(tie)
```

```
avgloss=np.mean(loss)
print(avgwin,avgtie,avgloss)
import numpy as np
import statsmodels.api as sm # recommended import according to the docs
import matplotlib.pyplot as plt

sample = win
ecdf1 = sm.distributions.ECDF(sample)
ecdf2 = sm.distributions.ECDF(tie)
ecdf3 = sm.distributions.ECDF(loss)
#                  X
x = np.linspace(35,63)
#               x
y1 = ecdf1(x)
y2 = ecdf2(x)
y3 = ecdf3(x)
#
plt.step(x, y1,label='win(EX=52.75)',linewidth=4)
plt.step(x, y2,label='tie(EX=50.29)',linewidth=4)
plt.step(x, y3,label='loss(EX=48,21)',linewidth=4)
plt.xlabel('Score')
plt.ylabel('ECDF of game')
plt.legend(['win(EX=52.75)','tie(EX=50.29)','loss(EX=48,21)'])
plt.show()
plt.savefig('ecdf.png')

import matplotlib.pyplot as plt
import numpy as np
T = x
#power1 = np.array([1.53E+03, 5.92E+02, 2.04E+02, 7.24E+01, 2.72E+01, 1.10E+01, 4.70E+00

from scipy.interpolate import spline
xnew = np.linspace(T.min(),T.max(),300) #300 represents number of points to make between
power_smooth = spline(T,y1,xnew)
power_smooth1 = spline(T,y2,xnew)
power_smooth2 = spline(T,y3,xnew)
plt.plot(xnew,power_smooth)
plt.plot(xnew,power_smooth1)
plt.plot(xnew,power_smooth2)
plt.plot(x,np.cumsum(y1)/sum(y1))
plt.show()


# In[487]:


win=list(np.array(score_H_minus_O)[[list(matches[matches['Outcome']=='win'].index)]])
tie=list(np.array(score_H_minus_O)[[list(matches[matches['Outcome']=='tie'].index)]])
loss=list(np.array(score_H_minus_O)[[list(matches[matches['Outcome']=='loss'].index)]])
avgwin=np.mean(win)
avgtie=np.mean(tie)
avgloss=np.mean(loss)
print(avgwin,avgtie,avgloss)
import numpy as np
import statsmodels.api as sm # recommended import according to the docs
import matplotlib.pyplot as plt

sample = win
```

```
ecdf1 = sm.distributions.ECDF(sample)
ecdf2 = sm.distributions.ECDF(tie)
ecdf3 = sm.distributions.ECDF(loss)
#                  X
x = np.linspace(-50,35)
#               x
y1 = ecdf1(x)
y2 = ecdf2(x)
y3 = ecdf3(x)
#
plt.step(x, y1,label='win(EX=11.389604832351818)',linewidth=4)
plt.step(x, y2,label='tie(EX=-4.02682154853765)',linewidth=4)
plt.step(x, y3,label='loss(EX=-15.053165490988189)',linewidth=4)
plt.xlabel('Score')
plt.ylabel('ECDF of game')
plt.legend(['win(EX=11.389604832351818)','tie(EX=-4.02682154853765)','loss(EX=-15.053165
plt.show()
plt.savefig('ecdf_2team_minus.png')


# In[541]:


from scipy.optimize import curve_fit

def sigmoid2(x, c, d):
    #y = a/ (1 + np.exp(-1*(x-d)))
    y=1/(1+np.exp(-c*(x-d)))
    return y

xdata = np.linspace(35,63)
ydata = ecdf1(np.linspace(35,63))
xdata2 = np.linspace(35,63)
ydata2 = ecdf2(np.linspace(35,63))
xdata3 = np.linspace(35,63)
ydata3 = ecdf3(np.linspace(35,63))
popt, pcov = curve_fit(sigmoid2, xdata, ydata,p0=(1,47))
print(popt)
popt2, pcov2 = curve_fit(sigmoid2, xdata2, ydata2,p0=(1,50))
print(popt2)
popt3, pcov3 = curve_fit(sigmoid2, xdata3, ydata3,p0=(1,53))
print(popt3)
x = np.arange(35,63)#np.linspace(-50, 50, 50)
y = sigmoid2(x, *popt)
y2 = sigmoid2(x, *popt2)
y3 = sigmoid2(x, *popt3)
pylab.step(xdata, ydata, label='Win(ECDF)')
pylab.plot(x,y, label='Win(Sigmoid)',linewidth=3)
pylab.step(xdata2, ydata2, label='Lie(ECDF)')
pylab.plot(x,y2, label='Tie(Sigmoid)',linewidth=3)
pylab.step(xdata3, ydata3, label='Loss(ECDF)')
pylab.plot(x,y3, label='Loss(Sigmoid)',linewidth=3)
#pylab.ylim(0, 1.05)
pylab.legend(loc='best')
plt.xlabel('Score')
plt.ylabel('ECDF of game')
pylab.show()
```

```
# In[539]:


from scipy.optimize import curve_fit

def sigmoid2(x, c, d):
    #y = a/ (1 + np.exp(-1*(x-d)))
    y=1/(1+np.exp(-c*(x-d)))
    return y

xdata = np.linspace(-50,35)
ydata = ecdf1(np.linspace(-50,35))
xdata2 = np.linspace(-50,35)
ydata2 = ecdf2(np.linspace(-50,35))
xdata3 = np.linspace(-50,35)
ydata3 = ecdf3(np.linspace(-50,35))
popt, pcov = curve_fit(sigmoid2, xdata, ydata,p0=(1,-15))
print(popt)
popt2, pcov2 = curve_fit(sigmoid2, xdata2, ydata2,p0=(1,-4))
print(popt2)
popt3, pcov3 = curve_fit(sigmoid2, xdata3, ydata3,p0=(1,11.3))
print(popt3)
x = np.arange(-50,35)#np.linspace(-50, 50, 50)
y = sigmoid2(x, *popt)
y2 = sigmoid2(x, *popt2)
y3 = sigmoid2(x, *popt3)
pylab.step(xdata, ydata, label='Win(ECDF)')
pylab.plot(x,y, label='Win(Sigmoid)',linewidth=3)
pylab.step(xdata2, ydata2, label='Lie(ECDF)')
pylab.plot(x,y2, label='Tie(Sigmoid)',linewidth=3)
pylab.step(xdata3, ydata3, label='Loss(ECDF)')
pylab.plot(x,y3, label='Loss(Sigmoid)',linewidth=3)
#pylab.ylim(0, 1.05)
pylab.legend(loc='best')
plt.xlabel('Score')
plt.ylabel('ECDF of game')
pylab.show()


# In[197]:


score_H_minus_O=[]
for id, row in matches.iterrows():
    score_H_minus_O.append(H_score_per_game[row['MatchID']-1]-team_score[row['OpponentID
score_H_minus_O


# In[399]:


team_score


# In[498]:
```

```
from SALib.sample import saltelli
from SALib.analyze import sobol
import matplotlib.pyplot as plt

def ET(X):
    # column 0 = x1, column 1 = x2, column 2 = x3
    return(Q[0].real*X[:,0]+Q[1].real*X[:,1]+Q[2].real*X[:,2]+Q[3].real*X[:,3])

problem = {'num_vars': 3,
      'names': ['x1', 'x2', 'x3'],
      'bounds': [[10, 100],
        [3, 7],
        [-10, 30]]
      }

# Generate samples
param_values = saltelli.sample(problem, 1000, calc_second_order=False)
print(param_values)
# Run model (example)
Y = ET(param_values)

# Perform analysis
Si = sobol.analyze(problem, Y, print_to_console=True)
# Print the first-order sensitivity indices
print (Si['S1'])

print (Si['ST'])
plt.subplots(figsize=(9, 9)) #
plt.barh(range(len(Si['S1'])), Si['S1'])
plt.barh(range(len(Si['ST'])), Si['ST'])
plt.show()


# In[ ]:




# In[190]:


def sort_dict(dict_,type='value'):
    if type=='key':
        return dict(sorted(dict_.items(), key=lambda x: x[0], reverse=True))
    if type=='value':
        return dict(sorted(dict_.items(), key=lambda x: x[1], reverse=True))


# In[191]:


match_id=1
list_score={}
for i in H_score_per_game:
    list_score[match_id]=i
    match_id+=1
match_score_pair_Huskies=sort_dict(list_score)
```

```
match_score_pair_Huskies


# In[193]:


matches=pd.read_csv('matches.csv')
matches.head(5)


# In[194]:


list_goal_diff={}
for id,row in matches.iterrows():
    list_goal_diff[id+1]=row['OwnScore']-row['OpponentScore']
list_goal_diff


# In[195]:


list_goal_num={}
for id,row in matches.iterrows():
    list_goal_num[id+1]=row['OwnScore']
list_goal_num
```