# DS210 Final Project Report
***By Seokhoon Shin***

Analyzing the Premier League Network (2006/2007 to 2017/2018)

Dataset: Premier League Results and Stats

## 1. Introduction

As a passionate soccer enthusiast and data analysis enthusiast, I embarked on a fascinating journey to explore the dynamics of the Premier League. This project involves analyzing a dataset spanning the results and statistics from the 2006/2007 season to the 2017/2018 season. The Premier League has always captivated football enthusiasts worldwide, and this dataset provided an excellent opportunity to apply network analysis techniques using Rust. The goal was to compute centrality measures for the graph composed of Premier League teams as nodes and matches as edges. The project merges my love for soccer with my interest in data analysis, aiming to unveil hidden patterns within the league's structure that may not be immediately apparent through traditional statistics.

## 2. Project Overview

In this project, I designed a clear structure with several key components:

- main.rs: Serving as the entry point, this module is the foundation of the project.
- network_centrality.rs: This module houses functions for computing betweenness centrality using Brandes' algorithm.
- graph_construction.rs: It includes functionality for constructing the graph structure from the dataset.
- I imported essential components from the petgraph crate, a Rust library for graph data structures and algorithms.

## 3. Network Analysis Components

network_centrality.rs:

- calculate_betweenness_centrality: This function calculates betweenness centrality for each node in the graph using Brandes' algorithm.
- find_shortest_paths: It finds the shortest paths from a given start node to all other nodes using Dijkstra's algorithm.

graph_construction.rs:

- I designed a custom match struct representing a match with various fields, including home_team, away_team, home_goals, away_goals, result, and season. This struct also includes a method match_weight to determine edge weights.
- match_weight function calculates edge weights based on match results and goals.
- The construct_graph function reads data from a CSV file, constructs a directed graph, and assigns weights to edges based on match results and goals.

**4. Analysis Insights**

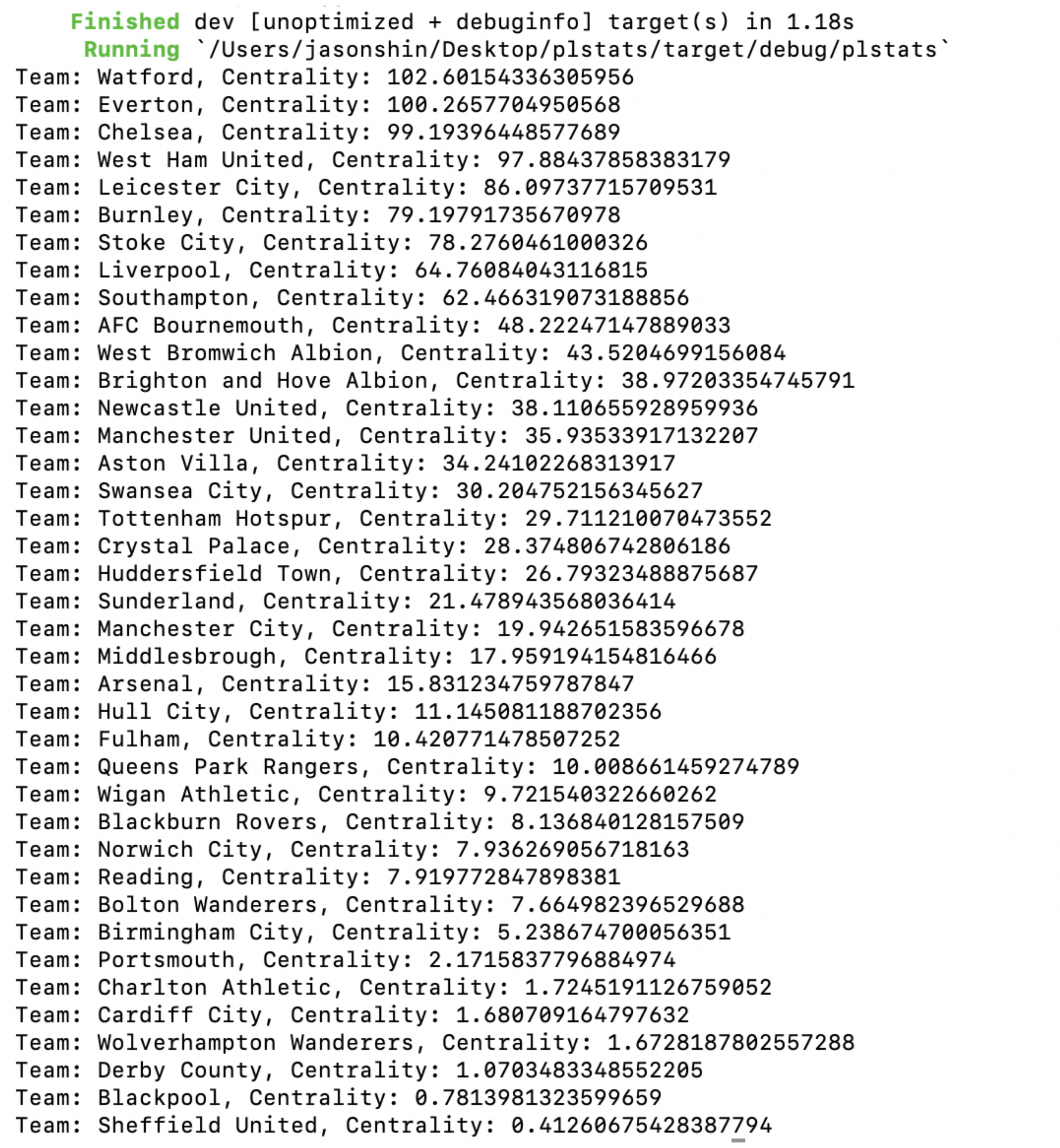
One of the most intriguing discoveries from this analysis was the revelation that Watford emerged with the highest betweenness centrality score of 102.60. This indicates that, within the network of Premier League teams, Watford is a highly influential entity. Such high centrality suggests that Watford, perhaps unexpectedly, plays a central role in the league's structure, significantly affecting the flow of the season's progress. Notable teams like Everton and Chelsea also exhibit high centrality scores, indicative of their significant roles within the network. These findings suggest that a team's impact on the league's dynamics is not solely based on media visibility or historical prestige but can also be evaluated by its strategic influence on the connectivity between teams. In the competitive fabric of the Premier League, teams like Watford can be just as pivotal in shaping the season's narrative as more commonly acclaimed squads.

**5. Dijkstra's Algorithm for the Dataset**

I chose to use the Betweenness Centrality measure algorithm based on Dijkstra's shortest path algorithm for this project for several reasons:

- Weighted Graph: The dataset includes match results with different outcomes (wins, losses, draws), making it essential to consider edge weights. Dijkstra's algorithm naturally supports weighted graphs.
- Importance of Connections: The Betweenness Centrality measure helps identify nodes (teams) with a significant impact on the graph's connectivity. Teams with high betweenness centrality can have a crucial influence on other teams' performance and connectivity within the league.
- Shortest Paths: To calculate Betweenness Centrality, we need to find the shortest paths between all pairs of nodes. Dijkstra's algorithm efficiently computes these paths, especially for sparse graphs like the Premier League network.
- Complexity: Despite the dataset containing results from 4560 Premier League matches (380 matches across 12 seasons from 2006/2007 to 2017/2018), equating to 9120 edges, and 9121 nodes, Dijkstra's algorithm remains effective with a complexity of O(18241*log(9121)).
- Directionality: Premier League matches are directed (team A vs. team B), and Dijkstra's algorithm handles directed graphs effectively.

## 6. Output - Betweenness Centrality



```
     Finished dev [unoptimized + debuginfo] target(s) in 1.18s
      Running `/Users/jasonshin/Desktop/plstats/target/debug/plstats`
Team: Watford, Centrality: 102.60154336305956
Team: Everton, Centrality: 100.2657704950568
Team: Chelsea, Centrality: 99.19396448577689
Team: West Ham United, Centrality: 97.88437858383179
Team: Leicester City, Centrality: 86.09737715709531
Team: Burnley, Centrality: 79.19791735670978
Team: Stoke City, Centrality: 78.2760461000326
Team: Liverpool, Centrality: 64.76084043116815
Team: Southampton, Centrality: 62.466319073188856
Team: AFC Bournemouth, Centrality: 48.22247147889033
Team: West Bromwich Albion, Centrality: 43.5204699156084
Team: Brighton and Hove Albion, Centrality: 38.97203354745791
Team: Newcastle United, Centrality: 38.110655928959936
Team: Manchester United, Centrality: 35.93533917132207
Team: Aston Villa, Centrality: 34.24102268313917
Team: Swansea City, Centrality: 30.204752156345627
Team: Tottenham Hotspur, Centrality: 29.711210070473552
Team: Crystal Palace, Centrality: 28.374806742806186
Team: Huddersfield Town, Centrality: 26.79323488875687
Team: Sunderland, Centrality: 21.478943568036414
Team: Manchester City, Centrality: 19.942651583596678
Team: Middlesbrough, Centrality: 17.959194154816466
Team: Arsenal, Centrality: 15.831234759787847
Team: Hull City, Centrality: 11.145081188702356
Team: Fulham, Centrality: 10.420771478507252
Team: Queens Park Rangers, Centrality: 10.008661459274789
Team: Wigan Athletic, Centrality: 9.721540322660262
Team: Blackburn Rovers, Centrality: 8.136840128157509
Team: Norwich City, Centrality: 7.936269056718163
Team: Reading, Centrality: 7.919772847898381
Team: Bolton Wanderers, Centrality: 7.664982396529688
Team: Birmingham City, Centrality: 5.238674700056351
Team: Portsmouth, Centrality: 2.1715837796884974
Team: Charlton Athletic, Centrality: 1.7245191126759052
Team: Cardiff City, Centrality: 1.680709164797632
Team: Wolverhampton Wanderers, Centrality: 1.6728187802557288
Team: Derby County, Centrality: 1.0703483348552205
Team: Blackpool, Centrality: 0.7813981323599659
Team: Sheffield United, Centrality: 0.41260675428387794
```

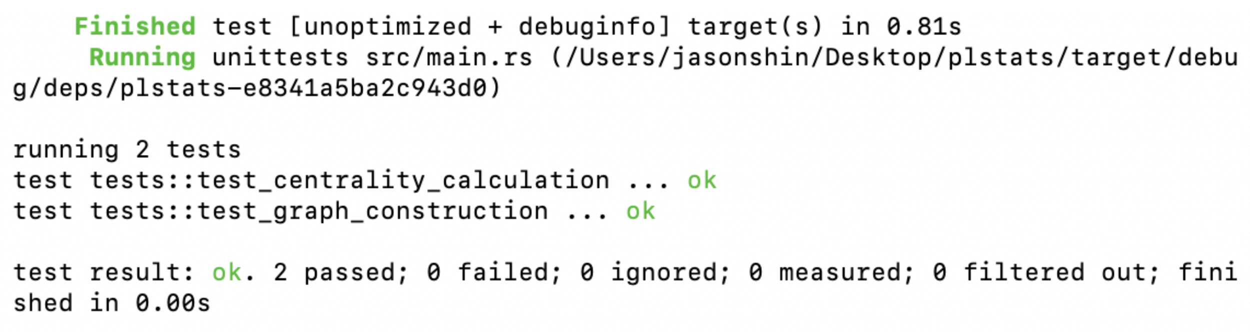
Betweenness centrality measures the extent to which a node (in this case, a football team) stands on the shortest path between other nodes. A node with high betweenness centrality has a greater 'control' over the network because more shortest paths pass through it.

Teams like Watford, with surprisingly high centrality scores, seem to have a significant impact on the Premier League network. This suggests that they are involved in a large number of matches crucial for other teams to advance in the league, regardless of their position in the standings. Conversely, teams with minimal centrality scores, like Sheffield United, have matches that influence the league's structure to a lesser extent. Such insights are valuable for understanding the strategic importance of matches throughout the season beyond wins and losses, highlighting the interconnectedness of the league.

## 7. Conclusions

Utilizing Dijkstra's algorithm to compute Betweenness Centrality, this project uncovered the hidden influencers of the Premier League. Watford's high centrality score was a revelation, demonstrating their considerable impact on the league's connectivity. This insight is valuable for predicting potential league outcomes and underscores how teams contribute to the league beyond their direct competition. This project highlights the importance of looking beyond the standings and showcases the analytical power of network theory in sports analytics.

## 8. Test



```
    Finished test [unoptimized + debuginfo] target(s) in 0.81s
     Running unittests src/main.rs (/Users/jasonshin/Desktop/plstats/target/debu
g/deps/plstats-e8341a5ba2c943d0)

running 2 tests
test tests::test_centrality_calculation ... ok
test tests::test_graph_construction ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; fini
shed in 0.00s
```

## 9. Resources

- petgraph Documentation
- DS210 lectures (28, 30)
- tempfile Documentation
- std::io::Write Documentation
- graphrs Crate
- Dijkstra's Algorithm on Wikipedia
- petgraph::prelude Documentation
- petgraph::graph::Graph Documentation

## 10. Acknowledgments

I would like to express my gratitude to the creators of the Premier League dataset, the Rust programming community, and the instructors of DS210 for their valuable resources and support throughout this project.