The Best Defense is a Good Offense ?

Offensive Strategy Success Analysis Using Network Characteristics in FIFAWorld Cup 2018

Syeda Narmeen ec19531@qmul.ac.uk Problem formulation, Algorithm, Coding ,Graph drawing,Report writing. 25%	Amina Quraishi ec19374@qmul.ac.uk Data acquisition and pre-processing, Literature Review, Related work, Report writing. 25%	Arvind Menon a.menon@se19.qmul.ac.uk Concept and approach formulation , Match Analysis, Report writing, Literature review, Validation 25%	Maneesha Majeed ec18592@qmul.ac.uk Results and Findings,Network comparison,Tabulating results, Data Analysis, Report writing Literature Review 25%

Abstract

'The Best Defense is a Good Offense' is a motto with which many sports fans align themselves.

The present study's main purpose was to test the ubiquitous belief that offensive football strategies as opposed to defensive football strategies result in successful outcomes, using the most current datasets of the World Cup 2018 competition. Various methods have been applied to football analysis in previous research, and this work contributes a holistic approach, using a novel score-based technique, which has been coined the 'LucaBall' score, to compare the network metrics of the attacking sections in isolation of both home and away teams calculated across all 55 non-draw matches. The calculations of the LucaBall score involved five network metrics: average node degree, algebraic connectivity, clustering coefficient, density and global reaching centrality. To some extent, the LucaBall score successfully illustrated the synchronicity between the teams that won their matches and the offensive strategy they used, which supports the hypothesis; indeed, the best defense is a good offense.

***keywords*:Football, passing network, graphs, FIFA World Cup, network analysis and network metrics.**

I. Introduction

Graph theory constitutes a type of computer science genre called 'network theory,' in which nodes (individuals, places, things) are connected by edges. Data science and analytics uses network theory as a tool to visualize and explain structures and problems; specifically, within the field of sports analysis, studies have applied network theory to sports such as football, in order to assess passing networks [3], [4], and football styles [2]. These types of studies are relatively common, whereas the availability of a new, unique and comprehensive dataset in recent months have opened possibilities and opportunities to conduct more complex systems analysis such as the dynamics of player interactions [1]. It is through the use of this dataset primarily and another dataset secondarily that this paper's analysis focuses upon.

Problem Formulation

Many data scientists, psychologists, and game theorists have conducted football game analysis, especially due to the widely available data science tools and now, detailed and comprehensive data. In the following Related Work section, these numerous studies are discussed, and the reader will discover that most of them isolate network metrics and draw conclusions based on each one separately. Some research on success prediction conducts descriptive and inferential statistics such as correlation and hypothesis testing in addition to, or in isolation. The inspiration for this paper's hypothesis originated from a study in which Brazilian and Italian soccer styles were analyzed in order to prove or disprove the same hypothesis, 'Is the best defence a good offense?'[2] The difference between this research and the former's is two-fold. Firstly, this analysis is conducted over the whole subset of non-draw matches for the participating countries in 2018 FIFA World Cup, not only between two countries' leagues as in the former study. Secondly, a novel comparative methodology was created and implemented which was coined the LucaBall score, rather than the former's descriptive and inferential statistics methodology. Thus, the authors of this paper intend to offer to the scientific world of sports, namely, football analysis a unique and interesting contribution.

II. Related work

A. Team Success and Various Methodologies

Among the methods used to explore the possible factors influencing the success of a football team, the play-by-play analysis in one study measured the statistical significance between successful and unsuccessful positions and their plays, focusing on flow centrality and flow betweenness. It found that successful plays are reflective of the level of offensiveness in the playing position on the pitch [16].

Another study investigated the success of a team based on network density, centralization, and clustering coefficient and found that low densities translated into predictability of a win, while high clustering coefficients and low centralization were less significant indicators of a win. [17] In addition, team performance increased with high connectivity [23]. Likewise, this study also employed the same network metrics (density, centrality, clustering coefficient) with similar results; one exception is that in the present study, the majority of the winning teams reflected higher densities, rather than lower ones. Two additional metrics included in the LucaBall score were average node degree and a specific form of connectivity, the algebraic connectivity. The three metrics that were inspired by Pappalardo et al.'s paper, describing "the largest open collection of soccer-logs ever released [1]" were density, algebraic connectivity and global reaching centrality.

These soccer-logs enable researchers to reconstruct a team's passing network, defined as "a representation of the movements of the ball between teammates during a match" in which further analysis may be conducted [1].

Pappalardo et al. employed two indices, an acceleration and an invasion index, in which they found that within their example match between Roma and Fiorentina, Fiorentina proved successful by spending more time near the opponent's goal (invasion index) and by rushing to the opponent's field faster than Roma (acceleration index) [1]. Kawasaki et al.'s research also employed the use of pass position to investigate successful plays [24]. Despite the fact that the authors of this paper chose not to duplicate this methodology, in lieu of a novel scoring methodology, the LucaBall score, in essence, the results also support the offensive strategies which may predict the winning team.

In addition to these methodologies to analyze offensive and defensive strategies in football matches, a number of researchers have implemented their analysis based on goals scored and goals conceded [19-22]. The authors of this paper initially planned on including these metrics within the LucaBall score, but chose to reject that idea, as these simplified metrics' success prediction capability has been disproved by researchers Filho et al. in a more recent paper published in 2013 [2]. Filho et al. found that both of the winning teams that were analyzed had balanced defensive strategies, while Brazilian league's teams' offensive strategy was definitely related to its success [2]. Irrespective of the differences in methodology between the two papers aforementioned, descriptive/inferential statistics versus the present study's novel methodology, the LucaBall score, the results mirror each other; offensive strategies are the most effective in winning a match.

The various methodologies in the above discussion of Related Work bears little significance as to the present study's aim to prove the hypothesis of offensive strategies reflecting team success. All related work supports the hypothesis, thus, the authors desire to demonstrate a novel and unique methodology, the LucaBall score, to arrive at the same conclusion.

B. PageRank (extended work)

The PageRank centrality is one of the fundamental measures in element-level analysis of a network element, i.e. a node or a link, which indicates how significant this node is in the network. A fundamental idea of PageRank is that links from an "important" node must weigh more than links from nodes with less "important" ones.

Other than the well-known algorithm of rating web-pages [2] in which the PageRank value is calculated based on an analysis of the links between web pages in a web graph, other applications of PageRank have emerged. In recent years, data mining in online social networks has become a trendy research topic [3, 4, 5]. Resources and content of social networks are generated by users, reflecting an increasingly rich social life and spirit of human society. Therefore, ranking objects in social networks (people, content, etc) has also become important.It's likewise utilized in tasks, for example,link prediction, data dispersion and network recognition. What's more, it is utilized in Natural Language Processing (NLP) with the end goal of text summarization and word sense disambiguation[25].

Relevance of Pagerank in Football Network

As mentioned, PageRank centrality is a recursive notion of popularity when a node is referred to by another popular node in the network. A PageRank algorithm measures a player's popularity, as judged by the number of passes he receives from other popular players. It gives a rough idea of who is most likely to end up with the ball after a suitably large number of passes. Since the PageRank score of a player depends on the score of all his teammates, all PageRank scores in a team must be computed at the same time.

PageRank centrality roughly assigns to each player the likelihood that he will have the ball after a reasonable number of passes has been made. If additional precision is required for this measurement, the probability can be replaced by player-dependent probabilities, which would make more sense if certain players are more prone to keep the ball than others. In either case, the value of probability does not come from the network alone, as it may, in general, be very different from one team to another, and ought to be controlled by heuristics.

III. Datasets and Approach

A. Datasets

For this paper we are using two datasets. The first dataset has been collected and provided by a football company named Wyscout. This collection of datasets had been used during the Soccer Data Challenge initiative and is "the largest collection of soccer-logs ever released to the public [1]." This data set covers seven prominent male soccer competitions which includes 2017/2018 five national soccer competitions in Europe: Spanish first division, Italian first division, English first division, German first division and French first division. In addition to that there is World cup 2018 and European cup 2016 datasets as provided by [1].

The dataset has been provided in seven files in JSON format and consists of information corresponding to competitions, matches, teams, players, events, referees and coaches. Basically, data covers a total of around 1,941 matches, 3,251,294 events and 4,299 players.

However, this study's interest lies in only the 55 non-draw matches that took place in the 2018 World Cup. For each match a graph was constructed with players as nodes and passes between players as edges. The only files required were matches, teams, players and events to form these graphs and conduct the analysis. This approach will be elaborated later in paper.

The second dataset was obtained from a GitHub repository for a project called 2018-world-cup-stats. For this project, data was collected by scraping https://fifa.com statistics webpages and then a .csv file was generated for each team/game. This dataset was required to obtain the data in one column called "attempts on target" [18].

B. Approach

To prove the hypothesis, it was necessary to understand whether teams who played offensive during matches won or lost. Although all 64 matches of World Cup 2018 were examined, the drawn matches were omitted, as they were not informative. This study introduces a novel method to compare the network statistics of the offensive sections of the two participating teams. The offensive section of a team comprises its midfielders and forwards while the defensive section comprises the defenders and goalkeeper(s). This novel method is named the LucaBall score.

Inspiration for this methodology originated from Luca Pappalardo et al.'s paper [1]. The LucaBall score is therefore homage to the original paper. In addition to three metrics used, algebraic connectivity, density and global reaching centrality, in the original paper, clustering coefficient and Average Node Degree were included as they were deemed relevant as per previous research (refer to the Related Work section). The LucaBall score takes into consideration a total of 5 network metrics, namely, average node degree, algebraic connectivity, clustering coefficient, density and global reaching centrality.

A brief overview of important measures of networks used in analyzing the passing networks follows.

1) Clustering Coefficient: The clustering coefficient indicates tightly knit groups within a network. The overall level of clustering in a network is measured by Watts and Strogatz [8] as the average of the local clustering coefficients of all the vertices. A high clustering coefficient in a team implies players in a team are tightly knit. This means more groups of players are available to pass the ball and attempt to make a goal.

2) Algebraic Connectivity: The algebraic connectivity of a graph is defined as the second smallest eigenvalue of the Laplacian matrix [9] of the graph, which is a parameter to measure how well a graph is connected. This value indicates the robustness of the attacking team. High connectivity in the team implies that each player can pass the ball to another player, subsequently increasing the chances of a goal and is associated with a better overall attacking performance.

3) Density: Density is a measure of the number of potential connections in a network. In real world applications, a dense group of objects has many connections among its entities (i.e., has a high density), while a sparse group has few of them (i.e., has a low density). High density in a team implies that players are passing the ball frequently with each other, thus they are more connected and coordinated.

4) Average node degree: Average node degree measures the structural cohesion of a network. The degree of a node is the number of edges connected to the node. Average degree is simply the average number of edges per node in the graph. To calculate the average degree, all degrees are summed up first and then divided by the total number of nodes in the network. In a football context, a team's performance may be indicated through its average node degree, high values indicating overall high passing behaviour, possibly translating to more evenly distributed involvement among teammates compared to the opposing team.

5) Global reaching centrality: The global reaching centrality of a weighted directed graph is the average over all nodes of the difference between the local reaching centrality of the node and the greatest local reaching centrality of any node in the graph. Informally, the local reaching centrality is the proportion of the graph that is reachable from the neighbors of a particular node. In a football context, this refers to the area or the players a node could reach through his closest teammate.

In the following discussion, the steps involved to calculate the LucaBall score and use it to compare teams' success will be outlined. Note that the two teams involved in a match are referred to as Home and Away teams in this study.

Firstly, from the match's passing network of each pair of teams, the two team's networks were separated into the Home and Away teams. The next step involved separating the offensive and defensive sections of each Home and Away team.. Thirdly, the network metrics were calculated, namely, the average node degree, algebraic connectivity, clustering coefficient, density and global reaching centrality of both team's offensive network. Taking these five metrics into consideration, LucaBall has been scored out of five. The average node degree, algebraic connectivity, clustering coefficient, and density of the Home and Away team's offensive networks were then compared. The team with the *higher* metric for each of the above mentioned four statistics was assigned a score of one. Conversely, with respect to the global reaching centrality metric, the team with *lower* metric was assigned a score of one. All scores were summed to calculate the grand total LucaBall score. For example, if the average node degree, clustering coefficient, algebraic connectivity and density of a particular Home team was higher compared to the match's Away team, its LucaBall score would have been four. Similarly, if the Away team's algebraic centrality was lower than the Home team, the LucaBall score of the Away team would have been one.

Thus, the LucaBall score attempts to quantify which team had a more offensive approach with regards to network statistics and whether a higher LucaBall score affects the outcome of a match. The final score validated the study's hypothesis.

C. *Algorithm*

In network analysis, several tools have been developed to address different problems related to the extraction of useful information from systems modelled as graphs [6]. "Soccer-logs enable the analysis of interactions between players through the reconstruction of a team's passing network, a representation of the movement of the ball between players during a match [1]." This study analysed the network metrics outlined in detail in the preceding section, using the LucaBall score for the participating teams in the 2018 FIFA World Cup.

The code was written using Python. The python libraries chosen for the coding were pandas, NumPy, xlsxwriter,xlrd,matplotlib and NetworkX®.

All network graphs contained nodes, which represented players, and edges, which represented all passes. Each player was connected with other players if there were passes between them. Among the seven possible events in the primary dataset, namely, pass, foul, shot, duel, free kick, offside, and touch, only simple passes were considered.

The process of creating graphs initially involved the construction of a single match graph. Subsequently, in order to separate the match graph into subgraphs using a primary key of the team ID, it was necessary to remove all the passes *between* the Home and Away teams. The next step involved the same type of code, which was implemented to separate the offensive and defensive sections in each team graph. The offensive or 'attack' graph constituted midfielder and forward roles only, while the defense graph constituted defender and goalkeeper roles only.

Iterations within the code were required to construct a total of 64*7 graphs. Graphs were created for all 64 matches and originating from each match, a set of seven graphs were subsequently constructed:

- Match graph
- Home team graph
- Home team attack graph
- Home team defence graph
- Away team graph
- Away team attack graph
- Away team defence graph

For each Home and Away attack graph, the novel LucaBall score algorithm calculated the five network metrics (average node degree, algebraic connectivity, clustering coefficient, density and global reaching centrality). The LucaBall score algorithm assigned a value of one to the team which possessed:

- The higher metric score for average node degree, algebraic connectivity, clustering coefficient, and density (total possible summed score = 4)
- The lower metric score for global reaching centrality (total possible score = 1)
- In cases of a tie, each team was assigned a score of 1 (total possible score = 1).

The algorithm coded the LucaBall score out of 5. Refer to the Appendix for all algorithms produced. It should be noted that most coding was original, with a small amount based on the code created by Naomi Arnold.

III. Results and Findings

In the FIFA World Cup 2018, only 55 out of 64 matches had results, as the other 9 were drawn. This study analysed the network characteristics of Home and Away attack graphs created by the proposed pass network for those 55 matches.

Analysis indicates that the total LucaBall score for all teams which won their matches was 153 compared to 126 for all teams which lost. It validates the study's attempt to associate specific network properties with successful (or unsuccessful) offensive strategies, supporting the idea that strong cooperation between teammates makes teams stronger and more successful [10].

Table I reveals that teams with a LucaBall score of zero suffered almost 50% more losses than victories. Similarly, teams with a LucaBall score above three enjoyed 40% more victories than losses whereas those with LucaBall scores below two suffered 40% more losses than victories.. Ultimately, as the LucaBall score increases, the greater the probability that a team would enjoy a win also increases.

TABLE I. LucaBall Score and number of team wins and losses

LucaBall score	# of wins	# of loss
0	9	13
1	7	11
2	9	6
3	6	8
4	10	6
5	14	11

Below an examination of a few interesting matches of the FIFA World Cup 2018 reveals unexpected results.

1)Russia vs Saudi Arabia match

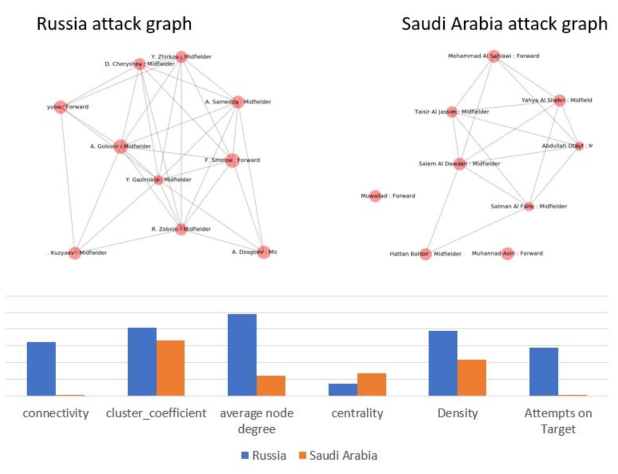

Fig.1 Representation of the player passing networks of the match Russia-Saudi Arabia. Nodes represent attacking players, edges represent passes between players. Russia vs Saudi Arabia attack graphs and the network characteristics comparison of Russia and Saudi Arabia attack graphs.

An article in "Time" revealed that Russia having dominated Saudi Arabia was among the top five shocking surprises during the group stages [26]. However, as per this study's approach, the results are hardly surprising; Saudi Arabia was outclassed by Russia in all five metrics within the LucaBall score (LucaBall score = 5), while Saudi Arabia scored zero which is shown in Fig 1.

2) Spain vs Russia match

As per "The Washington Post", Russia's victory over Spain was a huge upset [27], yet Spain and Russia scored five and zero respectively in their LucaBall score. In this case, the LucaBall method supporting Spain as the winner contradicts the actual result of the match. One possible explanation is that the consequence of penalties determined Spain's defeat and the LucaBall method did not account for this penalty factor. Opportunities to improve the LucaBall algorithm by incorporating penalty shootout statistics are available.

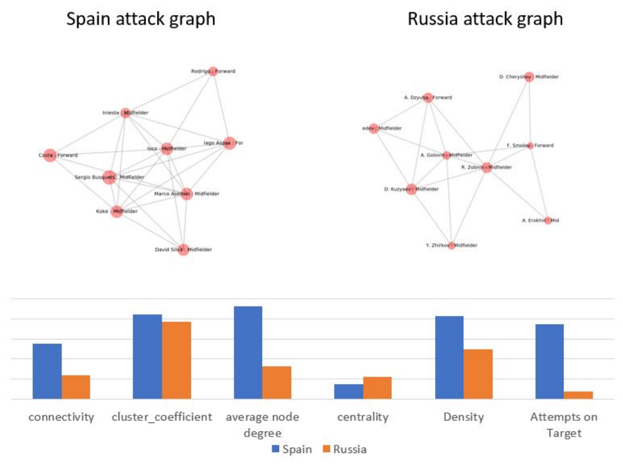

Fig. 2 Representation of the player passing networks of the match Spain-Russia. Nodes represent attacking players, edges represent passes between players.Spain vs Russia attack graphs and the network characteristics comparison of Spain and Russia attack graphs

3) Mexico vs Germany match

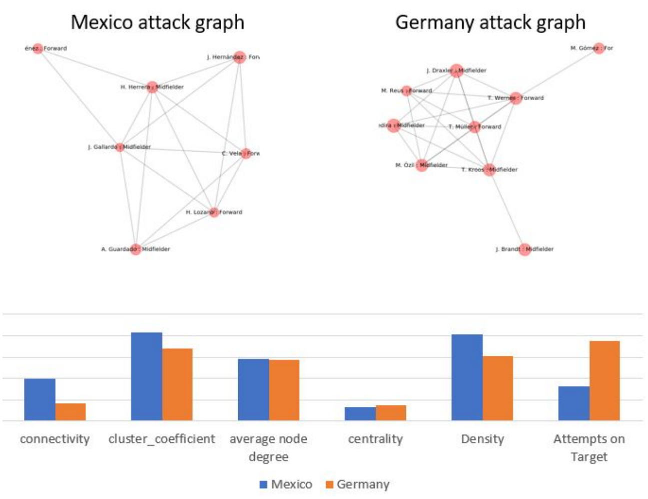

Fig.3 Representation of the player passing networks of the match Mexico-Germany. Nodes represent attacking players, edges represent passes between players. Mexico vs Germany attack graphs and the network characteristics comparison of Mexico and Germany attack graphs

Mexico, ranked 16, pipped defending champions Germany, ranked 1, to produce arguably one of the biggest surprises in FIFA World Cup's recent history. On the contrary, as per the LucaBall method, this seemed like a fair result. Mexico and Germany obtained a LucaBall score of four and one respectively. In other words, Mexico outclassed Germany in 4 out of the 5 metrics discussed in our approach and came out victorious. Intuitively, the team with higher attempts on target has a better chance of winning. The interesting point with regards to this match is that Germany had more than double the number of shots on target compared to Mexico. This study's approach supports the fact that a sole reliance on a high number of shots on target is inadequate to predict a victory.

TABLE II TOTAL LUCABALL SCORE OBTAINED BY TOP 20 TEAMS IN THE GROUP STAGE AND TOTAL LUCABALL SCORE BY TEAMS IN KNOCKOUT PHASES

Group stage	Total LucaBall score	Round of 16	Total LucaBall score	Quarter Finals	Total LucaBall score	Semi-Finals and Finals	Total LucaBall score
Croatia	14	Spain	5	England	4	Croatia	7
Tunisia	14	Mexico	5	Croatia	3	England	6
Uruguay	13	Denmark	5	Brazil	3	Belgium	6
Russia	11	Belgium	5	Russia	2	France	3
Colombia	11	Colombia	4	France	2		
Korea Republic	11	Uruguay	3	Belgium	2		
Germany	10	Switzerland	3	Sweden	1		
Peru	10	Argentina	3				
Brazil	10	Sweden	2				
Mexico	9	Portugal	2				
Morocco	8	France	2				
Serbia	8	England	1				
France	7	Russia	0				
Japan	7	Japan	0				
England	7	Croatia	0				
Poland	6	Brazil	0				
Argentina	6						
Belgium	5						
Iceland	4						
Saudi Arabia	4						

Table II provides the LucaBall score for all three matches played by every team in the Group stage and were summed. Three-quarters of the teams that progressed to the knockout phases were in the Top 20 list of teams.

V. Future work for improving LucaBall score

This work is limited by the sample size; football has been played in over 200 countries, out of which only 55 matches (from the 2018 World Cup) were included in our sample. Although the main reasons for that are of practical matters, given the hardship in obtaining reliable and time-consistent data, the events become very sparse for most of the leagues going back in time.

An area for improvement is how to divide the score when two teams' metrics are tied. In our study, all the metrics were given equal weightage, but giving different weightage to different metrics may improve the result. Another area for improvement is that there could be additional network metrics to consider, which may improve the algorithm.

VI. Extended Work

Task 1. Path Length and Clustering Coefficients

This task compares static network properties such as average shortest path length and cluster coefficient of a match graph obtained using our algorithm to that of a random network. Considering the same number of nodes and average number of edges per node as that of a match graph, the aim is to check if random graphs can model our real network.

Figure 4 shows a graph of a football passing network of England against Belgium in the World Cup 2018 which was an interesting match. Here, nodes represent players, edges represent passes between players.

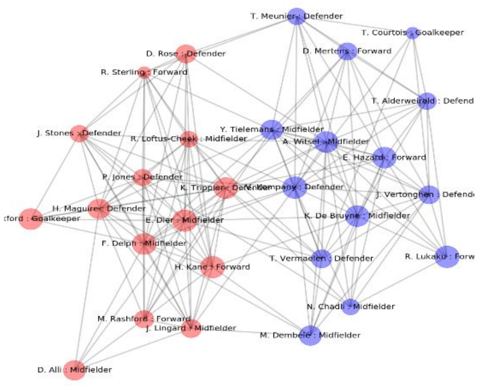

Fig. 4 England vs Belgium graph

The simplest, most well-studied and famous random graph model is most commonly known as the Erdos-Renyi model [11,12]. In this model each possible edge appears independently and with identical probability. We took three such random graphs and compared our graph network statistics with the average calculated across three.

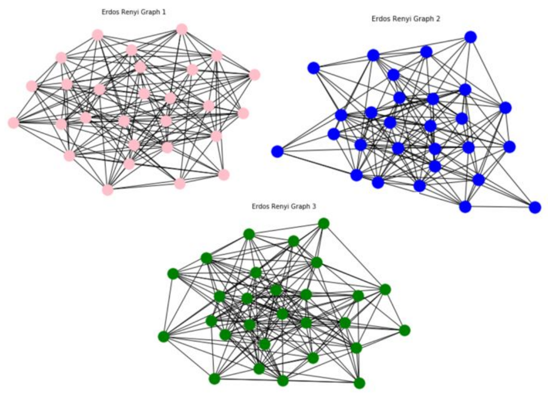

Fig. 5 Erdos-Renyi Graph.

Table III shows the network characteristics of the England vs Belgium match graph and the Erdos-Renyi graph. In summary, we find that the random network model does not capture the clustering of real networks. Instead, real networks have a higher clustering coefficients than expected for a random network of similar number of nodes and edges.

TABLE III. Network statistics of England vs Belgium graph and Erdos-Renyi Graph

Network Statistics	Number of nodes	Number of edges	Average cluster coefficient	Average shortest path length
Erdos-Renyi Graph (average of 3)	28	169.33	0.44	1.59
England vs Belgium Match graph	28	168	0.67	1.63

The match network was found to present a higher clustering coefficient than the corresponding random network, which means a higher tendency to create triangles and tightly-knit groups, which is shown in Fig. 6.

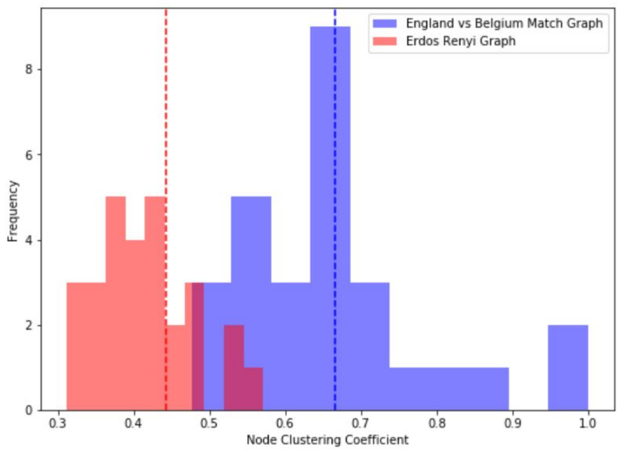

Fig. 6 Cluster coefficient comparison between England Vs Belgium match graph and three Erdos-Renyi random graphs.

L(i,j) is the shortest path length(s) between i and j. The average shortest path length over all pairs of nodes characterizes how large the world represented by the network is, where a small length implies that the network is well connected globally. Fig. 7 shows that the real network and the random network is well-connected. It is likewise observed that the match graph's average shortest path length is well represented by the random graph.

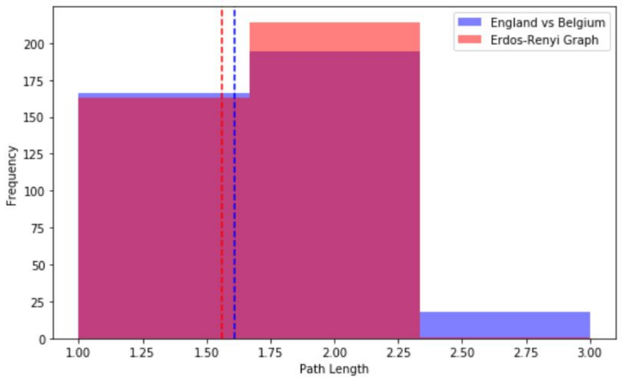

Fig. 7 Average shortest path length comparison between England Vs Belgium match graph and three Erdos-Renyi random graphs.

Task 2. Robustness of graph

Network robustness tells us the ability of a network to withstand failures. This is a very critical attribute to know especially in complex systems such as infrastructure or power grids where the failure of one node can cause huge problems and cost millions. In football, high robustness of a team represents strong links between its players [1].

We wanted to check the robustness of our match graph and compare it along with three Erdos-Renyi generated random networks. We checked the robustness by removing a high degree node in each iteration and looked at the component size. Below are the results:

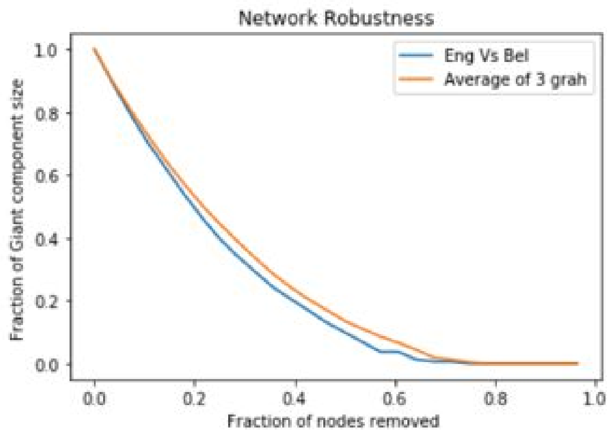

Fig.8 Robustness comparison between England vs Belgium(Eng Vs Bel) match graph and three Erdos-Renyi random graphs

The above graph shows the results of removing fractions of nodes on the component size.

The overall trend we observed is that as the fraction of removed nodes increases, the giant component size decreases. Initially, there is a huge drop in component size. By removing only 20% of the nodes, the giant component size decreases by roughly more than 60%. This can be expected, as we are removing high degree nodes each time, which greatly affects the size of the giant component. Overall, the giant component breaks at around the 80% level of node removal. This shows that, overall, all four networks are very robust and our match graph reveals strong links between the players.

If we compare our match graph with the random graphs, generally they degrade at the same level and are almost equally robust. But according to our results, overall, the random graphs are insignificantly more robust. When we remove 40% of nodes, the component size of our match graph drops to 20% as compared to the random graph whose component size is approximately 30%. This is because in random graphs each node has approximately the same degree and thus contributes to the interconnectedness by relatively the same amount. Therefore, removing nodes has a greater effect on the giant component of the match graph as compared to the random graphs.

Task 3. Pagerank and its comparison to the random graphs

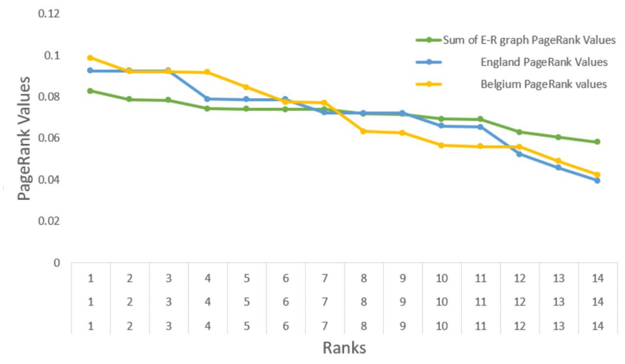

Fig.9 PageRank comparison between England vs Belgium(Eng Vs Bel) match graph and three Erdos-Renyi random graphs

PageRank, as mentioned previously in this paper, is a popular algorithm coined by Larry Page. This algorithm can be used in the passing network of a football match to find the player who has been his team's fulcrum. To proceed with this analysis, the passing network of each team involved in a match was separated. This was done, to take into consideration that not all passes are amongst team members. Some passes occur between players of different teams. Since these passes are not intentional, thus, they were discarded.

It was seen that within the Belgium team, Kevin De Bruyne was the most influential while Musa Dembele was the least. With regards to England, Fabian Delph was the most influential with Delle Ali being the least. The range of the two real graphs is wider than the random graph. It is observed that the Erdos Renyi random graph has values which are closer to the mean when compared with Belgium and England's graph. Also, England's graph is similar to the random graph, which could have contributed to England's defeat, but more work is needed to validate this theory.

Task 4. Cascades

In network theory, cascades refer to the sequential adoption of a particular behaviour or information. As one 'seed' node adopts a novel idea or behaviour, the direct neighbours may or may not also adopt the same, resulting in a 'cascade' or spread of the new idea/behaviour, sometimes throughout the network, depending on the threshold in which adopters require to make that leap.

In a football team, we are assuming that the new behaviour is the adoption of an offensive strategy versus a defensive one. In order for the direct neighbour of the initial player to adopt an offensive strategy, the percentage of passes made to offensive players, either forwards or midfielders, is calculated with respect to the total number of direct neighbours. Since a football team is so well-connected and highly clustered, it would be challenging to isolate direct neighbours. Thus, for this task, we will make ***theoretical*** calculations.

In the England team, if the seed node, midfielder E. Dier, is the highest degree node, he initializes the offensive strategy by passing to midfielder F. Delph. If he continues to pass the ball to the next forward, H. Kane, H. Kane will also choose the offensive strategy by passing to another midfielder, D. Alli. If D. Alli only has one offensive player and 4 defensive players as his direct neighbours, he will choose to pass to a defender such as H. Macguire at the 0.25 threshold level, and the cascade stops. In the case of Alli, only one of his direct neighbours was a forward from a total of five direct neighbours. At the 0.1 threshold level, he would pass to a forward and the cascade would continue. Fig. 10 shows the England team's passing network.

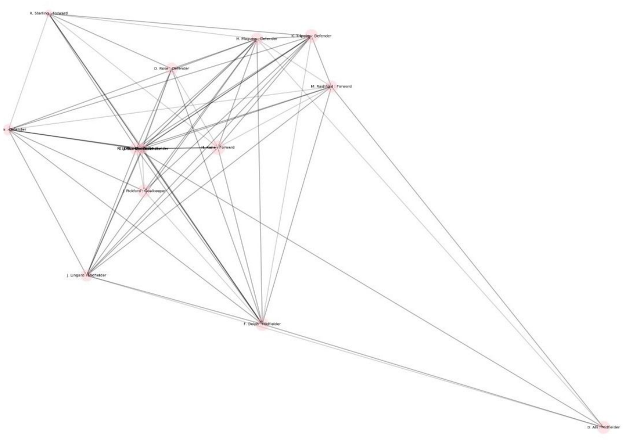

Fig.10 England's passing network

VII. Conclusion

The motivation behind this study was to break down the contrasts between the triumphant and the losing groups in World Cup 2018. Specifically, we examined the notion that the successful winning teams display an attacking style of play rather than a solid protective way to deal with the game. The LucaBall score was able to successfully justify 30 out of 55 winning matches. The LucaBall score does not give a definitive result, however, it is a promising method.

Lastly, this is a case study and the results cannot be generalized. There are many hidden factors that lead to an unpredictability that is one of the beautiful aspects of the game. Nonetheless, this examination can be utilized as a platform to create strategies that would help in understanding the most mainstream sport on the planet.

Acknowledgment

The authors thank lecturers Laurissa Tokarchuk, Mathieu Barthet and Naomi Arnold for valuable discussions and guidance.

The authors also thank Luca Pappalardo for having a look at this paper's 1st draft and providing the below comments, via LinkedIn:

"I went through it, very good work! Your score seems to be a good indicator of the success of a team (and thanks for the homage!)"

"I am curious to see how the method works for club competitions."

"One thing you may try is to construct evolving attack networks during the match (e.g., recompute the network every 10 minute)"

References

[1] L. Pappalardo et al., "A public data set of spatio-temporal match events in soccer competitions", Scientific Data, vol. 6, no. 1, 2019.

[2] E. Filho, I. Basevitch, Y. Yang i G. Tenenbaum, "Is the best defense a good offense? comparing the brazilian and italian soccer styles", Kinesiology, vol.45., br. 2., str. 213-221, 2013.

[3] T. Kawasaki, K. Sakaue, R. Matsubara and S. Ishizaki, "Football pass network based on the measurement of player position by using network theory and clustering", International Journal of Performance Analysis in Sport, vol. 19, no. 3, pp. 381-392, 2019.

[4] J. Ribeiro, P. Silva, R. Duarte, K. Davids and J. Garganta, "Team Sports Performance Analysed Through the Lens of Social Network Theory: Implications for Research and Practice", Sports Medicine, vol. 47, no. 9, pp. 1689-1696, 2017.

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

[6] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in Proceedings of the 19th international conference on the World wide web. ACM, 2010, pp. 591–600

[7] M. Huang, Y. Yang, and X. Zhu, "Quality-biased ranking of short texts in microblogging services," in In IJCNLP?11, 2011.

[8] H. Huang, A. Zubiaga, H. Ji, H. Deng, D. Wang, H. Le, T. Abdelzaher, J. Han, A. Leung, J. Hancock, and C. Voss, "Tweet ranking based on heterogeneous networks," in Proceedings of COLING 2012. Mumbai, India: The COLING 2012 Organizing Committee, December 2012, pp. 1239–1256. [Online]. Available: http://www.aclweb.org/anthology/C12-1076

[9] Saito R, Smoot ME, Ono K, et al. : A travel guide to Cytoscape plugins. Nat Methods. 2012;9(11):1069–1076. 10.1038/nmeth.2212.

[10] Duch, J., Waitzman, J. S. & Amaral, L. A. N. Quantifying the Performance of Individual Players in a Team Activity. PlosOne 5(6), 1–7 (2010)

[11] D. J. Watts and Steven Strogatz (June 1998). "Collective dynamics of 'small-world' networks". Nature. 393 (6684): 440–442. Bibcode:1998Natur.393..440W. doi:10.1038/30918. PMID 9623998.

[12] FIEDLER, M.: Algebraic connectivity of graphs, Czechoslovak Mathematical Journal 23, pp. 298-305 (1973).

[13] Balkundi, P., and Harrison, D. A. (2006). Ties, leaders, and time in teams: strong inference about network structure's effects on team viability and performance. Acad. Manage. J. 49, 49–68. doi: 10.5465/AMJ.2006.20785500.

[14] Erdo˝s P, Re´nyi A (1960) On the evolution of random graphs. Publ Math Inst Hung Acad Sci 5: 17–60

[15] Bolloba´s B (2001) Random Graphs. Cambridge: Cambridge University Press.

[16] F. Korte, D. Link, J. Groll and M. Lames, "Play-by-Play Network Analysis in Football", Frontiers in Psychology, vol. 10, 2019.

[17] T. Pina, A. Paulo and D. Araújo, "Network Characteristics of Successful Performance in Association Football. A Study on the UEFA Champions League", Frontiers in Psychology, vol. 8, 2017. Available: 10.3389/fpsyg.2017.01173 [Accessed 12 May 2020].

[18] GitLab. 2020. 2018-World-Cup-Stats. [online] Available at: <https://gitlab.com/djh_or/2018-world-cup-stats/-/tree/master> [Accessed 24 April 2020].

[19] M. Dixon and M. Robinson, "A birth process model for association football matches", Journal of the Royal Statistical Society: Series D (The Statistician), vol. 47, no. 3, pp. 523-538, 1998.

[20] D. Dyte and S. Clarke, "A ratings based Poisson model for World Cup soccer simulation", Journal of the Operational Research Society, vol. 51, no. 8, pp. 993-998, 2000.

[21] N. Hirotsu and M. Wright, "An evaluation of characteristics of teams in association football by using a Markov process model", Journal of the Royal Statistical Society: Series D (The Statistician), vol. 52, no. 4, pp. 591-602, 2003.

[22] I. Yates, J. North, P. Ford and M. Williams, “A quantitative analysis of Italy’s World Cup Performances. A comparison of World Cup winners”, Insight - The FA Coaches Association Journal, vol. 6, pp. 55-59.

[23] F. Clemente, M. Couceiro, F. Martins and R. Mendes, "Using Network Metrics in Soccer: A Macro-Analysis", Journal of Human Kinetics, vol. 45, no. 1, pp. 123-134, 2015.

[24] T. Kawasaki, K. Sakaue, R. Matsubara and S. Ishizaki, "Football pass network based on the measurement of player position by using network theory and clustering", International Journal of Performance Analysis in Sport, vol. 19, no. 3, pp. 381-392, 2019. Available: 10.1080/24748668.2019.1611292 [Accessed 12 May 2020].

[25] Boren Wang,Zongwei Luo,”PageRank Approach to Ranking Football Teams’ Network”.

[26] "Five Most Shocking World Cup 2018 Moments so Far", Time, 2020.[Online].Available: https://time.com/5314646/fifa-world-cup-2018-russia-shocks-surprises/. [Accessed: 14- May- 2020].

[27] "Russia goes into the bunker vs. Spain, and emerges with a huge World Cup upset", Washington Post, 2020. [Online]. Available: https://www.google.com/url?q=https://www.washingtonpost.com/news/soccer-insider/wp/2018/07/01/spain-vs-russia-2018-world-cup/&sa=D&ust=1589417436423000&usg=AFQjCNGcpLr-OKi7QVMrI3JvmOnYBx2g1g. [Accessed: 14- May- 2020].

Appendix

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

# # Football Network

# In[8]:


import warnings
warnings.filterwarnings("ignore")
import networkx as nx
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
import scipy
import xlsxwriter
import xlrd
import openpyxl


# In[11]:


#import files players, teams, matches, events

raw_players = pd.read_csv('players.csv')
players = pd.DataFrame(data=raw_players )

worldcup_2018_matches = pd.read_excel('matches_teams_DMSN.xlsx')

with open("teams.json") as f:
    teams = pd.DataFrame(json.load(f))
    f.close()

with open("matches_World_Cup.json") as f:
    matches = pd.DataFrame(json.load(f))
    f.close()

with open("events_World_Cup.json") as f:
    events = pd.DataFrame(json.load(f))
    f.close()


# In[3]:


def create_and_show_graph(events,matchId, teamOneId,teamOneName, teamTwoId, teamTwoName):

    #Only interested in the events which happened in the Cup Final, with matchId 2058017.
    wc_final_events = events[events['matchId']==matchId]
    events = wc_final_events.reset_index(drop=True)

    #initialize the graph
    G = nx.Graph()
    teamColours = {teamOneId: 'red', teamTwoId: 'blue'}

    #Create Graph
    for i in range(len(events)-1):
        if (events['subEventName'][i]=="Simple pass" or events['subEventName'][i]=="High pass" ) and len(events['tags'][i])==1:
            n1, n2 = events['playerId'][i], events['playerId'][i+1]
            p1, p2 = list(players[players['wyId']==n1]['shortName'])[0], list(players[players['wyId']==n2]['shortName'])[0]
            t1, t2 = events['teamId'][i], events['teamId'][i+1]
            r1, r2 = list(players[players['wyId']==n1]['role/name'])[0], list(players[players['wyId']==n2]['role/name'])[0]
            t = events['tags'][i]
            if n1 not in G.nodes():
                G.add_node(n1, teamid=t1, role=r1, color=teamColours[t1], shortName=p1)
            if n2 not in G.nodes():
                G.add_node(n2, teamid=t2, role=r2, color=teamColours[t2], shortName=p2)
            if any([row['id']==1801 for row in t]):
                G.add_edge(n1,n2)

    #add graph node attributes
    player_information = {}

    for n in G.nodes():
        playerName = nx.get_node_attributes(G,'shortName')[n]
        playerRole = nx.get_node_attributes(G, 'role')[n]
        player_information[n] = (playerName + ' : ' + playerRole).encode().decode('unicode_escape')


    #Create graph nodes list
    teamOneNodes = [n for (n, team) in nx.get_node_attributes(G,"teamid").items() if team == teamOneId]
    teamTwoNodes = [n for (n, team) in nx.get_node_attributes(G,"teamid").items() if team == teamTwoId]

    matchDegree = nx.degree(G)
    #Show graph
    a4_dims = (20, 20)
    fig, ax = plt.subplots(figsize=a4_dims)
    ax.set_axis_off()

    pos = nx.kamada_kawai_layout(G)
    nx.draw_networkx_nodes(G, pos, nodelist=teamOneNodes, node_color='red', alpha=0.4, labels=player_information, with_labels=True, node_size=[v * 200 for v in dict(matchDegree).values()])
    nx.draw_networkx_nodes(G, pos, nodelist=teamTwoNodes, node_color='blue', alpha=0.4, labels=player_information, with_labels=True, node_size=[v * 200 for v in dict(matchDegree).values()])
    nx.draw_networkx_edges(G, pos, with_labels=True, alpha=0.2,width=3)
    nx.draw_networkx_labels(G, pos, labels= player_information, font_size=20)

    # create folder based on match

    rootPath= '/Users/Maneesha Majeed/DMSN_results/'
    folderName = teamOneName + ' vs ' + teamTwoName + ' matchId ' + str(matchId)
    _dir = os.path.join(rootPath, folderName)

    # create 'dynamic' dir, if it does not exist
    if not os.path.exists(_dir):
        os.makedirs(_dir)

    plt.savefig(_dir + '/match_graph.PNG')
    plt.title(teamOneName + ' vs ' + teamTwoName, fontsize=20)
#   plt.show()

    return {'graph':G, 'folderName': _dir}


# In[20]:


def create_and_show_subgraph(G,nodeList,color,folderName,fileName):
    #get sub graph
    G_subGraph = G.subgraph(nodeList)

    #create nodelist
    player_information= {}
    for n in G.nodes():
        playerName = nx.get_node_attributes(G,'shortName')[n]
        playerRole = nx.get_node_attributes(G, 'role')[n]
        player_information[n] = (playerName + ' : ' + playerRole).encode().decode('unicode_escape')



    a4_dims = (15, 15)
    fig, ax = plt.subplots(figsize=a4_dims)
    ax.set_axis_off()

    #get node for specific criteria
    g_labels = {node:player_information[node] for node in nodeList}

    #draw graph
    graph_degree = nx.degree(G)
    pos = nx.kamada_kawai_layout(G_subGraph)
    nx.draw_networkx_nodes(G_subGraph, pos, node_color=color, alpha=0.4, labels=player_information, with_labels=True, node_size=[v * 200 for v in dict(graph_degree).values()])
    nx.draw_networkx_edges(G_subGraph, pos, with_labels=True,alpha=0.2,width=3)
    nx.draw_networkx_labels(G_subGraph,pos, labels=g_labels, font_size=20)

    plt.savefig(folderName + '/'+ fileName)
#     plt.show()

    return G_subGraph


# In[14]:


def calculate_graph_network_statistics(G):

    graph_degree = nx.degree(G)
    graph_degree_dic = dict(graph_degree)
    sum_graph_degree=np.sum(list(graph_degree_dic.values()))
    average_nodedegree = sum_graph_degree/float(len(G.nodes()))

    cluster_coefficient = nx.average_clustering(G)
    connectivity = nx.algebraic_connectivity(G)
    density = nx.density(G)
    centrality =nx.global_reaching_centrality(G)

    return {'cluster_coefficient': cluster_coefficient,'average_nodedegree': average_nodedegree,
            'connectivity':connectivity,'density': density,'centrality':centrality}




# In[ ]:


writer = pd.ExcelWriter('matches_teams_DMSN_output.xlsx',engine='xlsxwriter')

for index, match in worldcup_2018_matches.iterrows():

     # get match information from file
    matchId = match['matchId']
    teamName = match['Team']
    teamId = match['team_id']

    opponentName = match['Opponent']
    opponentId = match['opponent_id']

    #create match graph
    match_information = create_and_show_graph(events, matchId, teamId, teamName, opponentId, opponentName)
    match_graph = match_information['graph']
    folderName= match_information['folderName']

    #create team and opponent graph
    team = [n for (n, team) in nx.get_node_attributes(match_graph,"teamid").items() if team == teamId]
    opponent = [n for (n, team) in nx.get_node_attributes(match_graph,"teamid").items() if team == opponentId]
    team_graph = create_and_show_subgraph(match_graph,team, 'pink',folderName , (teamName +'_graph' ))
    opponent_graph =  create_and_show_subgraph(match_graph,opponent, 'green',folderName , (opponentName+'_graph'))

    #create team attack and defend graph
    team_attack_nodes = [n for (n, role) in nx.get_node_attributes(team_graph,"role").items() if role == 'Forward' or role == 'Midfielder']
    team_defend_nodes = [n for (n, role) in nx.get_node_attributes(team_graph,"role").items() if role == 'Defender' or  role == 'Goalkeeper']
    team_attack_graph = create_and_show_subgraph(team_graph ,team_attack_nodes, 'red',folderName,(teamName +'_attack_graph'))
    team_defend_graph = create_and_show_subgraph(team_graph ,team_defend_nodes, 'blue',folderName,(teamName +'_defend_graph'))

    #create opponent attack and defend graph
    opponent_attack_nodes = [n for (n, role) in nx.get_node_attributes(opponent_graph,"role").items() if role == 'Forward' or role == 'Midfielder']
    opponent_defend_nodes = [n for (n, role) in nx.get_node_attributes(opponent_graph,"role").items() if role == 'Defender' or  role == 'Goalkeeper']
    opponent_attack_graph = create_and_show_subgraph(opponent_graph ,opponent_attack_nodes, 'red',folderName,(opponentName +'_attack_graph'))
    opponent_defend_graph = create_and_show_subgraph(opponent_graph ,opponent_defend_nodes, 'blue',folderName,(opponentName +'_defend_graph'))

    #team attack network statistics
    team_attack_statistics = calculate_graph_network_statistics(team_attack_graph)

    team_cluster = pd.DataFrame([team_attack_statistics['cluster_coefficient']])
    team_connectivity = pd.DataFrame([team_attack_statistics['connectivity']])
    team_average_nodedegree =pd.DataFrame([team_attack_statistics['average_nodedegree']])
    team_density=pd.DataFrame([team_attack_statistics['density']])
    team_centrality=pd.DataFrame([team_attack_statistics['centrality']])

    #write to excel columns
    team_cluster.to_excel(writer,startcol=17, startrow=index, header=None, index=False)
    team_connectivity.to_excel(writer,startcol=18, startrow=index, header=None, index=False)
    team_average_nodedegree.to_excel(writer,startcol=19, startrow=index, header=None, index=False)
    team_density.to_excel(writer,startcol=20, startrow=index, header=None, index=False)
    team_centrality.to_excel(writer,startcol=21, startrow=index, header=None, index=False)



    #opponent attack network statistics
    opponent_attack_statistics = calculate_graph_network_statistics(opponent_attack_graph)

    opponent_cluster = pd.DataFrame([opponent_attack_statistics['cluster_coefficient']])
    opponent_connectivity = pd.DataFrame([opponent_attack_statistics['connectivity']])
    opponent_average_nodedegree =pd.DataFrame([opponent_attack_statistics['average_nodedegree']])
    opponent_density=pd.DataFrame([opponent_attack_statistics['density']])
    opponent_centrality=pd.DataFrame([opponent_attack_statistics['centrality']])

    #write to excel columns
    opponent_cluster.to_excel(writer,startcol=10, startrow=index, header=None, index=False)
    opponent_connectivity.to_excel(writer,startcol=11, startrow=index, header=None, index=False)
    opponent_average_nodedegree.to_excel(writer,startcol=12, startrow=index, header=None, index=False)
    opponent_density.to_excel(writer,startcol=13, startrow=index, header=None, index=False)
    opponent_centrality.to_excel(writer,startcol=14, startrow=index, header=None, index=False)


writer.save()

print('DataFrame is written successfully to Excel File.')


# # Results Analysis to calculate LucaBall score
#

# In[6]:


#Uploading the results excel
#results_raw = pd.read_excel('matches_teams_DMSN_Maneesha.xlsx')
results_raw = pd.read_excel('DMSN_football_LucaBall score_updated.xlsx')
results = pd.DataFrame(data=results_raw )
results


# In[7]:


#To take the count of WDL
print(results['WDL'].value_counts())
print(results.shape)
print(results.info())


# In[8]:


#To remove all the draw matches
results = results[results['WDL'] != 'D']

#To get the data for win matches
results_win=results.loc[results['WDL']=='W']

print(results_win['teams_attempts_on_target'].sum())
print(results_win['opponent_attempts_on_target'].sum())


#To get the data for loss matches
results_loss=results.loc[results['WDL']=='L']
print(results_loss['teams_attempts_on_target'].sum())
print(results_loss['opponent_attempts_on_target'].sum())


# In[9]:


#Select the required columns to calculate the network statistics
results_cal =results[[ 'Game','Group','Team', 'Opponent', 'Goals For', 'Goals Against','matchId',
                    'teams_attempts_on_target', 'opponent_attempts_on_target',
                   'opponent_attack_cluster_coefficient','opponent_attack_connectivity', 'opponent_attack_average_nodedegree',
      'opponent_attack_density','opponent_attack_centrality','team_attack_centrality',
       'team_attack_cluster_coefficient','team_attack_connectivity', 'team_attack_average_nodedegree','team_attack_density',
                   'WDL']]
results_cal


# ### Attack statistics

# In[10]:


#Attack cluster coefficient
results_cal[['team_attack_cluster_coefficient','team_attack_connectivity','team_attack_density','team_attack_centrality','team_attack_average_nodedegree','te
ams_attempts_on_target']].describe()


# In[11]:


#Normalized Data
cols_to_norm = ['teams_attempts_on_target', 'opponent_attempts_on_target',
              'team_attack_connectivity','opponent_attack_connectivity',
             'team_attack_average_nodedegree', 'opponent_attack_average_nodedegree']
results_cal[cols_to_norm] = results_cal[cols_to_norm].apply(lambda x: (x - x.min()) / (x.max() - x.min()))


# In[12]:


results_cal['team_attack_cluster_coefficient'].value_counts(bins=5)


# In[13]:


#To find the matches with tie for metric score
results_cal.loc[results_cal['teams_attempts_on_target'] == results_cal['opponent_attempts_on_target']]


# In[14]:


results_cal.loc[results_cal['team_attack_connectivity'] == results_cal['opponent_attack_connectivity']]


# In[15]:


results_cal.loc[results_cal['team_attack_average_nodedegree'] == results_cal['opponent_attack_average_nodedegree']]


# In[16]:


results_cal.loc[results_cal['team_attack_density'] == results_cal['opponent_attack_density']]


# In[17]:


results_cal.loc[results_cal['team_attack_cluster_coefficient'] == results_cal['opponent_attack_cluster_coefficient']]


# In[18]:
```

```
results_cal.loc[results_cal['team_attack_centrality'] == results_cal['opponent_attack_centrality']]


# In[19]:


#Attack
results_cal['attack_cluster_coefficient'] = np.where(results_cal['team_attack_cluster_coefficient'] >results_cal['opponent_attack_cluster_coefficient'], 1,0)
results_cal['attack_attempts_on_target']=  np.where(results_cal['teams_attempts_on_target'] >results_cal['opponent_attempts_on_target'], 1,0)
results_cal['attack_connectivity'] = np.where((results_cal['team_attack_connectivity'] >=results_cal['opponent_attack_connectivity']),1,0)
results_cal['attack_average_nodedegree'] = np.where(results_cal['team_attack_average_nodedegree'] >= results_cal['opponent_attack_average_nodedegree'],
1,0)
results_cal['attack_density'] = np.where(results_cal['team_attack_density'] >= results_cal['opponent_attack_density'], 1,0)
results_cal['attack_centrality'] = np.where(results_cal['team_attack_centrality'] < results_cal['opponent_attack_centrality'],1,0)
results_cal


# In[20]:


#Calculating LucaBall score(attack stats)
results_cal['attack_stats'] =  results_cal[['attack_average_nodedegree','attack_cluster_coefficient',
                                'attack_connectivity','attack_centrality','attack_density']].sum(axis=1)

results_cal['attack_stats_total'] = np.where(results_cal['attack_stats'] >=2.5, 1,0)


# In[21]:


print(results_cal.groupby(['WDL'])['attack_stats_total'].value_counts())
print(results_cal.groupby(['WDL'])['attack_stats'].value_counts())


# In[22]:


#To calculate the total attack stats with respect to W and L
print(results_cal.groupby(['WDL'])['attack_stats'].sum())


# In[23]:


results_cal.groupby(['attack_stats','attack_attempts_on_target'])['WDL'].value_counts()


# In[24]:


print("Attempts on Target")
print(results_cal.groupby(['WDL'])['attack_attempts_on_target'].value_counts())
print("Attack Density")
print(results_cal.groupby(['WDL'])['attack_density'].value_counts())
print("Attack Clustering coefficient")
print(results_cal.groupby(['WDL'])['attack_cluster_coefficient'].value_counts())
print("Attack average Node degree")
print(results_cal.groupby(['WDL'])['attack_average_nodedegree'].value_counts())
print("Attack Centrality")
print(results_cal.groupby(['WDL'])['attack_centrality'].value_counts())
print("Attack Connectivity")
print(results_cal.groupby(['WDL'])['attack_connectivity'].value_counts())


# In[25]:


pd.pivot_table(results_cal, values = 'matchId', index = 'WDL',
            columns = ['attack_stats_total','attack_attempts_on_target'],
            aggfunc = lambda x: sum(x)).plot.bar()
plt.legend(loc='upper right',fontsize=6)
```

```
# In[26]:


pd.pivot_table(results_cal, values = 'matchId', index = 'WDL',
               columns = ['attack_attempts_on_target','attack_centrality'],
               aggfunc = lambda x: len(x)).plot.bar()


# In[27]:


pd.pivot_table(results_cal, values = 'matchId', index = 'WDL', columns =['attack_attempts_on_target' ,'attack_connectivity'],
               aggfunc = lambda x: len(x)).plot.bar()


# In[28]:


pd.pivot_table(results_cal, values = 'matchId', index = 'WDL', columns = ['attack_attempts_on_target','attack_average_nodedegree'],
               aggfunc = lambda x: len(x)).plot.bar()


# In[29]:


pd.pivot_table(results_cal, values = 'matchId', index = 'WDL', columns =['attack_attempts_on_target', 'attack_density'],
               aggfunc = lambda x: len(x)).plot.bar()


# In[30]:



sns.countplot(data=results_cal[['teams_attempts_on_target','attack_stats','WDL']],x='attack_stats',hue='WDL')


# In[31]:


#To get the data for win matches
results_win=results_cal.loc[results_cal['WDL']=='W']
print(results_win['attack_stats'].value_counts())
print(results_win['attack_stats'].sum())
print(results_win['teams_attempts_on_target'].sum())
print(results_win['opponent_attempts_on_target'].sum())
print(results_win.groupby(['WDL'])['attack_attempts_on_target'].sum())


#To get the data for loss matches
results_loss=results_cal.loc[results_cal['WDL']=='L']
print(results_loss['attack_stats'].value_counts())
print(results_loss['attack_stats'].sum())
print(results_loss.groupby(['WDL'])['attack_attempts_on_target'].sum())


# ### Teams Analysis

# In[32]:


#Results group stage
results_groupstage=results_cal[0:77]
results_teamscore=pd.DataFrame(results_groupstage.groupby(['Team'])['attack_stats'].sum())
#results_teamscore.sort_values(ascending=False)

results_groupstage_teams=results_groupstage.groupby(['Team'])['attack_stats'].sum()
# list= results_roundof16['Team'].unique()
#print(results_teamscore)
print(results_groupstage_teams.sort_values(ascending=False))


# In[33]:
```

```
#Results round of 16
results_roundof16=results_cal[78:94]
results_teams_roundof16=results_roundof16.groupby(['Team'])['attack_stats'].sum()
results_teams_roundof16.sort_values(ascending=False)


# In[34]:


#Results quarter finals
results_quarterfinals=results_cal[95:102]
results_teams_quarterfinals=results_quarterfinals.groupby(['Team'])['attack_stats'].sum()
results_teams_quarterfinals.sort_values(ascending=False)


# In[35]:


#Results Semifinals and finals
results_semifinals=results_cal[102:110]
results_teams_semifinals=results_semifinals.groupby(['Team'])['attack_stats'].sum()
results_teams_semifinals.sort_values(ascending=False)


# In[36]:


#Knock out stage
results_knockout=results_cal[78:110]
results_teams_knockout=results_knockout.groupby(['Team'])['attack_stats'].sum()
results_teams_knockout.sort_values(ascending=False)


# In[37]:


#winning teams
results_win=results_cal.loc[results_cal['WDL']=='W']
results_teams_win=results_win.groupby(['Team'])['attack_stats'].sum()
results_teams_win.sort_values(ascending=False)


# ### France and  Croatia  - Final teams

# In[38]:


#France
results_france=results_cal.loc[results_cal['Team']== 'France']
results_france[['WDL','Group','Team','Opponent','matchId','attack_stats','teams_attempts_on_target','team_attack_density',
             'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]


# In[39]:


#Croatia
results_croatia=results_cal.loc[results_cal['Team']== 'Croatia']
results_croatia[['WDL','Group','Team','Opponent','attack_stats','matchId','teams_attempts_on_target','team_attack_density',
             'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]


# # Match Analysis

# ### Spain vs Russia

# In[40]:


#Spain vs Russia
```

```
results_spain=results_cal.loc[results_cal['matchId']== 2058004]
results_spain[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density','attack_stats','WDL',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity','team_attack_centrality']]


# ### Mexico vs Germany

# In[41]:


#Matches 1 mexico vs germany 2057984
results_mexico=results_cal.loc[results_cal['matchId']== 2057984]
results_mexico[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','attack_stats','opponent_attempts_on_target','team_attack_connectivit
y',
            'opponent_attack_centrality','team_attack_centrality','opponent_attack_density','team_attack_density',
            'team_attack_average_nodedegree','opponent_attack_average_nodedegree']]


# ### Argentina vs Croatia

# In[42]:


# match 3 argentina vs croatia 2057974
results_croatia=results_cal.loc[results_cal['matchId']== 2057974]
results_croatia[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]


# ### Belgium vs Brazil

# In[43]:


#match 2 belgium vs brazil 2058011
results_belgiumvsbrazil=results_cal.loc[results_cal['matchId']== 2058011]
results_belgiumvsbrazil[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]


# ### Belgium vs Japan

# In[44]:



#match 2 belgium vs japan 2058007
results_belgiumvsjapan=results_cal.loc[results_cal['matchId']== 2058007]
results_belgiumvsjapan[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]


# ## Russia vs Saudi arabia

# In[45]:



results_russia=results_cal.loc[results_cal['matchId']== 2057954]
results_russia[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density','attack_stats','WDL',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity','team_attack_centrality']]


# ### Semi Finals

# In[46]:


#Semi Finals
results_semifinals=results_cal.loc[results_cal['Group']== 'Semi-finals']
results_semifinals[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density','attack_stats',
            'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity']]
```

```
# ### Quarter Finals

# In[47]:


#Quarter Finals
results_quarterfinals=results_cal.loc[results_cal['Group']== 'Quarter-finals']
results_quarterfinals[['WDL','Group','Team','Opponent','matchId','teams_attempts_on_target','team_attack_density','attack_stats',
              'team_attack_cluster_coefficient','team_attack_average_nodedegree','team_attack_connectivity','team_attack_centrality']]


# ####  Correlation

# In[48]:


results_corr=results_cal.copy()


# In[49]:


results_corr


# In[50]:


#replace wdl to 1 and 0
results_corr['WDL']=results_corr['WDL'].replace({'W': 1, 'L': 0})
results_corr['WDL']


# In[51]:


print(sns.heatmap(results_corr.corr()))
print(results_corr.corr())


# In[52]:


#to get the correlation
results_attack=results_corr[['WDL','Goals For','teams_attempts_on_target','team_attack_density','team_attack_cluster_coefficient','attack_connectivity',
                         'team_attack_average_nodedegree','team_attack_centrality','attack_stats']]
print(results_attack.corr())
print(sns.heatmap(results_attack.corr()))


# In[53]:



#To find the correlation between columns

x = results_corr['teams_attempts_on_target']
y = results_corr['Goals For']

print(scipy.stats.pearsonr(x, y))

'''Here I can see both: the correlation - which is really strong and then the p-value, which indicates that
this is statistically significant.'''


# In[54]:


x = results_corr['teams_attempts_on_target']
y = results_corr['team_attack_connectivity']

print(scipy.stats.pearsonr(x, y))
```

```
# In[55]:


x = results_corr['teams_attempts_on_target']
y = results_corr['team_attack_average_nodedegree']

print(scipy.stats.pearsonr(x, y))


# In[56]:


x = results_corr['attack_attempts_on_target']
y = results_corr['attack_cluster_coefficient']

print(scipy.stats.pearsonr(x, y))


# In[57]:


x = results_corr['teams_attempts_on_target']
y = results_corr['WDL']

print(scipy.stats.pearsonr(x, y))


# In[58]:


x = results_corr['teams_attempts_on_target']
y = results_corr['team_attack_centrality']

print(scipy.stats.pearsonr(x, y))


# # PART 2

# ### England vs Belgium

# In[59]:


#To get a match graph of England vs Belgium
EngvsBel =create_and_show_graph(events,2058016, 2413,'England', 5629, 'Belgium')

print("Number of nodes in this random graph :" ,EngvsBel['graph'].number_of_nodes())
print("Number of edges in this random graph :" ,EngvsBel['graph'].number_of_edges())
print("Clustering Coefficient: %f "%nx.average_clustering(EngvsBel['graph']))
print(nx.info(EngvsBel['graph']))


# In[60]:


#TO get the individual attack graph of team and opponet in a match
eng=[n for (n,team) in nx.get_node_attributes(EngvsBel['graph'],"teamid").items() if team == 2413]
bel=[n for (n,team) in nx.get_node_attributes(EngvsBel['graph'],"teamid").items() if team == 5629]

#To create and show subgraph for england and belgium
eng_graph=create_and_show_subgraph(EngvsBel['graph'],eng,'pink','/Users/Maneesha Majeed/DMSN_results/',('England'+ '_graph'))
bel_graph=create_and_show_subgraph(EngvsBel['graph'],bel,'blue','/Users/Maneesha Majeed/DMSN_results/',('Belgium'+ '_graph'))


# In[61]:


#To create adn show subgraph for england and belgium attack graph


#create team attack graph England
eng_attack_nodes = [n for (n, role) in nx.get_node_attributes(eng_graph,"role").items() if role == 'Forward' or role == 'Midfielder']
```

```
eng_attack_graph = create_and_show_subgraph(eng_graph ,eng_attack_nodes, 'blue','/Users/Maneesha Majeed/DMSN_results/',('England' +'_attack_graph'))


# In[62]:


#create opponent attack  graph - Belgium
bel_attack_nodes = [n for (n, role) in nx.get_node_attributes(bel_graph,"role").items() if role == 'Forward' or role == 'Midfielder']
bel_attack_graph = create_and_show_subgraph(bel_graph ,bel_attack_nodes, 'red','/Users/Maneesha Majeed/DMSN_results/',('Belgium' +'_attack_graph'))




# In[63]:


#eng_attack_graph
graph_degree = nx.degree(eng_attack_graph)
graph_degree_dic = dict(graph_degree)
sum_graph_degree=np.sum(list(graph_degree_dic.values()))
average_nodedegree = (sum_graph_degree)/float(len(eng_attack_graph.nodes()))

print("average_nodedegree=",average_nodedegree)
print("cluster_coefficient =", nx.average_clustering(eng_attack_graph))
print("connectivity =",nx.algebraic_connectivity(eng_attack_graph))
print("density =" ,nx.density(eng_attack_graph))
print("centrality = ",nx.global_reaching_centrality(eng_attack_graph))
print("bet_centrality=",nx.betweenness_centrality(eng_attack_graph))



# In[64]:


#To get the number of nodes and edges
print(eng_attack_graph.number_of_nodes())
print(eng_attack_graph.number_of_edges())
print(bel_attack_graph.number_of_nodes())
print(bel_attack_graph.number_of_edges())
print(nx.info(eng_attack_graph))


# In[65]:


graph_degree=nx.degree(eng_attack_graph)
graph_degree_dic=dict(graph_degree)
#np.mean(list(graph_degree_dictionary.values())
print(float(len(eng_attack_graph.nodes())))
print(list(graph_degree_dic.values()))
print(np.sum(list(graph_degree_dic.values()))/float(len(eng_attack_graph.nodes())))


# In[66]:


graph_degree = nx.degree(eng_attack_graph)
graph_degree_dic = dict(graph_degree)
average_nodedegreee = np.sum(list(graph_degree_dic.values()))/float(len(eng_attack_graph.nodes()))
average_nodedegreee


# ### Degree based statistics = England vs Belgium
#

# In[67]:


#eng_attack_graph
#bel_attack_graph

no_nodes = len(eng_attack_graph.nodes())
```

```
no_edges = len(eng_attack_graph.edges())

degrees_E = [d for n, d in eng_attack_graph.degree()]
degrees_B = [d for n, d in bel_attack_graph.degree()]

avg_deg = 2.0 * no_edges/no_nodes

a4_dims = (11.7, 8.27)
fig, ax = plt.subplots(figsize=a4_dims)

plt.xlabel('Node Degree',fontsize=20)
plt.ylabel('Frequency',fontsize=20)
plt.tick_params(labelsize=20)

plt.axvline(x=avg_deg, linewidth=2, linestyle='--', color='black')

plt.hist(degrees_E,bins=17,label="England Attack Graph",alpha=0.5,color='blue')
plt.hist(degrees_B,bins=17,label='Belgium Attack Graph',alpha=0.5,color='red')
plt.legend(loc='upper right',fontsize=20)
plt.show()


# ### Clustering coefficent = England vs Belgium

# In[68]:


#eng_attack_graph
#bel_attack_graph

clusters_E = list(nx.clustering(eng_attack_graph).values())
avg_cluster_E = sum(clusters_E)/float(no_nodes)
clusters_B = list(nx.clustering(bel_attack_graph).values())
avg_cluster_B = sum(clusters_B)/float(no_nodes)

a4_dims = (11.7, 8.27)

fig, ax = plt.subplots(figsize=a4_dims)

plt.xlabel('Node Clustering Coefficient',fontsize=20)
plt.ylabel('Frequency',fontsize=20)
plt.tick_params(labelsize=20)

plt.hist(clusters_E,bins=20,label="England Attack Graph",alpha=0.5,color='blue')
plt.hist(clusters_B,bins=20,label='Belgium Attack Graph',alpha=0.5,color='red')

plt.axvline(x=avg_cluster_E, linestyle='--', color='blue')
plt.axvline(x=avg_cluster_B, linestyle='--', color='red')

plt.legend(loc='upper centre',fontsize=20)

plt.show()


# ## Part two - random networks vs match graph comparison

# In[69]:


#England vs Belgium match graph
#To get a match graph of England vs Belgium
EngvsBel =create_and_show_graph(events,2058016, 2413,'England', 5629, 'Belgium')

print("Clustering Coefficient: %f "%nx.average_clustering(EngvsBel['graph']))
print(nx.info(EngvsBel['graph']))

#Network statistics of eng vs bel graph
print("cluster coefficient",nx.average_clustering(EngvsBel['graph']))
print("connectivity",nx.algebraic_connectivity(EngvsBel['graph']))
print("Density",nx.density(EngvsBel['graph']))
print("Centrality",nx.global_reaching_centrality(EngvsBel['graph']))
```

```
# In[70]:


#Erdos renyi Graph
#Erdos renyi graph 1
edos1=nx.erdos_renyi_graph(28,0.44)
nx.draw(edos1,with_labels=False,node_color='pink')
plt.title("Erdos Renyi Graph 1",fontsize=10)
plt.show()
#Erdos renyi graph 2
edos2=nx.erdos_renyi_graph(28,0.44)
nx.draw(edos2,with_labels=False,node_color='blue')
plt.title("Erdos Renyi Graph 2",fontsize=10)
plt.show()
#Erdos renyi graph 3
edos3=nx.erdos_renyi_graph(28,0.44)
nx.draw(edos3,with_labels=False,node_color='green')
plt.title("Erdos Renyi Graph 3",fontsize=10)
plt.show()

print(nx.info(edos1))
print("Clustering Coefficient 1: %f "%nx.average_clustering(edos1))

print(nx.info(edos2))
print("Clustering Coefficient 2: %f "%nx.average_clustering(edos2))

print(nx.info(edos3))
print("Clustering Coefficient 3: %f "%nx.average_clustering(edos3))

#To take the average
edges_edos1=edos1.number_of_edges()
edges_edos2= edos2.number_of_edges()
edges_edos3=edos3.number_of_edges()

avg_edges =(edges_edos1+edges_edos2+edges_edos3)/3

avg_cluster_edos = (nx.average_clustering(edos1)+nx.average_clustering(edos2)+nx.average_clustering(edos3))/3
print(avg_cluster_edos)
print(avg_edges)


# ### Task 1 - Differences between path length and clustering coefficients

# For Tasks 1 to 3 below, compare your network to 3 generated random networks (Erdos-Renyi or any other)
# based on the statistics of your dataset, i.e. you will need to maintain the same number of nodes and edges
# (or very close to) as the original dataset.  In answering/doing the below analysis,
# you should perform the task on your dataset and the 3 generated networks
# (the presented results should be the average seen across these 3).
# Task 1. What are the differences between path length and clustering coefficients?

# In[71]:


#To get  cluster coefficients
no_nodes =28
clusters_engvsbel = list(nx.clustering(EngvsBel['graph']).values())
avg_cluster_engvsbel = sum(clusters_engvsbel)/float(no_nodes)



# Average of 3 radnom erdos reny graph : avg_cluster_edos
print(" England vs Belgium match average cluster coefficient is ",avg_cluster_engvsbel)
print(" Erdos renyi 3 random graphs average cluster coefficient is ",avg_cluster_edos)


# In[72]:


# Cluster coefficient graph

cluster_edos=list(nx.clustering(edos1).values())
a4_dims = (8.7, 6.27)
```

```
fig, ax = plt.subplots(figsize=a4_dims)

plt.xlabel('Node Clustering Coefficient',fontsize=10)
plt.ylabel('Frequency',fontsize=10)
plt.tick_params(labelsize=10)

plt.hist(clusters_engvsbel,bins=10,label="England vs Belgium Match Graph",alpha=0.5,color='blue')
plt.hist(cluster_edos,bins=10,label='Erdos Renyi Graph',alpha=0.5,color='red')

plt.axvline(x=avg_cluster_engvsbel, linestyle='--', color='blue')
plt.axvline(x=avg_cluster_edos, linestyle='--', color='red')

plt.legend(loc='upper right',fontsize=10)

plt.show()


# In[73]:


#Path length
def get_path_lengths(graph):
    lengths = []
    for i in range(28):
        for j in range(i):
            lengths.append(len(nx.shortest_path(graph,i,j))-1)
    return lengths

path_lengths_edos1=get_path_lengths(edos1)
path_lengths_edos2=get_path_lengths(edos2)
path_lengths_edos3=get_path_lengths(edos3)


# In[74]:


#path length for EngvsBel
def get_path_lengths(G):
    lengths = []
    all_nodes= list(G.nodes())
    for i in range (len(all_nodes)):
        for j in range(i):
            lengths.append(len(nx.shortest_path(G,all_nodes[i],all_nodes[j]))-1)
    return lengths

path_lengths_EngvsBel = get_path_lengths(EngvsBel['graph'])
print(path_lengths_EngvsBel)
avg_path_EngvsBel = sum(path_lengths_EngvsBel)/float(len(path_lengths_EngvsBel))
print(avg_path_EngvsBel)


# In[75]:


#Erdos renyi graph : edos1=nx.erdos_renyi_graph(28,0.44)
#EngvsBel graph : EngvsBel['graph']

print(nx.average_shortest_path_length(EngvsBel['graph']))
print(nx.average_shortest_path_length(edos1))
print(nx.average_shortest_path_length(edos2))
print(nx.average_shortest_path_length(edos3))
avg_edos_shortest_path_length                                                                                     =
(nx.average_shortest_path_length(edos1)+nx.average_shortest_path_length(edos2)+nx.average_shortest_path_length(edos3))/3
print(avg_edos_shortest_path_length)


# In[76]:


a4_dims = (8.7, 5.27)

fig, ax = plt.subplots(figsize=a4_dims)

plt.xlabel('Path Length',fontsize=10)
plt.ylabel('Frequency',fontsize=10)

plt.hist(path_lengths_EngvsBel,bins=3,label="England vs Belgium",alpha=0.5,color='blue')
plt.hist(path_lengths_edos1,bins=3,label='Erdos-Renyi Graph',alpha=0.5,color='red')

plt.axvline(x=avg_path_EngvsBel, linestyle='--', color='blue')
plt.axvline(x=avg_edos_shortest_path_length, linestyle='--', color='red')

plt.legend(loc='upper right',fontsize=10)

plt.show()


# In[77]:


#network statistics of erdos-renyi graph

gd_edos1= nx.degree(edos1)
edos1_dic = dict(gd_edos1)
sum_edos1=np.sum(list(edos1_dic.values()))
average_nodedegree_edos1 = sum_edos1/float(len(edos1.nodes()))

gd_edos2= nx.degree(edos2)
edos2_dic = dict(gd_edos2)
sum_edos2=np.sum(list(edos2_dic.values()))
average_nodedegree_edos2 = sum_edos2/float(len(edos2.nodes()))

gd_edos3= nx.degree(edos3)
edos3_dic = dict(gd_edos3)
sum_edos3=np.sum(list(edos3_dic.values()))
average_nodedegree_edos3 = sum_edos3/float(len(edos3.nodes()))


average_nodedegree=(average_nodedegree_edos1+average_nodedegree_edos2+average_nodedegree_edos3)/3
cluster_coefficient = (nx.average_clustering(edos1)+nx.average_clustering(edos2)+nx.average_clustering(edos3))/3
connectivity =(nx.algebraic_connectivity(edos1)+nx.algebraic_connectivity(edos2)+nx.algebraic_connectivity(edos3))/3
density = (nx.density(edos1)+nx.density(edos2)+nx.density(edos3))/3
centrality =(nx.global_reaching_centrality(edos1)+nx.global_reaching_centrality(edos2)+nx.global_reaching_centrality(edos3))/3


print("cluster coefficient", cluster_coefficient)
print("connectivity", connectivity)
print("Density",density)
print("Centrality",centrality)
print("Average Node degree",average_nodedegree)


# # Task 2 - Robustness comparison

# Task 2. Which is more robust? To do this, you should remove edges based on tie strength (weak/strong) or some other measure (edge weight), or remove nodes (e.g. based on node degree), and plot what happens to the size of the giant component as you remove more and more edges/nodes.

# In[79]:


# Graph to use
engVsBel = create_and_show_graph(events,2058016, 2413,'England', 5629, 'Belgium')
engVsBel = engVsBel['graph']
print('Numbers of nodes: ',engVsBel.number_of_nodes())
print('Number of edges: ',engVsBel.number_of_edges())


# In[80]:


# Function to find the giant component in Graph

def giant_component(G):
    largest_components= max(nx.connected_components(G), key=len)
    giant_comp_graph = (G).subgraph(largest_components)

    #draw graph
#     a4_dims = (20, 20)
#     fig, ax = plt.subplots(figsize=a4_dims)
#     ax.set_axis_off()
#     pos = nx.kamada_kawai_layout(giant_comp)
#     nx.draw_networkx_edges(giant_comp_graph, pos, with_labels=False,edge_color="pink", width=6.0)
#     nx.draw_networkx_nodes(giant_comp_graph, pos, node_color='red', alpha=0.4, with_labels=False)
#     nx.draw_networkx_edges(giant_comp_graph, pos, with_labels=False,alpha=0.2,width=3)

    return giant_comp_graph


# In[81]:


# Function to plot giant component number of nodes vs giant component size

def giant_component_sizes(G):
    giant_component_sizes = {}
    no_of_nodes_removed = 0
    original_no_nodes= G.number_of_nodes()
    original_largest_component_size = (giant_component(G)).size()

    while G.number_of_nodes() > 0:

        #get giant component size and save it
        giant_component_graph= giant_component(G)
        component_size = giant_component_graph.size()
        fraction_of_nodes_removed = no_of_nodes_removed/original_no_nodes
        giant_component_sizes[fraction_of_nodes_removed]=component_size/original_largest_component_size

        # get node with highest degree
        degrees=nx.degree(G)
        highest_degree_node = max(degrees, key=lambda x: x[1])
        # remove the highest degree node
        G.remove_node(highest_degree_node[0])
        no_of_nodes_removed = no_of_nodes_removed+1

    return giant_component_sizes


# In[82]:


# England VS Belgium
engVsBel_component_sizes = giant_component_sizes(engVsBel)

# Creating all Random graphs and getting their giant component sizes

#Erdos Renyi Graph 1
edos1= nx.erdos_renyi_graph(28,0.44)
nx.draw(edos1, with_labels=False)
plt.show()
erdos1_component_sizes = giant_component_sizes(edos1)

#Erdos Renyi Graph 2
edos2= nx.erdos_renyi_graph(28,0.44)
nx.draw(edos2, with_labels=False)
plt.show()
erdos2_component_sizes = giant_component_sizes(edos2)

#Erdos Renyi Graph 3
edos3= nx.erdos_renyi_graph(28,0.44)
nx.draw(edos3, with_labels=False)
plt.show()
erdos3_component_sizes = giant_component_sizes(edos3)

plt.plot(list(engVsBel_component_sizes.keys()),list(engVsBel_component_sizes.values()), label = "Eng Vs Bel")
plt.plot(list(erdos1_component_sizes.keys()),list(erdos1_component_sizes.values()), label = "Erdos graph 1")
plt.plot(list(erdos2_component_sizes.keys()),list(erdos2_component_sizes.values()), label = "Erdos graph 2")
plt.plot(list(erdos3_component_sizes.keys()),list(erdos3_component_sizes.values()), label = "Erdos graph 3")



plt.legend()
plt.title('Network Robustness')
plt.ylabel('Fraction of Giant component size')
plt.xlabel('Fraction of nodes removed ')
plt.show()


# In[83]:


#Average component size values of three random graphs
y=[]
for key in erdos1_component_sizes:
    average_component_size = (erdos1_component_sizes[key] + erdos2_component_sizes[key] + erdos3_component_sizes[key])/3
    y.append(average_component_size)

plt.plot(list(engVsBel_component_sizes.keys()),list(engVsBel_component_sizes.values()), label = "Eng Vs Bel")
plt.plot(list(erdos1_component_sizes.keys()),y, label = "Average of 3 graph")
plt.legend()
plt.title('Network Robustness')
plt.ylabel('Fraction of Giant component size')
plt.xlabel('Fraction of nodes removed ')
plt.show()


# # Task 3 PageRank Analysis

# Task 3. Calculate the pageranks for the network related to your dataset and the 3 generated random graphs. Compare the pageranks of the dataset network with the average pageranks of the 3 generated random graphs (present and discuss a graph or histogram).

# In[84]:


#Erdos renyi Graph for PageRanks

#Erdos renyi graph 1
edos1=nx.erdos_renyi_graph(14,0.75)
nx.draw(edos1,with_labels=False,node_color='red')
plt.title("Erdos Renyi Graph 1",fontsize=20)
plt.show()
#Erdos renyi graph 2
edos2=nx.erdos_renyi_graph(14,0.75)
nx.draw(edos2,with_labels=False,node_color='blue')
plt.title("Erdos Renyi Graph 2",fontsize=20)
plt.show()
#Erdos renyi graph 3
edos3=nx.erdos_renyi_graph(14,0.75)
nx.draw(edos3,with_labels=False,node_color='green')
plt.title("Erdos Renyi Graph 3",fontsize=20)
plt.show()

print("Number of nodes in this graph 1:" ,edos1.number_of_nodes())
print("Number of edges in this graph1 :" ,edos1.number_of_edges())

print("\nNumber of nodes in this graph 2:" ,edos2.number_of_nodes())
print("Number of edges in this graph 2:" ,edos2.number_of_edges())

print("\nNumber of nodes in this graph 3:" ,edos3.number_of_nodes())
print("Number of edges in this graph 3 :" ,edos3.number_of_edges())

print("\nNumber of nodes in the England's graph :" ,eng_graph.number_of_nodes())
print("Number of edges in the England's graph :" ,eng_graph.number_of_edges())

print("\nNumber of nodes in the Belgium's graph :" ,bel_graph.number_of_nodes())
print("Number of edges in the Belgium's graph :" ,bel_graph.number_of_edges())

#To take the average
edges_edos1=edos1.number_of_edges()
edges_edos2= edos2.number_of_edges()
edges_edos3=edos3.number_of_edges()

avg_edges =int((edges_edos1+edges_edos2+edges_edos3)/3)

print("\nNumber of nodes in this Random graphs is:", edos1.number_of_nodes())
print("Average NO. of Edges of the Random graphs is:",avg_edges)
```

```
# In[85]:


#Average of the 3 ER graphs' PageRank

Erdos1=nx.pagerank(edos1, alpha=0.9)
Erdos2=nx.pagerank(edos2, alpha=0.9)
Erdos3=nx.pagerank(edos3, alpha=0.9)

result = {key: Erdos1.get(key, 0) + Erdos2.get(key, 0) + Erdos3.get(key, 0) for key in set(Erdos1) | set(Erdos2) | set(Erdos3)}
for key in result:
    result[key] /=  3

result_sort = sorted(result.items(), key=lambda x: x[1], reverse=True)


# In[86]:


#Belgium's PageRank

a = nx.pagerank(bel_graph, alpha=0.9)
matchRank_bel = sorted(a.items(), key=lambda x: x[1], reverse=True)


# In[87]:


#England's PageRank

a = nx.pagerank(eng_graph, alpha=0.9)
matchRank_eng = sorted(a.items(), key=lambda x: x[1], reverse=True)


# In[88]:


#average PageRank of ER graphs

sum1 = 0
avg = 0
for i in range(3):
    edos=(edos1, edos2, edos3)
    a=nx.pagerank(edos[i], alpha=0.9)
    s = sorted(a.items(), key=lambda x: x[1], reverse=True)

    # for k, v in s:
    #     print(k, v)
    #average of pagerank for real AD network
    p=0
    for d in a.values():
        p=p+d
    avg = p/len(a)
    print('Average of pagerank for Edos{}: {}'.format(i, avg))
    sum1 += avg

print("\nAverage PageRank of all the Random Graphs is:{}".format(sum1/3))
edosRank= p/len(a)


# In[89]:


#England's average PageRank

a=nx.pagerank(eng_graph, alpha=0.9)
s = sorted(a.items(), key=lambda x: x[1], reverse=True)
p=0
for d in a.values():
    p=p+d
print('Average of pagerank for England is: ',p/len(a))
```

```
# In[90]:


#The most and least influencial players of the England and Belgium respectively.

topRank_bel = nx.get_node_attributes(bel_graph,'shortName')[matchRank_bel[0][0]]
topRank_eng = nx.get_node_attributes(eng_graph,'shortName')[matchRank_eng[0][0]]

botRank_bel = nx.get_node_attributes(bel_graph,'shortName')[matchRank_bel[13][0]]
botRank_eng = nx.get_node_attributes(eng_graph,'shortName')[matchRank_eng[13][0]]

print("The highest ranked player in Belgium is ->", topRank_bel)
print("The lowest ranked player in Belgium is ->", botRank_bel)


print("\nThe highest ranked player in England is ->", topRank_eng)
print("The lowest ranked player in England is ->", botRank_eng)


# # Task 4 Cascade

#
# Task 4. [This task has to be conducted on your dataset network only, not on the generated random graphs]. How cascade friendly is your network? Explore 2 different seed nodes (e.g. lowest degree centrality and direct neighbor, a node with average degree centrality and direct neighbor, highest degree centrality and direct neighbor, a random node and direct neighbor), or 2 different fractions of reached/infected nodes, in combination with 2 different thresholds (e.g. 0.1, 0.25, 0.5, 0.75), to determine how information flows/disease spreads through your network. Are there areas which are more likely to cause cascades than others when the initial reached/infected nodes belong to them? You can report the percentage or number of nodes reached/infected for each setting (e.g. table or graph).

# In[16]:


# to create a directed graph
def create_and_show_graph(events,matchId, teamOneId,teamOneName, teamTwoId, teamTwoName):

    #Only interested in the events which happened in the Cup Final, with matchId 2058017.
    wc_final_events = events[events['matchId']==matchId]
    events = wc_final_events.reset_index(drop=True)

    #initialize the graph
    G = nx.DiGraph()
    teamColours = {teamOneId: 'red', teamTwoId: 'blue'}

    #Create Graph
    for i in range(len(events)-1):
        if (events['subEventName'][i]=="Simple pass") and len(events['tags'][i])==1:
            n1, n2 = events['playerId'][i], events['playerId'][i+1]
            p1, p2 = list(players[players['wyId']==n1]['shortName'])[0], list(players[players['wyId']==n2]['shortName'])[0]
            t1, t2 = events['teamId'][i], events['teamId'][i+1]
            r1, r2 = list(players[players['wyId']==n1]['role/name'])[0], list(players[players['wyId']==n2]['role/name'])[0]
            e1, e2 = events['eventSec'][i], events['eventSec'][i+1]
            t = events['tags'][i]
            if n1 not in G.nodes():
                G.add_node(n1, teamid=t1, role=r1, color=teamColours[t1], shortName=p1)
            if n2 not in G.nodes():
                G.add_node(n2, teamid=t2, role=r2, color=teamColours[t2], shortName=p2)
            if any([row['id']==1801 for row in t]):
                G.add_edge(n1,n2, weight=e1)




    #add graph node attributes
    player_information = {}

    for n in G.nodes():
        playerName = nx.get_node_attributes(G,'shortName')[n]
        playerRole = nx.get_node_attributes(G, 'role')[n]
        player_information[n] = (playerName + ' : ' + playerRole).encode().decode('unicode_escape')


    e_labels = nx.get_edge_attributes(G,'weight')
    #Create graph nodes list
    teamOneNodes = [n for (n, team) in nx.get_node_attributes(G,"teamid").items() if team == teamOneId]
    teamTwoNodes = [n for (n, team) in nx.get_node_attributes(G,"teamid").items() if team == teamTwoId]

    matchDegree = nx.degree(G)
    #Show graph
    a4_dims = (50, 50)
    fig, ax = plt.subplots(figsize=a4_dims)
    ax.set_axis_off()

    pos = nx.kamada_kawai_layout(G)
    nx.draw_networkx_nodes(G, pos, nodelist=teamOneNodes, node_color='red', alpha=0.4, labels=player_information, with_labels=True, node_size=[v *
200 for v in dict(matchDegree).values()])
    nx.draw_networkx_nodes(G, pos, nodelist=teamTwoNodes, node_color='blue', alpha=0.4, labels=player_information, with_labels=True, node_size=[v *
200 for v in dict(matchDegree).values()])
    nx.draw_networkx_edges(G, pos, with_labels=True, alpha=0.2,width=4, edge_labels=e_labels)
    nx.draw_networkx_labels(G, pos, labels= player_information, font_size=20)

    # create folder based on match

    rootPath= '/Users/syedanarmeen/Documents/DSMN/'
    folderName = teamOneName + ' vs ' + teamTwoName + ' matchId ' + str(matchId)
    _dir = os.path.join(rootPath, folderName)

    # create 'dynamic' dir, if it does not exist
    if not os.path.exists(_dir):
        os.makedirs(_dir)

    plt.savefig(_dir + '/match_graph.pdf')
    plt.title(teamOneName + ' vs ' + teamTwoName, fontsize=20)
#     plt.show()

    return {'graph':G, 'folderName': _dir}


# In[19]:


# To get directed graph of england to do manual cascade anaylsis

engVsBel = create_and_show_graph(events,2058016, 2413,'England', 5629, 'Belgium')
engVsBel = engVsBel['graph']

england_nodes = [n for (n,team) in nx.get_node_attributes(engVsBel,"teamid").items() if team == 2413]
england_graph = create_and_show_subgraph(engVsBel,england_nodes,'pink', '/Users/syedanarmeen/Documents/DSMN',('bel_graph'))


# In[ ]:
```