Problem Chosen	2020 MCM/ICM Summary Sheet	Team Control Number
D		2003280

Abstract

We built a passing network to describe the delivery relationship between team members, where each player is a node and each pass is a link between players. The weight of the nodes is proportional to the number of passes made by the player and the thickness of the edge is proportional to the number of passes made between the two players associated with the edge. We set up the adjacency weight matrix to measure the dyadic configuration pattern and obtain the maximum eigenvalue of the matrix. The largest eigenvalue λ_1 of the weighted adjacency matrix A of a network is a measure of the network strength. For the triadic configuration model, we first calculate the Clustering coefficient in the network, which can reflect triangulation between three players. Due to the sparse weight matrix of the three-way configuration, we analyzed the time series of the three who cooperated best in all the competitions.

For the measurement indexes, we analyze the two most important indexes of football players' ability at present, x_G (expected goals) and x_A(expected assists), and introduce a more fair evaluation of the Possession Value contributed by players in different positions to the team to evaluate the promotion effect of players' actions on the offensive. We analyze the influence of different players in different places the ball on offense, establish score probability parameter , which is determined by the position of the ball and the person who is handling the ball. By fitting several sets of shooting data, we got the ball by different players in different positions on the pitch control score probability function. We use this functionas the Possession Value of a player's move by analyzing the change in the scoring probability of the team's offense before and after the player's move.

We classify the Possession Value of each player and the interactions between every two players in different ways to obtain the degree of cooperation between members in Huskies and individual performance, which can be used to analyze the influence of Huskies players' coordination, personal ability, team strategy and opponents' counter-strategy on team scores. We used the optimized parameters to optimize our network model. After the above model establishment and analysis, we have a certain understanding of the construction of efficient team. We use our own model to propose the construction method of efficient team, and fully consider the subjective factors of team members except quantifiable indicators, and put forward our views on the construction of efficient team.

keywords Passing network;Team configuration model; Metrics analysis;Team building;

Contents

1 Introduction

1.1 Background

With the improvement of social connection, people begin to study strategies of team success. One of the most informative settings to explore is in competitive team sports. For team sports, many rules should be included, such as the number of players, allowable contact between players, their roles, their location and their movement.

Our company, Intrepid Champion Modeling, was asked to help understand the teams dynamics. The goals are to explore team dynamics throughout the game and develop a more effective winning strategy. The Huskies have provided all game information from last season.

1.2 Problem Restatement

Create a passing network of team members, where players are nodes and passes are links between players. The passing network is used to identify various network patterns.

Determine performance indicators which reflect successful team work. Many team level processes should be included, such as coordination among players, distribution of contributions or adaptability. Clarifying whether strategies are effective or dependent on opponents counter-strategies is of great importance.

Come up with effective strategies for the Huskies by using the insights gained from my teamwork model. Make some Suggestions to the coach about improving team success in the next season.

Understand the complex factors that make up certain groups and consider the settings for group dynamics team movements. Generalize findings to offer something about how to design more effective teams.

1.3 Problem Analysis

Once we have a set of data, we first read and process it in python. The first problem is to build a passing network, so we only use the data in the passingevent file. Specifically, we take a network for each game and integrate it into a huskies average delivery network. The weight of each node on the passing network represents the player's passes, and the thickness of the edge is proportional to the number of passes.[3]

Second, according to the dyadic and triadic configurations patterns in the prob-

lem, we set up different matrices to reflect the relationship between team members. The entries in the matrix are the number of passes between the players represented by this row and this column. We calculate the maximum eigenvalue of the adjacency weight matrix to represent the closeness of the connection between the player and the other players. In the triadic configuration, we first calculate the clustering coefficient, then we also construct weight matrix to analyze. Then We plot the passing performance of the best three players in the game to analyze the time series.

Then we set up a series of indicators to reflect the value of the players. Not every player in the front field has a shot. It is not the player who has no chance to shoot, there is no probability of scoring. When the possession reaches the front field, it should contribute to the score of the same attack, that is, there is probability of scoring in that position. To solve this problem, we assign a base scoring probability to each position. The next problem is that each player has a different probability of scoring in the same position, so we should assign a probability of scoring to each player according to their level.

In order to solve the problems of teamwork, individual contribution and team strategy, we conducted multi-dimensional contribution assessment for all players. In the development of society, more and more problems of team cooperation are encountered. We can correctly analyze the problems of team building through the analysis of this football match.

2 Assumptions

- Each player has performed relatively consistently throughout the season.
- Every player can shoot from any position on the field.
- We only consider passing between players when identifying network patterns.
- When building a team configuration model, we consider only the pass and duel behaviors.

3 List of Notation

Table 1: The List of Notation

Symbol	Meaning
P_e	A collection of all passing events
L	The set of players that appear in P_e
D	Weighted adjacency matrix
E	The determinant of the adjacent weight matrix
λ_1	The maximum eigenvalue of D

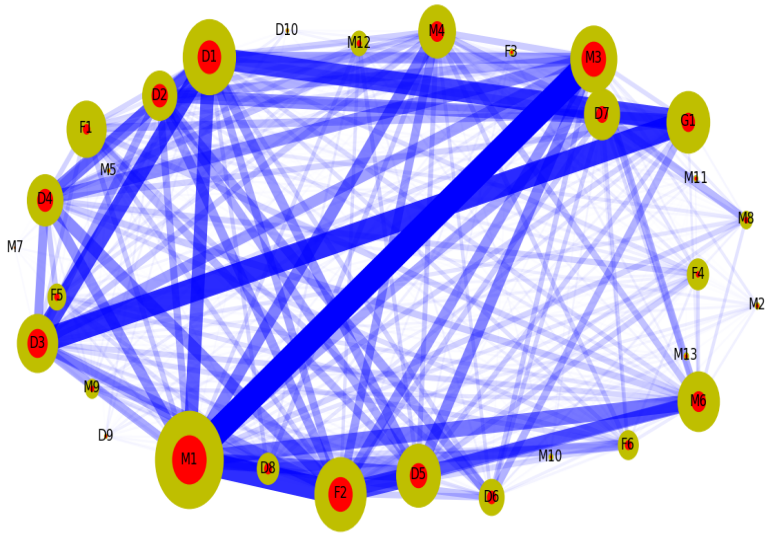
H_i	Clustering coefficient of node i in the network
H	Average clustering coefficient of network
b_{ij}	The total passing distance of i and binary j
R_e	The set of all triples
M	The set of members of a triple that appears in R_e
T	Triadic configuration matrix
P_{pass}	Passing distance
d_{max}	Maximum distance from player to goal
α	The number of people in the j shelter at that time
β_{base}	Player's base scoring percentage
GPE	Generalized pass event
d_t	The distance from a trigger event to a shot event
B	The set of triggering events
F	Triggering event
p_1	Shot probability of the receiver at the end of the field
p_2	shot probability of the server from the starting position
k	Shooting accuracy last season

4 Passing Network

We set up a graph model, where each player is a node and each pass constitutes a link between players. The weight of each node is the total number of passes made by this player, and the weight of the side is the total number of passes made by the two players associated with this side. As can be seen from the figure, the node is placed in the average position, and the thickness of the edge is proportional to the number of passes. Each node is represented by two concentric circles, and the ratio of the radius of the red circle to the radius of the larger circle is the pass success rate. To identify dyadic configuration, we first set up a set P_e that contains all the passing events, and then we put all the associated players into set L. Then we set up a matrix called D where the rows and columns represent different players in set L. The weighted adjacency matrix D is a $N \times N$ matrix whose elements a_{ij} contain passes going from player i to player j. Normalize the matrix by row, and then find the determinant of the matrix, which is represented by E. E, to some extent, reflects the cooperation between the two members of the whole team. By using matplotlib, we've plotted matrix D. From Figure 2, we can intuitively see the tacit understanding between husky team members. The analysis of the dyadic configuration pattern will help us to study the cooperation between players and the individual contribution.

The largest eigenvalue λ_1 of matrix D of a network can reflect the network strength. We know that the maximum eigenvalue can reflect the strength of the network[1], networks with higher number of passes will have a higher λ_1. We an-

Figure 1: The weighted adjacency matrix



alyze each of the 38 games and obtained all the maximum eigenvalues. This set of data played a role in the subsequent analysis, especially in the analysis of the enemy team data. And then we figure out the second smallest eigenvalue for all the games[2], which is also a useful metric to measure both structural and dynamical properties of networks.

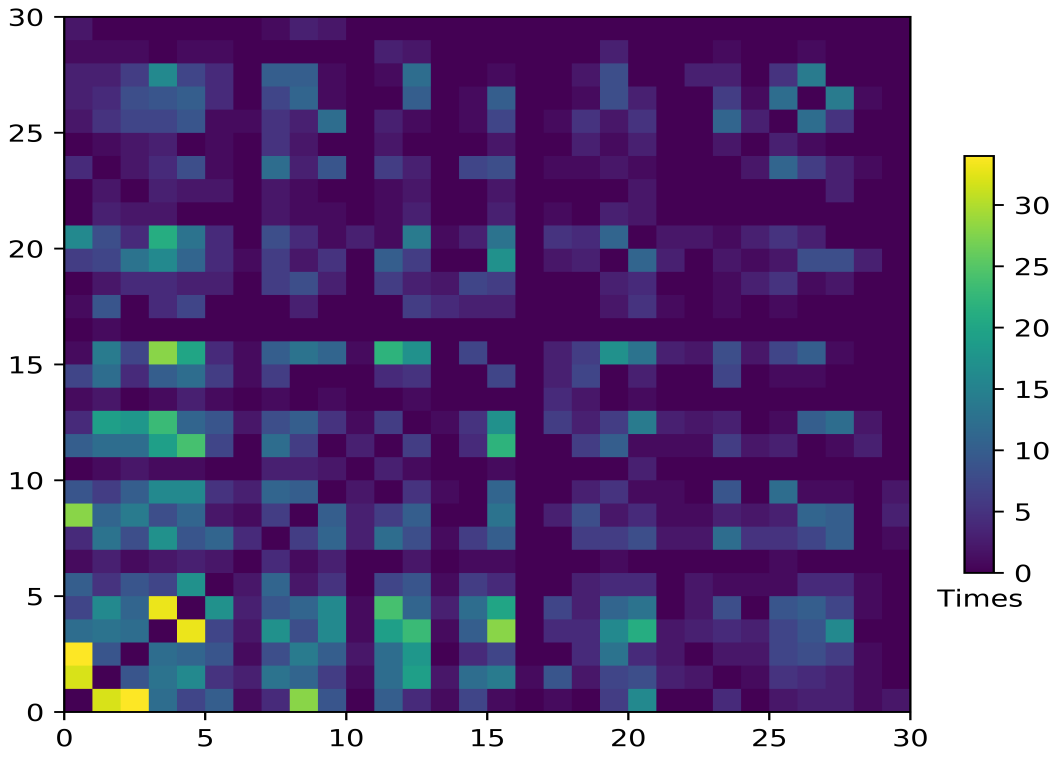
Generally, the local clustering coefficient is a parameter that describes the number of other nodes associated with this node. When weighting networks, we should consider not only the number of nodes connected between them, but also how the link weights are allocated. Because when a triangle exists that connects three nodes and a link lost between two nodes, there is another way to get to another node by the other two sides of the triangle. We make a detailed analysis of the network, and calculate the clustering coefficient[5] to measure triangulation between three players. The calculation formula is as follows:

$$H_i = \frac{\sum_{j,k} w_{ij} * w_{jk} * w_{ik}}{\sum_{j,k} w_{ij} * w_{ik}} \tag{1}$$

where j and k are two players of the team, w_{ij} and w_{ik} are the passes between a third player i. And then we calculate the average $H = \frac{1}{N}\sum_{i=1}^{N} H_{ij}$. We have calculated the clustering coefficient from the established network, and the final result is 0.46.

We built a set of all successful passes among three people, which is called R_e.

Figure 2: The weighted adjacency matrix

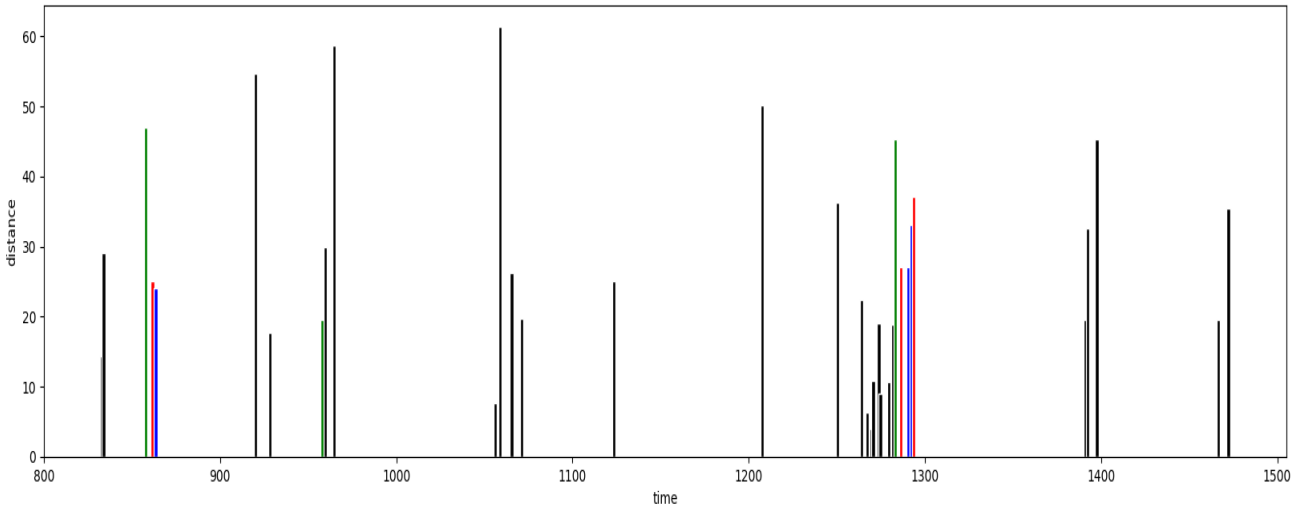


Each element of set R_e is a collection of triples. Second, for each of the components in set R_e, we take all the members of a triad. For all members of a triple, we put them into a set M. And then we take the members of the set as the rows of the ternary configuration matrix T, while each column of the matrix T is a binary group of any two elements in set M. The elements a_{ij} of the matrix M represents a particular function.

$$a_{ij} = (1 + \eta * b_{ij}) * p_{pass} \tag{2}$$

where η is a constant, p_{pass} is the passing distance. Similarly, we have drawn this matrix, which is too sparse to be meaningful for subsequent analysis. Therefore, we change our thinking and analyze the time series of the cooperation of three people in the game. Specifically, the relay pass events are studied. Then we analyze the triadic configuration throughout the last season, one of the games is shown below:

Figure 3: The configuration of three people in a match changes over time



5 Team configuration model based on metrics analysis

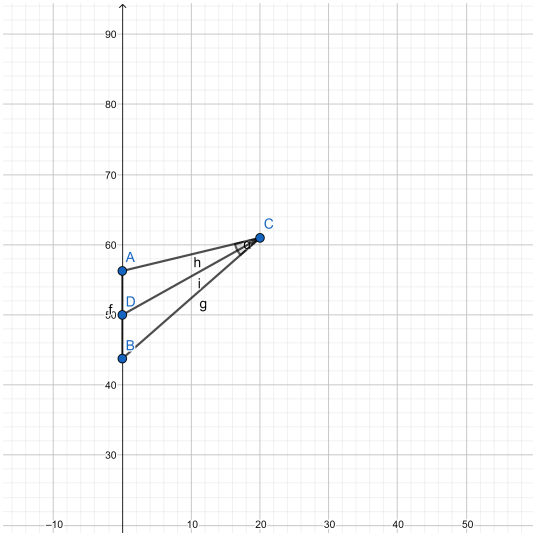
5.1 Model Preparation

For a team, each player has different responsibilities, and only a few players may have goals and assists throughout the season, but the positive effect of other players on the team cannot be measured by simple goals and assists. Therefore, when calculating the allocation of contribution of players, we refer to the evaluation method of Opta company for players, and introduce the index of Possession Value. At present, the two most effective indicators in football analysis are X_G and X_A, namely, the probability of achieving goals per foot and the probability of achieving assists per foot.

But these two indicators are often difficult to evaluate the contribution of different functional players. So we integrate the above two indicators, and for each person in front of each position gives the probability of a goal, by calculating each location to calculate the probability of the player's score, in the process of the ball is passed through the ball score probability to calculate contribution of each action to the goal, also is the ball Value (Value Possession).If the probability of scoring is increased by successive passes, then all players in the passing chain contribute more or less to the goal.Similarly, for every move that leads to the transfer of the ball to the opponent, we give them a negative contribution to each player's error during the match.

There are two key questions to consider when calculating the probability of each player scoring at each position up front. First of all, not every player in the front field has a shot. It is not the player who has no chance to shoot, there is no probability of scoring. When the possession reaches the front field, it should contribute to the score of the same attack, that is, there is probability of scoring in that position. To

Figure 4: geometric relationship between the position of the ball and the goal



solve this problem, we assign a base scoring probability to each position so that each player would contribute to the next offense after dribbling. The next problem is that each player has a different probability of scoring in the same position, so we should assign a probability of scoring to each player according to their level.

5.2 Modeling

To sum up, in order to construct the probability chart of each team member's score in the front field, we consider two indicators that affect the score success rate: one is the geometric relationship between the position of the ball and the goal, and the other is the scoring ability of the players in that position.

By analyzing the geometric relation, it can be concluded that the two most important factors that affect the goal are the distance from the goal and the angle range of the goal after the ball is shot. And from this we obtain the basic scoring probability.

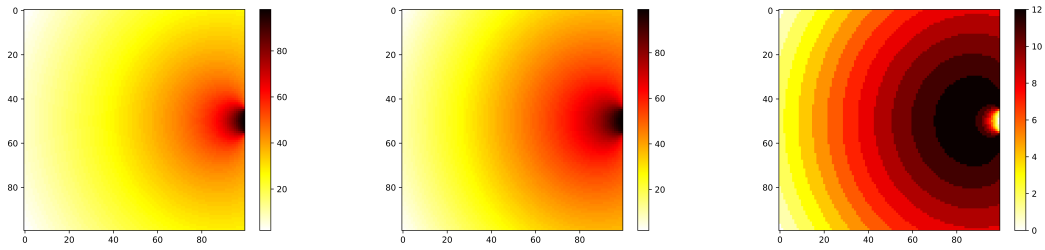
In Figure 4, AB represents the goal, D represents the midpoint of the goal, and C represents the player. We scale the entire football field according to the proportion of the standard football field and the given football field. d_{max} is the maximum value of CD, that is, the maximum distance a player can get from the center of the goal. α represents the angle between the left of the goal and the player and the right of the goal and the player.

From this we define a player's base scoring percentage β_{base}, which can be described by the following formula:

$$\beta_{base} = (1 - CD/d_{max} + sin(\alpha/2))/2 \tag{3}$$

where $1-CD/d_{max}$ reflects the effect of distance on scoring rate, $sin(\alpha/2)$ reflects the effect of angle on scoring rate.

When analyzing individual scoring ability of the players in that position, we calculate the scoring rate of the players in the whole season to get the scoring ability of the players.[4]



(a) pic1. (b) pic2. (c) pic4.

Figure 5: Player shooting percentage heat map

The matrix of the players' base-shot rate calculated from the left and right positions in the field is shown as a heat map on the left of Figure 5.

The numbers marked in the figure represent the percentage of the base scoring rate of the players. The dark part represents the low base scoring rate, and the light part represents the high base scoring rate. We can see that the distribution of the base scoring rate on the field is roughly in line with the actual situation.

In order to demonstrate the influence of individual shooting ability on scoring rate, we put 0.60 shooting accuracy into the function, and we can get the scoring ability matrix of a player with 0.60 shooting accuracy on the field, as shown in the middle of Figure. In order to show the addition of individual ability of players to their scoring rate, we subtract the two matrices and draw a heat map as shown in the right.

So, we have the individual scoring ability and the basic scoring ability of the players, and we combine them. We select $Huskies_{F1}$ and $Huskies_{F4}$ with the most shooting data in the whole season, and cluster their shooting data to obtain their shooting accuracy in part of the front field. Based on this, we fit the functions 4 of individual scoring ability and basic scoring ability of players to calculate the scoring rate of players at different positions, and the value of the parameter l is 0.996.

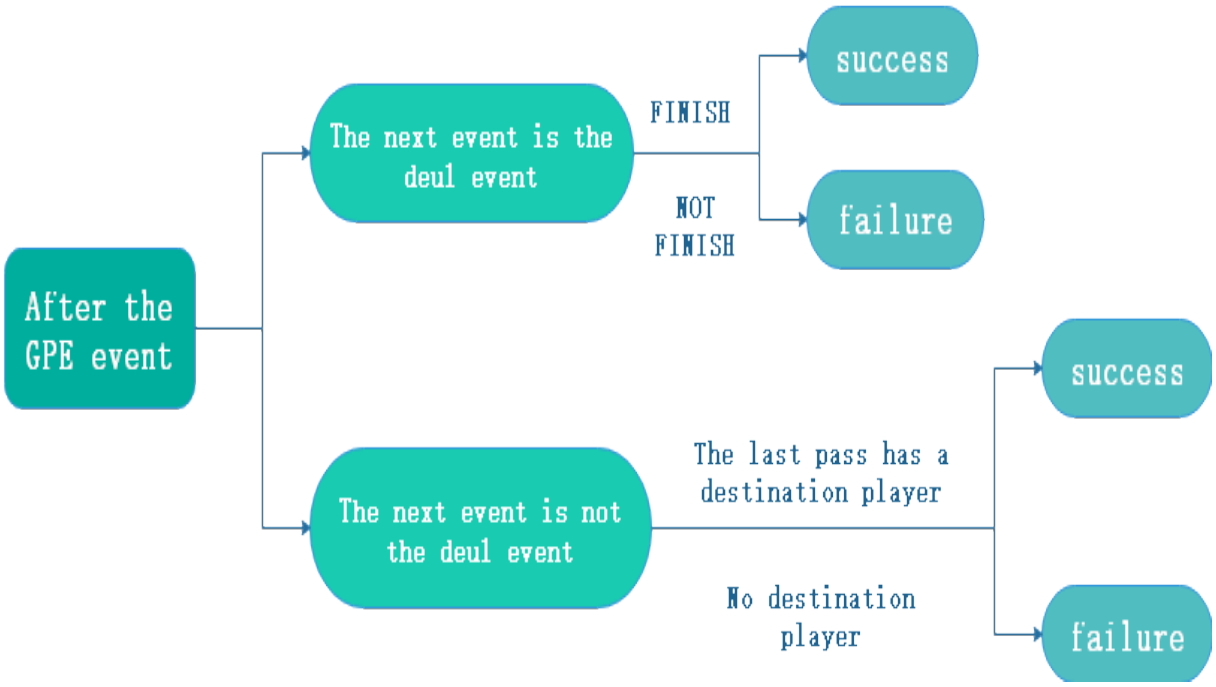
$$((l-k)*(1-CD/d_{max}+sin(\alpha/2))/2+k*sin(((1-CD/d_{max}+sin(\alpha/2))/2)*\Pi/2)) \quad (4)$$

Definition 5.1. *The relay pass is consecutive passes between players of the same team.*

Definition 5.2. *Relay distance is distance between start and end of relay.*

Without loss of generality, we define a generalized pass event called GPE, which concludes the pass between two people and the relay pass. We develop a standard

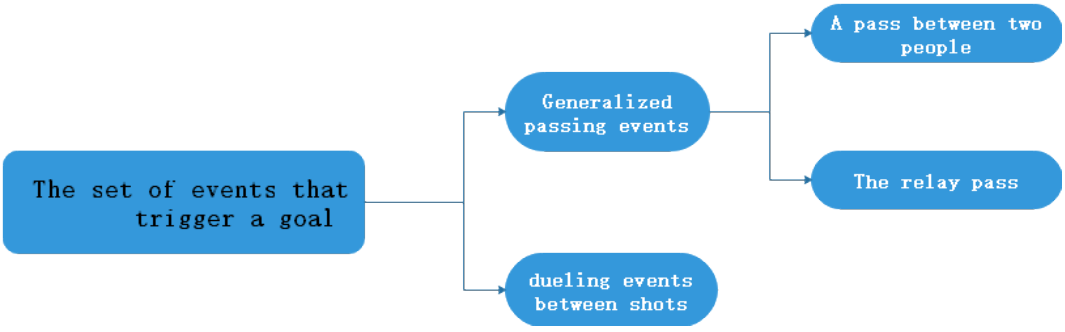
Figure 6: The success or failure of the GPE event



to measure the success of GPE. The passing distance of a generalized passing event is the distance of a single pass or the distance of a relay pass.

The criteria for success and failure in generalized passing events are shown in Figure 6 above. As illustrated in Figure 7, The trigger events includes dueling events between shots and generalized passing events, and all the trigger events constitute the set B. One thing to note here is that we are talking about the contribution of our players to the goal. Therefore, The generalized pass events mentioned above include only our team members.

Figure 7: Events that trigger a goal



Definition 5.3. *The distance from a trigger event to a shot event is defined as the total number of triggers that cause the shot event* $+1$*, which is represented by* d_t*.*

The contribution of a trigger event to the shot event is the sum of the contribu-

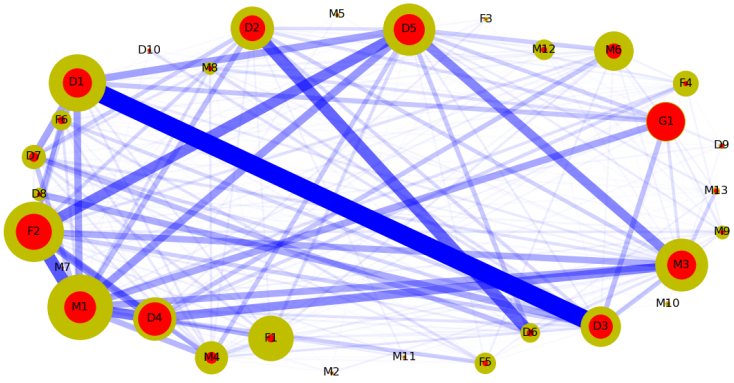
tions of all players involved to the shot event, and the contribution of the whole set of triggers to the shot events is the weighted sum of the contributions and d_t. The contribution of a team member to a triggering event F can be given by the following formula:

$$F = \begin{cases} p_1 - p_2 & \text{The single pass is successful} \\ 0 & \text{The single pass is failing} \\ p_1 - p_2 & \text{The relay pass is successful} \\ 0 & \text{The relay pass is failing} \\ 1 & \text{The dueling event is successful} \\ 0 & \text{The dueling event is failing} \end{cases} \tag{5}$$

where p_1 is the shot probability of the receiver at the end of the field, and p_2 is the shot probability of the server from the starting position.

After analyzing the above series of indicators, we introduce the calculated contribution into the model. Figure 8 is the network diagram after weighted contribution. The size of the nodes is proportional to the size of the contribution, which are still represented by concentric circles. The ratio of the radius of the red circle to the difference between the two circles is the ratio of pass events to the number of duels. The thickness of the edge between the nodes is proportional to the contribution of the pass.

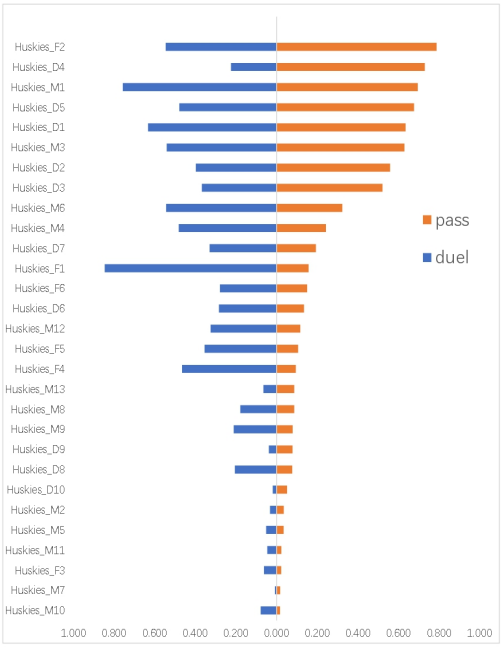
Figure 8: new network diagram



6 Huskies organizational strategy

Through the above parameter setting, we can calculate the promotion effect of each movement of each player on the team offensive. With this model, we focus on the effects of pass and duel. We figure out the value of each player's contribution

Figure 9: Value of each player's contribution



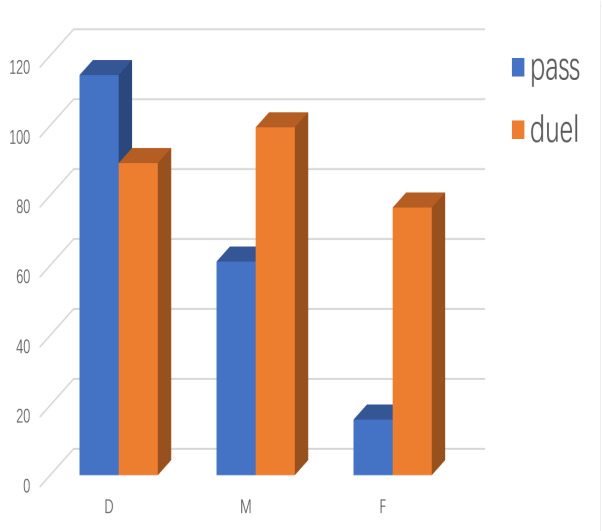
to pass and duel throughout the season, but the actual number of appearances and duration of each player's activity varied throughout the season, so we normalize this data to get the value of each player's contribution, as shown in the figure below:

This picture shows the value of each player's contribution, from which we can draw some simple conclusions. Among the players who contributed the most, the modified and forward players account for the majority, indicating that Huskies' midfielders and defenders are better at creating good offensive opportunities for the team through subtle passing. This data is also relatively reasonable, because we assume that the probability of scoring on the field is relatively regular. In fact, the forward players who are active in the front field are faced with the offensive pressure and defensive intensity, which make them unable to make the ball position of the players in the back field and the midfielders to make a large burst. $Huskies_{F1}$ is particularly unbalanced, with his individual duel being the best on the team, but his relatively small contribution to the ball suggests he prefers to attack on his own rather than with his teammates. $Huskies_{D4}$, by contrast, is not a great player on a duel, playing in the lower half of the team, but his pass ability is second to none, showing he is good at creating good chances for his teammates.

Here's an analysis of Huskies team communication at different positions (Figure10). We add up all players' pass and duel contributions to get an overall picture of the performance of all players at each position over the course of the season.

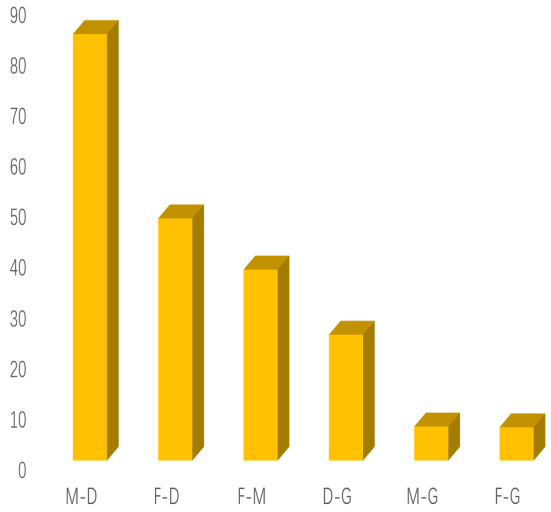
We can see the offensive style of each team member. Forward players tend to make breakthroughs with their own personal abilities to get good offensive oppor-

Figure 10: All players' pass and duel contributions



tunities for the team, while defense and kickoff players are more comprehensive We take the first game as an example to analyze the reasons for Huskies' victory. We plot the number of interactions between each part of the team in the first game to reflect the cooperation between each part of the team.

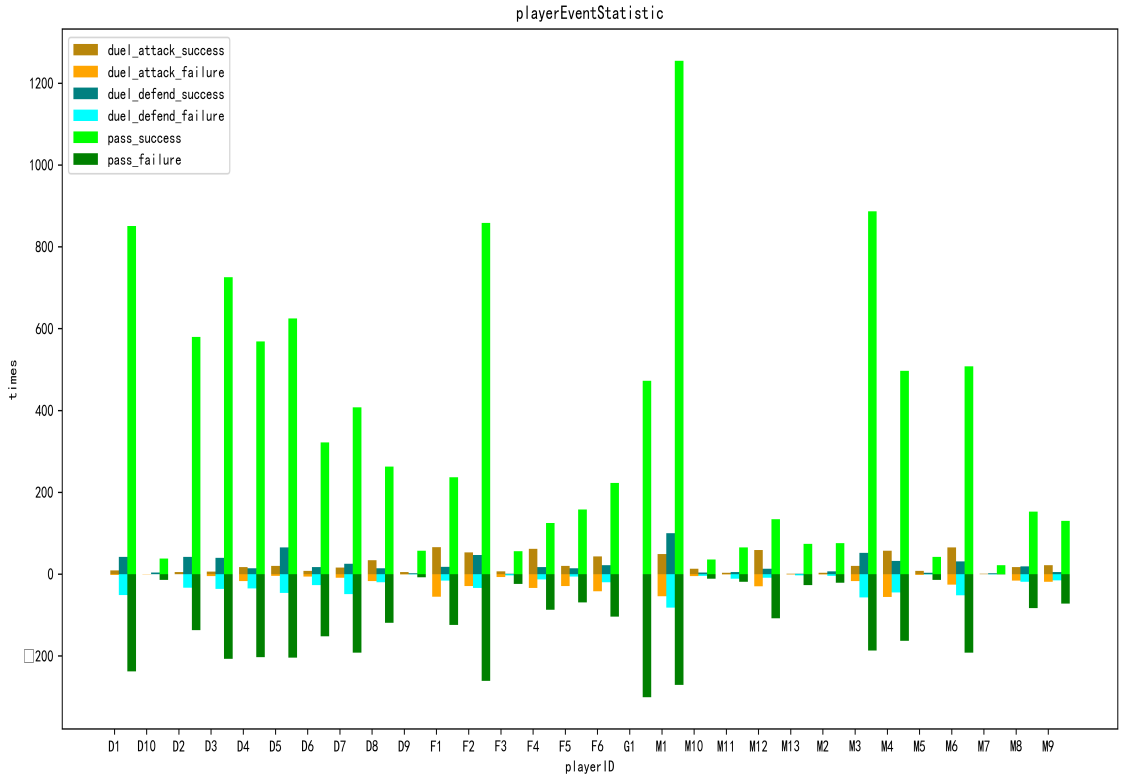
Figure 11: Value of each player's contribution



We can find that the most interactive players in the whole game are the backpackers and the backpackers, followed by the forward players and the backpackers. This shows that Huskies' midfielders are very capable of linking the Huskies' defense and offense, which makes Huskies very quick to switch between defense and offense, which is an important reason why Huskies can win. In summary, we can conclude that one of the most effective organizational strategies for the Huskies is to gain an advantage by using the defense and the kickoff kickers to quickly pass

the ball forward after a successful defense. Next season, the Huskies should focus more on the cooperation between the players in front of the field and attack more cooperatively, so as to achieve more effective offense.

Figure 12: Value of each player's contribution



7 Team building

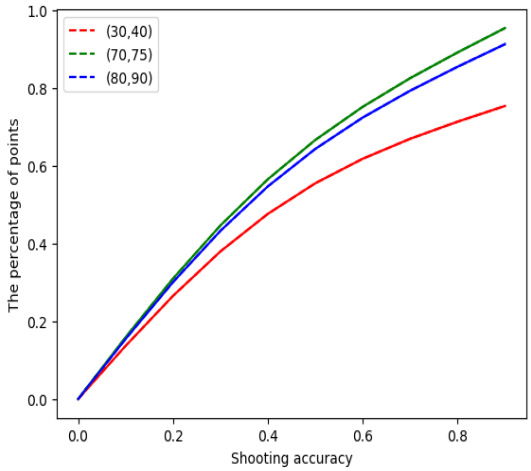
We found that in a team, compared with the outstanding individual ability, group cooperation and the fit between people's ability are more important for the effect of group writing. To design an effective team, first we need to define the team's goals. Take football as an example, the purpose of football is to score and stop the other team from scoring. In Huskies, for example, $Huskies_{F1}$ is good at penetration, while $Huskies_{F2}$ is good at passing to create opportunities for others, and $Huskies_{M1}$ is more comprehensive. By identifying the strengths and weaknesses of their own players, they can make the most of their abilities. Finally, assign responsibilities to team members according to their characteristics. The team is not just an individual, but the group that is assigned to it. Still take Huskies team as an example, $Huskies_{F1}$ is good at personal attack and $Huskies_{F2}$ is good at clever

pass to create good offensive opportunities for their own players. Obviously, the ability of the two players very well, we will make them up for a group to attack, and so on, our team can be divided into forward, midfield, defense and goalkeeper to separately and the connection of both offensive and defensive events. The above mentioned team design and construction are based on quantifiable indicators. In fact, in team building, many unquantifiable factors also affect the efficiency of the team, such as the relationship between team members, psychological status, morale and so on. These factors should also be taken into account to create a general model that describes team performance.

8 Sensitivity analysis

We probe into the sensitivity of the parameter in our Team configuration model. As shown in Figure16, when we are changing shooting accuracy $k$the percentage of points moves upward. In our model, one of the most important quantities is the scoring rate of players at each position in the field. In the figure below, we take out the coordinates of three points for analysis. Taking $(30, 40)$ as an example, when the value of k increased from 0.6 to 0.8, the percentage of points increased from 0.6 to 0.7. When the value of k changed, the model changed a lot, which means our model is sensitive to the parameter k.

Figure 13: Sensitivity analysis of k



9 Strengths and Weaknesses

9.1 Strengths

- Instead of using a single parameter to display the abilities of the team members and the degree of cooperation between the team members, the matrix is adopted, so that multiple aspects of the team members' abilities can be clearly displayed, and the connections between the team members can be visualized;
- It quantifies the influence of team members' behaviors on the whole team, which is more scientific than judging individual abilities by intuitive feelings;
- Possession Value parameter is introduced to analyze the role played by players at different positions on offense as precisely as possible, which maximally avoids the inaccurate evaluation of defenders' offensive behaviors by traditional x_G and x_A parameters;
- Define a series of events that are difficult to quantify on the field, and then list their contributions to the goal, which is conducive to the evaluation of the situation and the quantification of the contribution of the players, and more intuitively reflect the degree of cooperation between the players.

9.2 Weaknesses

- The evaluation of the parameters on the offensive behavior is relatively objective, but the contribution of each player in the defensive behavior is not well quantified;
- During the behavior analysis, only the pass and duel behaviors were analyzed, while other behaviors were ignored, resulting in a certain deviation of the model;
- The calculation of field scoring rate is idealized, which may lead to the deviation of the model;

References

[1] J. Aguirre, D. Papo, and J. M. Buld. Successful strategies for competing networks. *Nature Physics*, 9(4):230–234.

[2] S. E. Ahnert, D. Garlaschelli, T. M. A. Fink, and G. Caldarelli. Ensemble approach to the analysis of weighted networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 76(1):016101.

[3] C. Reepb. Benjamin. Skill and chance in association football. *Journal of the Royal Statistical Society*, 131(4):581–585.

[4] Martin Lames and Tim Mcgarry. On the search for reliable performance indicators in game sports. *International Journal of Performance Analysis in Sport*, 7(1):62–79, 2007.

[5] Van Mieghem and Piet. Graph spectra for complex networks —— spectra of complex networks. 2010.

A Appendix

```
#! python
import csv
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import matplotlib.gridspec as gridspec
import networkx as nx
import numpy as npy
import numpy as np
import math
import random

# row
MatchID = 0
TeamID = 1
OriginPlayerID = 2
DestinationPlayerID = 3
MatchPeriod = 4
EventTime = 5
EventType = 6
EventSubType = 7
EventOrigin_x = 8
EventOrigin_y = 9
EventDestination_x = 10
EventDestination_y = 11

OneDuelSuccessContribution = 0.1
OneDuelFailureContribution = 0

class Matches:

    class Match:
        def __init__( self , firsh_row ) :
            self.rival = None
            self.MatchID = firsh_row[MatchID]
            self.events = [ firsh_row ]
            if firsh_row[TeamID] != "Huskies" :
                self.rival = firsh_row[TeamID]

        def add( self , row ) :
            if self.rival == None and \
                    row[TeamID] != "Huskies":
                self.rival = row[TeamID]
            self.events.append( row )

    def __init__( self , fileName ) :
        self.matches = {}
        with open( fileName , newline="" ,
                   encoding='UTF-8') as csvfile:
            rows = csv.reader(csvfile)
            for row in list(rows)[1:]:
                row[MatchID ]= int(row[MatchID])
```

```
            try :
                self.matches[ row[
                    MatchID] ].add( row )
            except KeyError:
                self.matches[ row[MatchID] ] \
                    = Matches.Match( row )

    def getOneMatchHuskiesEvents( self ,
                                  matchID:int ):
        return [ event for event in self.matches[
            matchID].events
                 if event[TeamID] == "Huskies" ]
    def getOneMatchOpponentEvents( self ,
                                   matchID:int ):
        return [ event for event in self.matches[
            matchID].events
                 if event[TeamID].find(
                "Opponent") != 0 ]


    def getAllMatchHuskiesEvents( self ):
        events = []
        for match in self.matches.values():
            events.extend(
                self.getOneMatchHuskiesEvents(
                    match.MatchID ) )
        return events
    def getAllMatchOpponentEvents( self ):
        events = []
        for i in self.matches:
            events.extend(
                self.getOneMatchOpponentEvents(
                    i.matchID ) )
        return events

    def getOneMatchPlayerEvents(
            self , matchID:int , playerName:str ):
        return [ event for event in self.matches[
            matchID ].events
                 if playerName == event[
                     OriginPlayerID]
                 or playerName == event[
                     DestinationPlayerID] ]

    def getOneMatchOriginPlayerEvents(
            self , matchID:int , playerName:str ):
        return [ event for event in self.matches[
            matchID ].events
                 if playerName == event[
                     OriginPlayerID] ]

    def getOneMatchDestinationPlayerEvents(
            self , matchID:int ,
            playerName:str ):
```

```
        return [ event for event in
                 self.matches[ matchID ].events
                 if playerName == event[
                     DestinationPlayerID] ]


    def getAllMatchPlayerEvents( self ,
                                 playerName:str ):
        events = []
        for i in self.matches.values():
            events.extend(
                self.getOneMatchPlayerEvents(
                    i.MatchID, playerName) )
        return events

    def getAllMatchOriginPlayerEvents(
            self , playerName:str ):
        events = []
        for i in self.matches.values():
            events.extend(
                self.getOneMatchOriginPlayerEvents(
                    i.MatchID, playerName) )
        return events

    def getAllMatchDestinationPlayerEvents(
            self , playerName:str ):
        events = []
        for i in self.matches.values():
            events.extend(
                self.getOneMatchDestinationPlayerEvents(
                    i.MatchID, playerName) )
        return events

    def getTeamAllPlayerLabels( self ,
                                teamName:str ):
        teamAllPlayerLabel = []
        for match in self.matches.values():
            for event in match.events:
                if event[TeamID] == teamName :
                    try:
                        teamAllPlayerLabel.index(
                            event[OriginPlayerID] )
                    except ValueError:
                        teamAllPlayerLabel.append(
                            event[OriginPlayerID] )
                    if event[DestinationPlayerID] == "":
                        continue
                    try:
                        teamAllPlayerLabel.index(
                            event[DestinationPlayerID] )
                    except ValueError:
                        teamAllPlayerLabel.append(
                            event[DestinationPlayerID] )
        teamAllPlayerLabel.sort()
```

```
        return teamAllPlayerLabel

    def getTeamOneMatchPlayerLabels(
            self , matchID:int ,teamName:str ):
        teamOneMatchPlayerLabel = []
        for event in self.matches[matchID].events:
            if event[TeamID] == teamName:
                try:
                    teamOneMatchPlayerLabel.index(
                        event[OriginPlayerID] )
                except ValueError:
                    teamOneMatchPlayerLabel.append(
                        event[OriginPlayerID] )
                try:
                    teamOneMatchPlayerLabel.index(
                        event[DestinationPlayerID] )
                except ValueError:
                    teamOneMatchPlayerLabel.append(
                        event[DestinationPlayerID] )
        teamOneMatchPlayerLabel.remove( "" )
        return teamOneMatchPlayerLabel
    def playerLabel( self , matchIDbegin,
                     matchIDend ,teamName ):
        playerLabels = []
        for match_id in range(matchIDbegin,matchIDend):
            for event in self.matches[match_id].events:
                if event[TeamID] == teamName:
                    try:
                        playerLabels.index(
                            event[OriginPlayerID] )
                    except ValueError:
                        playerLabels.append(
                            event[OriginPlayerID] )
                    try:
                        playerLabels.index(
                            event[DestinationPlayerID] )
                    except ValueError:
                        playerLabels.append(
                            event[DestinationPlayerID] )
        playerLabels.remove("")
        return playerLabels
    def OneMatchEvents( self , matchID ,teamName ):
            events = []
            for event in self.matches[matchID].events:
                if event[TeamID] == teamName:
                    events.append(event)
            return events

class Filter: # event
    def judge( self , event ):
        return True

class Net:
```

```
    class Node:  #
        def __init__( self , playerLabel:str ):
            self.playerLabel = playerLabel
            self.passSuccess = 0
            self.passFailure = 0

        def addOneTimePassSuccess( self ):
            self.passSuccess += 1

        def addOneTimePassFailure( self ):
            self.passFailure += 1
    class Edge:  #

        def __init__( self ):
            self.passEvents = []
            self.passSuccess = 0
            self.passFailure = 0
        def addOneTimePassFailure( self ):
            self.passFailure += 1
        def addOneTimePassSuccess( self ):
            self.passSuccess += 1
        def addPassEvent( self , event ):
            self.passEvents.append( event )

    def __init__( self , playerLabels:list ,
                  events:list , filter:Filter ):
        self.matrix = [ [ Net.Edge() for j in
                          range(len(playerLabels)) ]
                        for i in range(len(playerLabels)) ]
        self.nodes = {}
        for playerLabel in playerLabels: #                              Ł
            self.nodes[playerLabel] = Net.Node( playerLabel )

        for event in events: #
            if not filter.judge( event ):
                continue
            if event[DestinationPlayerID] == "":  #
                self.nodes[ event[OriginPlayerID]
                ].addOneTimePassFailure()
            else : #           Ł
                self.matrix[ self.__matrixIndex(
                    event[DestinationPlayerID]) ][
                    self.__matrixIndex(event[
                            OriginPlayerID]) ] \
                      .addOneTimePassSuccess()
                self.matrix[ self.__matrixIndex(
                    event[OriginPlayerID]) ][
                    self.__matrixIndex(event[
                            DestinationPlayerID]) ] \
                      .addOneTimePassSuccess()
                self.matrix[ self.__matrixIndex(
                    event[OriginPlayerID]) ][
                    self.__matrixIndex(
                        event[DestinationPlayerID])
```

```
            ].addPassEvent(event)
            self.nodes[ event[OriginPlayerID
            ] ].addOneTimePassSuccess()

def __matrixIndex( self , playerID ):
    return list(self.nodes.keys()).index(
        playerID )

def graph( self ):

    G = nx.Graph()
    WW = nx.Graph()
    all_pass_time = 0
    max_pass_time = 0
    labels = {}

    for node in self.nodes.values():
        print(node.playerLabel)
        try :
            node_ = node.playerLabel.split(
                "_")[1]
        except IndexError:
            print( node.playerLabel )
        if max_pass_time < node.passSuccess\
                +node.passFailure:
            max_pass_time = node.passSuccess\
                            +node.passFailure
            all_pass_time += node.passSuccess
            all_pass_time += node.passFailure
        print(node.passSuccess)
        G.add_node( node_ , passSuccess =
        node.passSuccess
                    ,  passFailure = node.passFailure )
        labels[node_] = node_

    pos = nx.spring_layout(G)  # positions for all node
    print(pos)
    nx.draw_networkx_labels(G,pos,labels)

    for node in G.nodes( data=True ) :
        print(node)
        nx.draw_networkx_nodes( WW, pos,
                    nodelist=[node[0]],
                    node_size = 5000*((
            node[1]['passSuccess']
                    +node[1]['passFailure'] )
                        /max_pass_time)**2 ,
                    node_color='y',
                    node_shape='o',
                    alpha=1.   )
        nx.draw_networkx_nodes( WW, pos,
                    nodelist=[node[0]],
                    node_size = 5000*(
                    node[1]['passFailure']
```

```
                            /max_pass_time)**2 ,
                        node_color='r',
                        node_shape='o',
                        alpha=1.0 )

        all_pass_time = 0
        max_pass_time = 0

        for edge in G.edges(data=True):
            if edge[2]["weight"] > 0 :
                nx.draw_networkx_edges( WW, pos,
                                        edgelist = [ edge ],
                                        width =  1+20*(
                edge[2]["weight"])/max_pass_time ,
                                        edge_color = 'b',
                                        style = 'solid',
                                        alpha =  (
                    edge[2]["weight"])/max_pass_time ,
                                    )

        # plt.axis('off')
        nx.draw(WW)
        plt.show()

class Bar:
    def __init__( self ):
        pass
    def graph( self ):
        pass

def all( data:Matches , begin , end , teamName ):

    playerLabel = data.playerLabel( begin , end ,
                                    teamName )
    others = len(playerLabel) - 1
    matrix_2 = npy.zeros( (len(playerLabel),len(
        playerLabel)) )
    matrix_3 = npy.zeros( ( len(playerLabel),
                            len(playerLabel)*len(
                                playerLabel) ) )

    for i in range(begin , end) :
        events = data.OneMatchEvents(  i , teamName )
        in_combo = False
        last_relay = None
        combo_events = []

        def combo_process( events:list ):
            #         Ł
            def allLabels( events:list ):
                #
                labels= []
                for event in events:
                    try:
```

```
                labels.index(
                    event[OriginPlayerID] )
            except ValueError:
                labels.append(
                    event[OriginPlayerID] )
            try:
                labels.index(
                    event[DestinationPlayerID] )
            except ValueError:
                labels.append(
                    event[DestinationPlayerID] )
        return labels
    def dis( xy:list ): #
        return math.sqrt( (
        float(xy[0])-float(xy[2]) )**2
        + (float(xy[1])-float(xy[3]))**2 )

    if len(events) == 1 :
        for event in events :
            # print(event)
            if event[OriginPlayerID] == \
                    event[
                DestinationPlayerID] :
                continue
            A = playerLabel.index(
                event[OriginPlayerID] )
            B = playerLabel.index(
                event[DestinationPlayerID])
            matrix_2[ A ][ B ] += 1
            #dis( event[EventOrigin_x:] )
            matrix_2[ B ][ A ] += 1
            #dis( event[EventOrigin_x:] )
    elif len(events) > 1 :
        labels = allLabels( events )
        gain_coeffcient = 1+0.3*len(events)
        for label in labels :
            for event in events :
                if label == event[
                    OriginPlayerID] or \
                        label == \
                event[DestinationPlayerID] :
                    continue
                A = playerLabel.index(label)
                B = playerLabel.index(
                    event[DestinationPlayerID])
                C = playerLabel.index(
                    event[OriginPlayerID])
                matrix_3[ A ][ B*len(playerLabel) + C ] \
                    += gain_coeffcient
                #*dis( event[EventOrigin_x:] )
                matrix_3[ A ][ C*len(playerLabel) + B ] \
                    += gain_coeffcient
                #*dis( event[EventOrigin_x:] )
                matrix_3[ B ][ C*len(playerLabel) + A ] \
```

```
                        += gain_coeffcient
                    #*dis( event[EventOrigin_x:] )
                    matrix_3[ B ][ A*len(playerLabel) + C ] \
                        += gain_coeffcient
                    #*dis( event[EventOrigin_x:] )
                    matrix_3[ C ][ A*len(playerLabel) + B ] \
                        += gain_coeffcient
                    #*dis( event[EventOrigin_x:] )
                    matrix_3[ C ][ B*len(playerLabel) + A ] \
                        += gain_coeffcient
                    #*dis( event[EventOrigin_x:] )

        for event in events:
            if event[EventType] == "Pass" \
                    and in_combo :
                if event[OriginPlayerID] == \
                        last_relay and \
                        event[DestinationPlayerID] != "":
                    last_relay = event[
                        DestinationPlayerID]
                    combo_events.append(
                        event )
                else:
                    combo_process( combo_events )
                    in_combo = False
                    last_relay = None
                    combo_events = []
            elif event[EventType] == "Pass" \
                    and not in_combo \
                    and event[DestinationPlayerID] != "" :
                in_combo = True
                combo_events.append( event )
                last_relay = event[DestinationPlayerID]
            elif event[EventType] != "Pass" \
                    and in_combo :
                combo_process( combo_events )
                in_combo = False
                last_relay = None
                combo_events = []

    return matrix_2, matrix_3, playerLabel

def bar_( x:list , y:list ): # !
    plt.bar(x, y, label="label1", color='red')
    plt.xticks(np.arange(len(x)), x, rotation=0,
               fontsize=10)
    #
                                                        2 7 0

    plt.legend(loc="upper left")  #
                                                  l a b e l

    plt.rcParams['font.sans-serif'] = ['SimHei']  #
```

```
    plt.ylabel('      ')
    plt.xlabel('name')
    plt.rcParams['savefig.dpi'] = 300  #
    plt.rcParams['figure.dpi'] = 300  #
    plt.rcParams['figure.figsize'] = (15.0, 8.0)  #
    plt.title("")
    # plt.savefig('D:\\result.png')
    plt.show()

def playerCapacity( data:Matches , teamName:str ):

    playerLabels = data.getTeamAllPlayerLabels(teamName)

    duel_attack_success = [ 0 for i
                            in range(len(playerLabels))]
    duel_attack_failure = [ 0 for i
                            in range(len(playerLabels))]
    duel_deffend_success = [ 0 for i
                             in range(len(playerLabels))]
    duel_deffend_failure = [ 0 for i
                             in range(len(playerLabels))]
    pass_success = [ 0 for i in range(len(playerLabels))]
    pass_failure = [ 0 for i in range(len(playerLabels))]

    for match in data.matches.values():
        for i in range(len(match.events)-2):
            # if i+2 == len(match.events):
            #     continue
            event = match.events[i]
            next_event_ = match.events[i+1]
            next_event = match.events[i+2]

            if event[OriginPlayerID].split("_")[0] == teamName:
                if event[EventType] == "Duel": # duel

                    # defending duel
                    if event[EventSubType] ==\
                            "Ground defending duel" \
                            and next_event_[EventSubType] \
                            == "Ground attacking duel":
                        if next_event[
                            OriginPlayerID].split("_")[0] \
                                == teamName: # defending success
                            duel_deffend_success[
                                playerLabels.index(
                                event[OriginPlayerID]) ] += 1
                        else:
                            duel_deffend_failure[
                                playerLabels.index(
                                    event[OriginPlayerID]) ] -= 1
                    elif event[EventSubType] == \
                            "Ground attacking duel" \
                            and next_event_[EventSubType] \
                            == "Ground defending duel" :
```

```
                    # Ground attacking duel
                    if next_event[OriginPlayerID].split(
                            "_")[0] \
                            == teamName:# attacking success
                        duel_attack_success[
                            playerLabels.index(
                            event[OriginPlayerID]) ] += 1
                    else: # attacking failure
                        duel_attack_failure[
                            playerLabels.index(
                            event[OriginPlayerID]) ] -= 1
            elif event[EventType] == "Pass":
                if event[DestinationPlayerID] \
                        == "": # pass failure
                    pass_failure[playerLabels.index(
                        event[OriginPlayerID])] -= 1
                else: # pass success
                    pass_success[playerLabels.index(
                        event[OriginPlayerID])] += 1

size = len(playerLabels)

x = np.arange(size)
total_width, n = 0.8, 3
width = total_width / n
x = x - (total_width - width) / 2

plt.bar(x, duel_attack_success,
        width=width, label='duel_attack_success',
        color='darkgoldenrod')
plt.bar(x, duel_attack_failure,
        width=width, label='duel_attack_failure',
        color='orange')
plt.bar(x + width, duel_deffend_success,
        width=width, label='duel_defend_success',
        color='teal')
plt.bar(x + width, duel_deffend_failure,
        width=width, label='duel_defend_failure',
        color='cyan')
plt.bar(x + 2*width, pass_success,
        width=width, label='pass_success',
        color='lime')
plt.bar(x + 2*width, pass_failure,
        width=width, label='pass_failure',
        color='g')

plt.xticks( x , [ i.split("_")[1]
                  for i in playerLabels] )
plt.legend(loc="upper left")  #
                                        l a b e l

plt.rcParams['font.sans-serif'] \
    = ['SimHei']  #
plt.ylabel('times')
```

```
    plt.xlabel('playerID')
    plt.rcParams['savefig.dpi'] = 500  #
    plt.rcParams['figure.dpi'] = 900  #
    plt.rcParams['figure.figsize'] = (25.0, 16.0)  #       /
    plt.title("playerEventStatistic")
    plt.savefig(".")
    plt.show()

def matrix_2_fig( matrix_2:np.ndarray , playerLabel:list ):

    fig, ax = plt.subplots(figsize=(4, 4),
                           constrained_layout=True)
    norm = mcolors.Normalize(vmin=0.,
                             vmax=matrix_2.max()  )
    pc_kwargs = {'rasterized': True,
                 'cmap': 'viridis', 'norm': norm}
    im = ax.pcolormesh(matrix_2, **pc_kwargs)
    fig.colorbar(im, ax=ax, shrink=0.6)

    plt.figtext( 0.885, 0.2, "Times")

    plt.rcParams['savefig.dpi'] = 1400  #
    plt.rcParams['figure.dpi'] = 1400  #

    plt.show()

def max_3_( matrix_3:npy.ndarray ):
    playerLabel = data.getTeamAllPlayerLabels( "Huskies" )
    n = len(playerLabel)
    s = []
    for i in range(len(matrix_3)):
        for j in range(i):
            if matrix_3[i][j]>1:
                s.append( ( playerLabel[i] ,
                            playerLabel[j//n] ,
                            playerLabel[j%n] ,
                            matrix_3[i][j] ) )
    s.sort(key=lambda x:x[3])
    return s

class Contribution:
    class Shot:
        def __init__( self , matchID:int ):
            self.basicEvents = []
            self.basicEventsWinOrFail = []
            self.basicEventsType = []
            self.matchID = matchID
            self.contribution = 0
        def addBasicEvent( self, winOrFail:bool ,
                           basicEvent:list , type_:str ):
            self.basicEvents.append(basicEvent)
            self.basicEventsWinOrFail.append(winOrFail)
            self.basicEventsType.append(type_)
    def __init__( self, data:Matches , begin:int ,
```

```
          end:int , teamName:str ):
self.shots = []
self.playerLabels = data.playerLabel( begin,
                                      end, teamName )
self.playerLabels.sort()
self.playerContribution = npy.zeros( (
    len(self.playerLabels),2) )
self.playerContribution_coeffi = npy.zeros( (
    len(self.playerLabels),2) )
self.passContribution = npy.zeros( ( len(
    self.playerLabels),len(self.playerLabels) ) )
self.comeupTimes = npy.zeros( (len(
    self.playerLabels),end-begin) )
self.teamName = teamName

match_index = 0
for match in data.matches.values():
    for event in match.events:
        try:
            self.comeupTimes[
                self.playerLabels.index(
                    event[OriginPlayerID]) ][
                match_index] = 1
            self.comeupTimes[
                self.playerLabels.index(
                    event[OriginPlayerID]) ][
                match_index] = 1
        except ValueError:
            pass
        if event[TeamID]==teamName \
                and event[EventType]=="Pass" :
            index = self.playerLabels.index(
                event[OriginPlayerID])
            if event[DestinationPlayerID] != "":
                self.playerContribution_coeffi[index][0] += 1
            self.playerContribution_coeffi[index][1] += 1
self.comeupTimes = np.array( [ i.sum()
        for i in self.comeupTimes ] ) / (end-begin)
self.playerContribution_coeffi = \
    self.playerContribution_coeffi[:,0]/\
    self.playerContribution_coeffi[:,1]

for i in range(begin,end):
    shot = Contribution.Shot( i )
    events = data.matches[i].events
    index = 0
    relayEvents = []
    j = 0
    for index in range(len(events)):
        if j>0:
            j -= 1
            continue
        if events[index][EventType] == "Pass" \
                and len(relayEvents)==0 and \
```

```
            events[index][TeamID] == teamName:
        #     r e l a y
        relayEvents.append( events[index] )
    elif events[index][EventType] == "Pass" \
            and len(relayEvents)>0 and \
            events[index][TeamID] == teamName:
        #   r e l a y
        relayEvents.append( events[index] )
    elif events[index][EventType] != "Pass" \
            and len(relayEvents)>0 : #  r e l a y
        if len(relayEvents) == 1:
            if events[index][EventType]=="Duel" \
                    and events[index+1][EventType]\
                    =="Duel" :
                if events[index+2][
                    OriginPlayerID].split("_")[0]\
                        == teamName :
                    j = 1
                    relayEvents.extend([
                        events[index],events[index+1]])
                    shot.addBasicEvent(
                        True , relayEvents ,
                        "SingelPass+Duel" )
                    relayEvents = []
                else :
                    j = 1
                    relayEvents.extend([
                        events[index],
                        events[index+1]])
                    shot.addBasicEvent(
                        False ,
                        relayEvents ,
                        "SingelPass+Duel" )
                    relayEvents = []
            else :
                if events[index][OriginPlayerID].split(
                        "_")[0]==teamName :
                    shot.addBasicEvent(
                        True , relayEvents , "SingelPass" )
                    relayEvents = []
                else :
                    shot.addBasicEvent(
                        False , relayEvents ,
                        "SingelPass" )
                    relayEvents = []

        if events[index][EventType]==\
                "Duel" and events[index+1][
            EventType]=="Duel" :
            if events[index+2][
                OriginPlayerID].split("_")[0] == teamName :
                j = 1
                relayEvents.extend([
                    events[index],events[index+1]])
```

```
                        shot.addBasicEvent(
                            True , relayEvents , "relayPass+Duel" )
                        relayEvents = []
                    else :
                        j = 1
                        relayEvents.extend([
                            events[index],events[index+1]])
                        shot.addBasicEvent(
                            False , relayEvents , "relayPass+Duel" )
                        relayEvents = []
                else :
                    if events[index][OriginPlayerID].split(
                            "_")[0]==teamName :
                        shot.addBasicEvent(
                            True , relayEvents , "relayPass" )
                        relayEvents = []
                    else :
                        shot.addBasicEvent(
                            False , relayEvents , "relayPass" )
                        relayEvents = []
            elif events[index][EventType]==\
                    "Duel" and len(relayEvents)==0 :
                if index+2 == len(events):
                    break
                if events[index+1][EventType]==\
                        "Duel" and events[index+2][
                    OriginPlayerID].split(
                    "_")[0]==teamName:
                    shot.addBasicEvent( True ,
                                        [events[index+1],
                                         events[index]] , "Duel" )
                    j=1
                elif events[index+1][EventType]=="Duel" \
                        and events[index+2][OriginPlayerID].split(
                    "_")[0]!=teamName:
                    shot.addBasicEvent( False ,
                    [events[index+1],events[index]] , "Duel" )
                    j=1
                else:
                    continue
            elif events[index][EventType] == "Shot" :
                if len(relayEvents)>0 :
                    relayEvents.append(events[index])
                    shot.addBasicEvent( True ,
                                        relayEvents , "Shot_" )
                relayEvents = []
                self.shots.append( shot )
                shot = Contribution.Shot(i)
            pass
        pass
    pass
def calContribution( self ):
    for shot in self.shots:
        for i in zip( shot.basicEvents,
```

```
            shot.basicEventsWinOrFail,
            shot.basicEventsType ) :
    def likehood( x1,y1, co1 , x2,y2,co2):
        def hzl(i, j, k):  #  k                  Ł
            distanceLeft = np.sqrt((j - 100) *
                                   (j - 100)
                + (i - 44.62) * (i - 44.62))
            distanceRight = np.sqrt((j - 100) *
                                    (j - 100)
                + (i - 55.38) * (i - 55.38))
            distance = np.sqrt((j - 100) *
                               (j - 100)
                + (i - 50) * (i - 50))
            cos = (distanceLeft *
                   distanceLeft + distanceRight *
                   distanceRight -
                   10.76 * 10.76) / (2 * \
                    distanceLeft *
                    distanceRight )
            sinHalf = np.sqrt((1 - cos) / 2)
            if np.isnan(sinHalf):
                sinHalf = 0
            if distance == 0:
                img[i][j] = 255
            distanceX = i
            distanceY = abs(j - 50)
            return ((1-k) * (1 - distance /
                             111.8 + sinHalf)
                    / 2 + k * math.sin( \
                          ((1 - distance /
                            111.8 + sinHalf)
                           / 2) * math.pi / 2))
        x1 = int(x1)
        x2 = int(x2)
        y1 = int(y1)
        y2 = int(y2)
        sub = (hzl(x2,y2,co2) - hzl(x1,y1,co1))
        return  sub if sub > 0 else 0
    def calSingelPass( events:list , winOrFail:bool ):
        if winOrFail:
            sum_ = 0
            for event in events:
                OplayerIndex = self.playerLabels.index(
                    event[OriginPlayerID])
                if event[DestinationPlayerID] == "":
                    return 0
                coeffi_1 = self.playerContribution_coeffi[
                    self.playerLabels.index(
                        event[OriginPlayerID]) ]
                coeffi_2 = self.playerContribution_coeffi[
                    self.playerLabels.index(
                        event[DestinationPlayerID]) ]

                DplayerIndex = self.playerLabels.index(
```

```
                    event[DestinationPlayerID])
                likehood_ = likehood(
                    event[EventOrigin_x] ,
                    event[EventOrigin_y] ,coeffi_1 , \
                    event[EventDestination_x],
                    event[EventDestination_y] , coeffi_2 )
                self.playerContribution[OplayerIndex, 1] \
                    += likehood_
                self.passContribution[OplayerIndex,
                                      DplayerIndex] \
                    += likehood_
                sum_ += likehood_
            return sum_
        else:
            return 0
    def calSingelPass_Duel( events:list , winOrFail:bool ):
        if winOrFail:
            sum_ = 0
            for index_ in range(len(events)):
                event = events[index_]
                OplayerIndex = self.playerLabels.index(
                    events[0][OriginPlayerID])
                if events[0][EventType] == "Pass" \
                        and events[0][
                    DestinationPlayerID] == \
                        "":
                    pass
                elif events[0][EventType] == "Pass" \
                        and events[0][
                    DestinationPlayerID].split(
                    "_")[0] == self.   teamName:
                    DplayerIndex = \
                        self.playerLabels.index(
                        events[0][DestinationPlayerID])
                    coeffi_1 = self.playerContribution_coeffi
                                                        [
                        self.playerLabels.index(
                            event[OriginPlayerID]) ]
                    coeffi_2 = self.playerContribution_coeffi
                                                        [
                        self.playerLabels.index(event[
                            DestinationPlayerID]) ]
                    likehood_ = likehood( event[
                            EventOrigin_x] , event[
                                                                EventOrigin_
                                                                ]

                                                                ,

                                                coeffi_1,
                                    event[
                                EventDestination_x],
                                event[EventDestination_y] , coeffi_2
                                                            )
```

```
                    self.playerContribution[OplayerIndex,
                                            1] += likehood_
                    self.passContribution[OplayerIndex,
                                          DplayerIndex] +=


                    sum_ += likehood_
                elif events[1][EventType] == "Duel" :
                    if self.playerLabels[OplayerIndex].split(
                            "_")[0] == self.teamName :
                        self.playerContribution[OplayerIndex,

                                                                0
                                                                ]

                                                                \

                                += OneDuelSuccessContribution
                    else :
                        i_ = self.playerLabels.index(events[
                            -1][OriginPlayerID])
                        self.playerContribution[ i_ , 0] \
                        += OneDuelSuccessContribution
                    sum_ += OneDuelSuccessContribution
                return sum_
        else:
            return 0
    def calRelayPass( events:list , winOrFail:bool ):
        if winOrFail:
            sum_ = 0
            for event in events:
                OplayerIndex = self.playerLabels.index(
                    event[OriginPlayerID])
                if event[DestinationPlayerID].split("_")[0] \
                        == self.teamName:
                    DplayerIndex = self.playerLabels.index(
                        event[DestinationPlayerID])
                else:
                    continue
                coeffi_1 = self.playerContribution_coeffi[
                    self.playerLabels.index(event[
                            OriginPlayerID]) ]
                coeffi_2 = self.playerContribution_coeffi[
                    self.playerLabels.index(event[
                            DestinationPlayerID]) ]
                likehood_ = likehood( event[EventOrigin_x]
                                      , event[EventOrigin_y]

                                                                ,


                                      coeffi_1,
                                    event[EventDestination_x],
                                    event[EventDestination_y]
                                      , coeffi_2)
```

```
                    self.playerContribution[OplayerIndex, 1] \
                        += likehood_
                    self.passContribution[OplayerIndex,
                                                    DplayerIndex
                                                    ] \
                        += likehood_
                    sum_ += likehood_
                return sum_
            else:
                return 0
        def calRelayPass_Duel( events:list , winOrFail:bool ):
            if winOrFail:
                sum_ = 0
                for index_ in range(len(events)):
                    event = events[index_]
                    if event[EventType] == "Pass" and event[
                        DestinationPlayerID]!="" :
                        OplayerIndex = self.playerLabels.index(
                            event[OriginPlayerID])
                        DplayerIndex = self.playerLabels.index(
                            event[DestinationPlayerID])
                        coeffi_1 = self.playerContribution_coeffi
                                                            [
                            self.playerLabels.index(event[
                                OriginPlayerID]) ]
                        coeffi_2 = self.playerContribution_coeffi
                                                            [
                            self.playerLabels.index(event[
                                DestinationPlayerID]) ]
                        likehood_ = likehood( event[EventOrigin_x
                                                            ]
                                             , event[
                            EventOrigin_y] , coeffi_1,
                                    event[EventDestination_x],
                                             event[
                                    EventDestination_y] ,
                                                                coeffi_
                                                                )

                        self.playerContribution[OplayerIndex, 1]
                                                           +=
                                                           likehood_

                        self.passContribution[OplayerIndex,
                                              DplayerIndex] +=


                        sum_ += likehood_
                    elif event[EventType] == "Duel" :
                        if event[OriginPlayerID].split(
                                "_")[0] == self.teamName :
                            i_ = self.playerLabels.index(
                                event[OriginPlayerID] )
                        elif events[-1][OriginPlayerID].split(
```

```
                                    "_")[0] == self.teamName :
                                i_ = self.playerLabels.index(
                                    events[-1][OriginPlayerID])
                            else :
                                return 0
                            self.playerContribution[i_, 0] += \
                                OneDuelSuccessContribution
                            sum_ += OneDuelSuccessContribution
                            break
                    return sum_
                else:
                    return 0
            def calDuel( events:list , winOrFail:bool ):
                if winOrFail:
                    if events[-1][OriginPlayerID].split(
                            "_")[0] == self.teamName :
                        i_ = self.playerLabels.index(
                            events[-1][OriginPlayerID])
                    elif events[0][OriginPlayerID].split(
                            "_")[0] == self.teamName:
                        i_ = self.playerLabels.index(
                            events[0][OriginPlayerID])
                    else:
                        return 0
                    self.playerContribution[ i_ , 0] \
                        += OneDuelSuccessContribution
                    return OneDuelSuccessContribution
                else:
                    return 0
            basicEvent = i[0]
            winOrFail = i[1]
            type_ = i[2]
            if type_=="SingelPass":
                oneBasicEventContribution = \
                    calSingelPass( basicEvent , winOrFail )
            elif type_=="SingelPass+Duel":
                oneBasicEventContribution = \
                    calSingelPass_Duel( basicEvent , winOrFail )
            elif type_=="relayPass":
                oneBasicEventContribution = \
                    calRelayPass( basicEvent , winOrFail )
            elif type_=="relayPass+Duel":
                oneBasicEventContribution = \
                    calRelayPass_Duel( basicEvent , winOrFail )
            elif type_=="Duel":
                oneBasicEventContribution = \
                    calDuel( basicEvent , winOrFail )
            # print( type_ ," : ",winOrFail ," : ",
            # oneBasicEventContribution)
            # for i in basicEvent:
            #     print(i)
            # print()
            shot.contribution += oneBasicEventContribution
    print(self.comeupTimes)
```

```
        self.playerContribution[:,0] = \
            self.playerContribution[:,0] * self.comeupTimes
        self.playerContribution[:,1] = \
            self.playerContribution[:,1] * self.comeupTimes
    def check( self, index:int ):

        basicEvents = len(self.shots[index].basicEvents)
        # for i in zip( self.shots[index].basicEvents ,
        # self.shots[index].basicEventsType ,
        # self.shots[index].basicEventsWinOrFail ):
        #     print()
        #     print(i[1],end=":")
        #     if i[2]:
        #         print("Win")
        #     else :
        #         print("Fail")
        #     for event in i[0]:
        #         print(event)
        for i in zip( self.playerLabels,
                      self.playerContribution,
                      self.playerContribution_coeffi):
            print(i[0],end="  ")
            print(i[1])
            print()
        fig, ax = plt.subplots(figsize=(4, 4),
                               constrained_layout=True)
        norm = mcolors.Normalize(vmin=0.,
                                 vmax=self.passContribution.max()  )
        pc_kwargs = {'rasterized': True,
                     'cmap': 'viridis', 'norm': norm}
        im = ax.pcolormesh(self.passContribution, **pc_kwargs)
        fig.colorbar( im, ax=ax, shrink=0.6 )

        plt.figtext( 0.885, 0.2, "")

        plt.rcParams['savefig.dpi'] = 1400  #
        plt.rcParams['figure.dpi'] = 1400  #

        plt.show()
        print()
    def graph( self ) :

        self.playerLabels
        self.passContribution
        self.playerContribution

        G = nx.Graph()
        WW = nx.Graph()
        all_pass_contribution = 0
        max_pass_contribution = 0
        labels = {}

        for label in self.playerLabels:
            print(label)
```

```
            duel = self.playerContribution[
                self.playerLabels.index(label), 0]
            pass_ = self.playerContribution[
                self.playerLabels.index(label), 1]
            if max_pass_contribution < pass_:
                max_pass_contribution = pass_
            all_pass_contribution += pass_
            G.add_node( label.split("_")[1] ,
                        pass_=pass_ , duel=duel )
            labels[label.split("_")[1]] = label.split("_")[1]

        pos = nx.spring_layout(G)  # positions for all node
        nx.draw_networkx_labels(G,pos,labels)


        for node in G.nodes( data=True ) :
            print(node)
            nx.draw_networkx_nodes( WW, pos,
                        nodelist=[node[0]],
                        node_size = 1500*(( node[1][
                        'pass_']+node[1]['duel'] )/
                            max_pass_contribution)**2 ,
                        node_color='y',
                        node_shape='o',
                        alpha=1.0   )
            nx.draw_networkx_nodes( WW, pos,
                        nodelist=[node[0]],
                        node_size = 1500*(node[1][
                            'pass_']/max_pass_contribution)**2 ,
                        node_color='r',
                        node_shape='o',
                        alpha=1.0 )

        all_pass_time = 0
        max_pass_time = 0

        for sender in self.playerLabels:
            for receiver in self.playerLabels:
                if sender != receiver :
                    sender_ = sender.split("_")[1]
                    receiver_ = receiver.split("_")[1]
                    if ( sender_ , receiver_ ) not in G.edges():
                        weight_ = self.passContribution[
    self.playerLabels.index(sender) ][
            self.playerLabels.index(receiver) ]
                        all_pass_time += weight_
                        if max_pass_time < weight_ :
                            max_pass_time = weight_
                        G.add_edge( sender_ ,
            receiver_ , weight=weight_)

        for edge in G.edges(data=True):
            if edge[2]["weight"] > 0 :
                nx.draw_networkx_edges( WW, pos,
```

```
                                    edgelist = [ edge ],
                                    width =  1+18*((
                edge[2]["weight"]))/max_pass_time ,
                                    edge_color = 'b',
                                    style = 'solid',
                                    alpha =  (edge[2][
                "weight"])/max_pass_time ,
                                   )


        # plt.axis('off')
        nx.draw(WW)
        plt.show()
data = Matches("fullevents.csv")

# for i in data.getTeamAllPlayerLabels("Huskies"):
#     print(i)

# net = Net( data.getTeamAllPlayerLabels("Huskies") ,
# data.getAllMatchHuskiesEvents() , Filter() )
# net.graph()

# matrix_2, matrix_3 , playerLabel =
# all(data,1,39,"Huskies")
# bar_( [ i.split("_")[1] for i in
# data.playerLabel(1,7,"Huskies")] ,matrix_2[:,6] )

# matrix_2_fig( matrix_2 , playerLabel )

# playerCapacity(data,"Huskies")

# a, b = np.linalg.eig(matrix_2)
# a.sort()
# print( a )

# print(a)
# print()
# print(b)

contribution = Contribution(data,1,38,"Huskies")
contribution.calContribution()
contribution.graph()

# contribution.calContribution()

# print(contribution.playerContribution)
# print(contribution.passContribution)

# contribution.check(1)

# dddd = {}
# for match in data.matches.values():
#     for event in match.events:
#         if event[TeamID]=="Huskies" and
```

```
#          event[EventType] == "Shot":
#              try:
#                  dddd[event[OriginPlayerID]] += 1
#              except:
#                  dddd[event[OriginPlayerID]] = 1

# for i in dddd.keys():
#     print( i , "   ",  dddd[i] )
```