March 2019

Graph-Based Sports Rankings

Abigail Rose Roane
Worcester Polytechnic Institute

Chaiwat Ekkaewnumchai
Worcester Polytechnic Institute

Connor William McNamara
Worcester Polytechnic Institute

Kyle Richards
Worcester Polytechnic Institute

GRAPH-BASED SPORTS RANKINGS

Major Qualifying Project

Advisors:

CRAIG E WILLS
GÁBOR N SÁRKÖZY

Written By:

CHAIWAT EKKAEWNUMCHAI
CONNOR WILLIAM MCNAMARA
KYLE LIPFERT RICHARDS
ABIGAIL ROSE ROANE

WPI

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

MQP CEW - 1902

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.

MARCH 2019

Abstract

In this project, we developed an approach to sports rankings that reflects the strength of each team while accounting for game results. We implemented a dynamic programming algorithm that creates an optimal ranking for roughly thirty teams. We designed approximation algorithms that formulate near-optimal rankings in polynomial time. Finally, we created postprocess algorithms that determine the best possible ranking, given multiple equivalent optimal rankings. Our rankings remain true to actual game results, ultimately making them better than existing rankings.

ACKNOWLEDGEMENTS

We would like to thank Professors Craig Wills and Gábor Sárközy for their indispensable support as our advisors. Without their advice and expertise, this work would not have been possible.

This research was performed using computational resources supported by the Academic & Research Computing group at Worcester Polytechnic Institute.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Rankings are used by all types of sports enthusiasts - die hard fans argue over the ordering of certain teams, while sports writers spend time determining the newest edition of their “power rankings." Far too often, existing rankings seem to ignore the actual results of games played between teams. For example, in the 2018 NFL season, the Patriots defeated the Chiefs in Week 6 of the season in a tight game. Accordingly, that week, ESPN ranked the Patriots second, one spot ahead of the Chiefs. However, in Week 7, after both teams won against lower ranked opponents, ESPN ranked the Chiefs second, one spot ahead of the Patriots [1]. How could this ranking possibly reflect the actual outcomes of games played if it so blatantly ignored the results?

This project involved the algorithmic creation of rankings that honor game results as much as possible. In most leagues, there are teams that conflict with each other when being ranked. For example, if Team A beat Team B, Team B beat Team C, and Team C beat Team A, who should be ranked highest? No matter what, at least one result will be ignored. In most cases these conflicts cannot be completely resolved, but they can be minimized. Our goal was to create a ranking that minimizes the number of times a team was ranked below a team they beat.

To achieve this goal, we developed multiple algorithms all based around graph theory. Some of our algorithms find the best possible ranking to honor game results, while others find one close to the best. Further, we programmatically implemented these algorithms to actually create rankings using real data. We were able to find an optimal ranking for leagues with 32 teams or less in under an hour, checking billions of possible combinations. Taking recency of games and margin of victory into account, we found that the rankings our methods create are similar to (and arguably better than) real world rankings from sources such as ESPN and Sports Illustrated.

The following chapters provide an outline of our project: the background concepts used, the design of our algorithms, and how we actually implemented and ran the algorithms. Finally, our results are shared and discussed for accuracy, factors to consider, and timing trade-offs.

CHAPTER 2

BACKGROUND

2.1 Existing Sports Rankings

Rankings in sports are a common method to determine which teams or competitors are better than others. While there are many different systems of creating rankings, they all serve the purpose of sharing which teams are better than others. The modern concept of these systems date back to the late 1920's [2], when people attempted to predict the national champions in college football. Since then, many new forms of rankings have been developed, especially with the advent of cheap, powerful computers. Additionally, more and more data is being collected from games, leading to different factors being used in many of these rankings.

2.1.1 Considerations

Ranking systems often consider a number of factors. The most common factors considered are point differential, strength of schedule, and game recency. However, many ranking systems are created by aggregating the opinions of a committee of analysts or journalists. In such cases, these factors are considered implicitly rather than being an explicit metric as in algorithmic ranking systems.

2.1.1.1 Point Differential

Many differences in sports ranking systems come from how heavily they weigh point differential in their rankings. **Point differential** is the difference in the score between a team and their opponent [3]. Using point differential allows the ranking system to understand whether a game was close or a blowout.

However, point differential is also a controversial metric [4]. This is because if point differential is considered, then it incentivizes scoring more points in blowout games, which in many sports is considered unsportsmanlike. Additionally, many sports have

"garbage time," which is when the game outcome has already been decided due to a lopsided score, but the players still have to finish the game. During garbage time teams may not be trying as hard as usual, leading to scoring that would not have happened had the game been close.

2.1.1.2 Strength of Schedule

Strength of schedule is a measurement of the difficulty of a team's schedule. There are many different methods of determining strength of schedule, but most of them work by considering the general ranking and/or record of the team's opponents. As an example, prior to the introduction of the College Football Playoff, NCAA Football calculated strength of schedule for bowl selection using the following formula:

$$\text{Strength of Schedule} = (2 * P) + Q \tag{2.1}$$

In this formula, P represents the combined record of the team's opponents, and Q represents the combined record of their opponent's opponents. The resulting value is then used to rank a team's schedule against those of other teams, helping to differentiate them when creating rankings. Utilizing strength of schedule, a loss against a much stronger opponent is less impactful on a team's ranking compared to a loss against a closely-ranked or lower-ranked team.

2.1.1.3 Game Recency

Game recency is the time elapsed since a game was played. This metric is important, since teams often change throughout a season as players are injured, traded, etc. As a result, many ranking systems weigh games based on their recency, such that a game at the beginning of the season is considered less important than a game which happened much later.

2.1.2 Traditional Ranking Algorithms

2.1.2.1 Win/Loss

A win/loss system of ranking is the simplest method of producing a reasonable ranking. By ordering teams based upon their overall record, teams can be arranged in a way that gives a reasonable ranking. In some cases, strength of schedule can be used to improve the accuracy of the rankings.

Win/loss systems are used in most professional sports to determine playoff berths and seeding. The system benefits from being simple to understand, but often is seen as an incomplete metric that does not necessarily generate a true ranking.

2.1.2.2 Elo Rating

The Elo rating system is a method of ranking competitors based on their past performance. Devised by Arpad Elo [5], it was first used as a way to rank chess players, and to predict future matchups of players who had not previously played. More recently, it has gained significant popularity outside of chess, most notably in other board games and competitive video games as a way to find similarly-skilled players for fair matchmaking. Additionally, some traditional sports analysts use Elo Ratings to create their own rankings [6].

Elo's rating system works by assigning all new competitors a base rating. After each match, a competitor's rating will increase or decrease by a set amount determined by comparing their rating to their opponent's. Their opponent's rating will change by the equal but opposite amount. The size of these changes, along with the base rating, differ across implementations of the Elo system.

The Elo system is useful since it is self correcting: that is, as a competitor plays more matches, their ranking will more closely reflect their true skill. In this way, the Elo system produces ratings that, given an infinite number of games, creates an ideal ranking when sorting all competitors by their rating [7].

2.1.2.3 Committee Rankings

Committee rankings are frequently used as an alternative to algorithmic ranking systems. Rankings by committee are often based on a poll or the choices of a selection committee. Selection Committees are most notably used in NCAA Football and Basketball, where they are used to determine the teams and seeding for the end-of-season playoffs [8],[9]. These committees are given metrics on each team and tasked with deciding on which teams will make or miss these playoffs. By contrast, poll rankings are generated by aggregating the opinions of many people, most commonly journalists or analysts. These rankings are typically used during a season to track a team's progress, and are commonly referred to as **power rankings**.

2.2 Factors in Ranking Sports

In order to effectively create rankings for many different sports, one must consider nuances that apply to each sport. Different sports have different factors that play into rankings. Some factors affecting rankings include number of teams, number of games, and scoring systems. The focus of this project has been six specific sports leagues: NCAA Basketball, NCAA Football, the National Football League (NFL), Major League Baseball (MLB), the National Basketball Association (NBA), and the National Hockey League (NHL).

2.2.1 NCAA Basketball

NCAA Basketball is a college level men's basketball league. There are 351 Division 1 teams in this league, and over 5,000 games are played by these teams in each season. Division 1 is split up into 32 different conferences, mainly separated by regional location. In terms of scoring, teams tend to earn an average of 70 to 100 points per game [10]. Due to the considerable number of teams in this league, there is a large discrepancy between the talent level of the top and bottom teams. Therefore, point differentials vary significantly from game to game.

2.2.2 NCAA Football

NCAA Football is a college level men's football league. There are 250 Division 1 teams in this league, which are further broken down into the Football Bowl Subdivision (FBS) and the Football Championship Subdivision, the former having 130 teams in 2018. The FBS abides by more stringent rules than the FCS, and its teams end their seasons with bowl games, which ultimately determine the consensus Division 1 champion. The FBS subdivision is considered the stronger of the two subdivisions, and its championship team is considered the true winner of the division. Therefore, this project will only focus on ranking the FBS teams [11]. By subdividing Division 1, the NCAA decreased the talent disparity between two competing football teams in the same subdivision.

2.2.3 NFL

The NFL is a professional men's American football league. There are 32 teams in this league. A total of 256 games are played each season, with each team playing 16 games. There are two conferences, each with 16 teams. Within each conference, there are

four divisions, each with four teams. Each team plays six games within their division, facing each team in their division twice. Each team also plays six non-divisional games within their conference, as well as four games outside of their conference, all against the same division [12]. In terms of scoring, most teams earn 20 to 30 points per game in this league [13].

2.2.4 MLB

MLB is a professional men's baseball league. There are 30 teams in the league, which are split into two leagues. Each league is broken down into three divisions, with five teams per division. Each team plays 162 games in a season. A team plays 76 of these games against teams within the same division, 19 per team. A team plays an additional 66 games against teams within the same league and different division. A team also plays 20 games outside of their division [14]. Each team scores an average of three to six runs (points) per game [15].

2.2.5 NBA

The NBA is a professional men's basketball league. There are 30 teams in the league, which is split into two conferences containing 15 teams each. Each conference has three divisions. There are 1230 total games in the season, and each team plays 82 games. Each team plays every other team in the league at least twice in a season, and plays teams within their division four times [16]. Scoring averages at about 100 to 120 points per game [17].

2.2.6 NHL

The NHL is a professional men's hockey league. There are currently 31 teams in the league, which is split into two conferences, one with 15 teams and the other with 16 teams. Each conference has two divisions. There are 1271 total games in the season, and each team plays 82 games. Each team plays every other team in the league at least twice in a season, and plays teams within their division four or five times [18]. Most teams score around one to three goals (points) per game [19].

2.3 Graph Theory

2.3.1 Introduction to Graphs

In computer science, a **graph** is a convenient data structure consisting of **nodes** and **edges** between those nodes. An example graph is shown in Figure 2.1, where A, B, C, and D are nodes and the lines between them are edges. One might use a graph like this to represent flights between cities - nodes represent cities and edges represent existing flights between those cities.

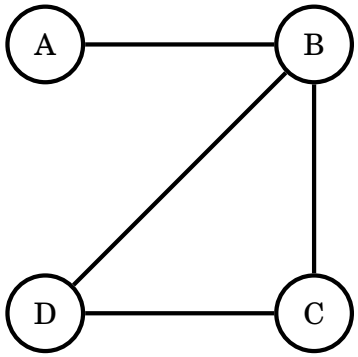

Figure 2.1: *Example Graph*

Graphs are defined by (V, E), where V is the set the set of vertices or nodes, and E is the set of edges between these vertices.

There are two different ways to represent edges: undirected or directed. In the graph in Figure 2.1, all of the edges are undirected, meaning they can be traversed from either vertex. Node B can be reached from node C, and node C can be reached from node B. However, there are scenarios where this would not represent the data accurately - consider the case where there is a flight from city B to city C, but no flight from city C back to city B. In this case, there would be a directed edge from node B to node C, as shown below in Figure 2.2. A directed graph is also known as a **digraph**.

There is a **path** from a starting node (s) to a terminal node (t) if t can be reached from s, even if other nodes must be visited along the way [20]. In Figure 2.2 there is a path from node A to node C. Starting at node A, we go to node B, and then to node C.

A **weighted graph** is a graph whose edges have values (weights) assigned to them. If there are no weights, then the graph is an unweighted graph. Figure 2.2 is an unweighted graph, and Figure 2.3 is a weighted graph. These weights generally signify more information about the data being represented. In the flight example, the edges can signify how many hours the flight takes.

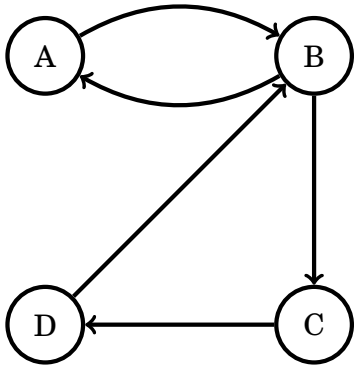

Figure 2.2: *A Directed Graph*

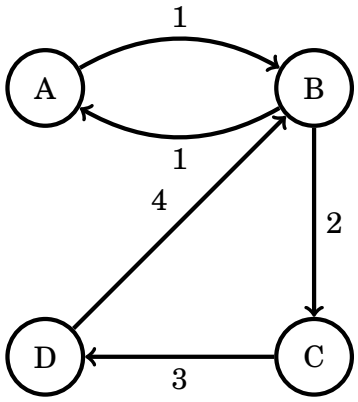

Figure 2.3: *A Directed and Weighted Graph*

Graphs can further be categorized by the frequency of edges. A **complete graph** has the maximum amount of edges: every node is directly connected to every other node [20]. A graph is considered “dense" when the number of edges in the graph is close to the maximum amount of edges. Conversely, a graph is considered “sparse" when it has relatively few edges. There is no precise technical definition of a graph’s density, and graph categorization often depends on the context it is viewed in.

A **connected graph** has an undirected path joining each pair of vertices. A **strongly connected graph** is a directed graph where there is a directed path from every node to every other node [20]. Figure 2.2 is an example of a strongly connected graph.

There are **cycles** in a graph if a node can reach itself by traversing through at least one other node. In Figure 2.3, there is a cycle between nodes A and B, and another one between nodes B, C, and D. A graph is called **acyclic** if there are no cycles in the graph.

Graphs can be separated into **components** in order to categorize sections of the graph. These components are smaller parts of the full graph, also known as subgraphs,

which retain the properties of graphs. For example, a component can by cyclic or strongly connected, or have any other graphic property.

2.3.2 Representing Graphs

Graphs are often represented by two dimensional matrices called **adjacency matrices**. If the value at row i, column j is 0 then there is no edge from node i to node j. If the value is non-zero, then there is an edge from node i to node j and the value is the weight of that edge. If the graph is unweighted, then the value is simply 1. Below is an adjacency matrix representing the graph in Figure 2.3. In this matrix, the first row and first column represents node A, the second row and column represents node B, and so on.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 4 & 0 \end{bmatrix}$$

2.3.3 Minimum Feedback Arc Set Problem

A **feedback arc set** is a set of edges that can be removed from a graph to turn it into a directed acyclic graph. The **minimum feedback arc set** is a feedback arc set with the least possible number of edges [21]. There can be multiple minimum feedback arc sets for each graph. Consider the graph G in Figure 2.4 below. G has a cycle with nodes A, B, C, and D. Removing the edge from A to C and from B to C results in graph G' (Figure 2.5) which is a directed acyclic graph. Therefore, edges AC and BC are a feedback arc set. However, removing only edge AD also results in a directed acyclic graph, which is shown as G'' in Figure 2.6. Since this feedback arc set only contains one edge, it is a minimum feedback arc set.

Finding a minimum feedback arc set (mfas) for any graph is an **NP-hard problem** (see [22], p. 192). This will be discussed further in the following sections, but, in short, a problem being NP-hard means that there is no known algorithm that can solve this problem in polynomial time [23]. A naïve solution to find an mfas is to try removing every combination of edges and for each combination, check if the remaining edges form an acyclic graph. If so, check the number of edges removed and see if it is the least. If n is the number of nodes in the graph, this costs $O(2^{n^2})$ because there can be up to $O(n^2)$ edges in a graph and for every edge there are two choices: it can be kept or removed. Additionally, for each graph, we also must ensure that it is acyclic, which would mean a

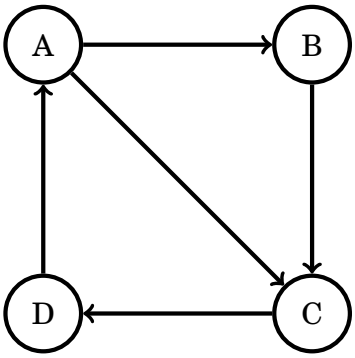

Figure 2.4: *Graph G, a Cyclic Digraph*

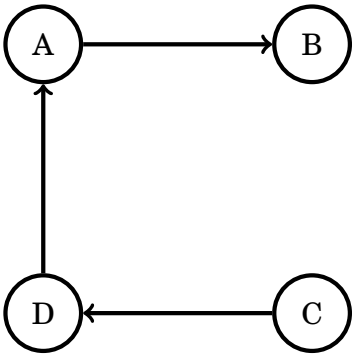

Figure 2.5: *Graph G', an Acyclic Digraph*

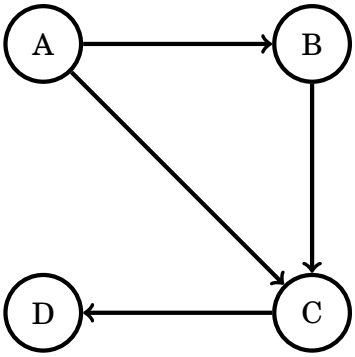

Figure 2.6: *Graph G'', an Acyclic Digraph*

feedback arc set was indeed removed. Using Depth-First-Search, this costs $O(n^2)$. The total time complexity of this solution is $O(2^{n^2} \times n^2)$, which is indeed exponential.

A weighted version of the minimum feedback arc set problem would be to find the feedback arc set that has the smallest weight instead of the smallest number of edges. The unweighted version is a special case of the weighted version where every edge has the same weight. To find the weighted minimum feedback arc set, simply add the weights of every possible feedback arc set and choose the set with the minimum weight.

Because removing a feedback arc set results in an acyclic digraph, the nodes of the graph can be ordered. The order must start from a node with no incoming edges. Looking at graph G'' above, it is clear that the nodes should be ordered ABCD by following the directed edges. An ordered image of G'' is shown below in Figure 2.7. Note that every edge points to the right.

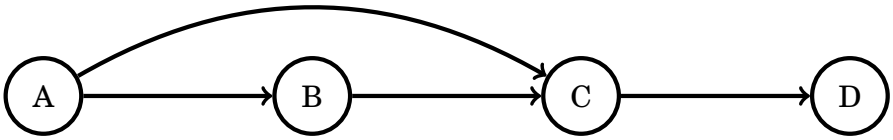

Figure 2.7: *Ordered Graph G''*

If the edges in the mfas were to be added back in, they would all be pointing left. These edges are therefore known as **backedges**. They point against the order, or backwards. Figure 2.8 shows the backedge removed from graph G in red. Because the edges in a feedback arc set are all backedges, finding the mfas is equivalent to finding the order of nodes with the least backedge weight. Therefore, another way to find the mfas is for every ordering of the nodes in the graph, find the weight of the backedges. The mfas is the backedges of the order whose backedge weight is the least [21]. This solution is $O(n^{n+2})$ since there are at most n^n ways to order n nodes, it takes n^2 operations to make an adjacency matrix, and $\frac{n^2}{2}$ operations to count the backedges in the adjacency matrix.

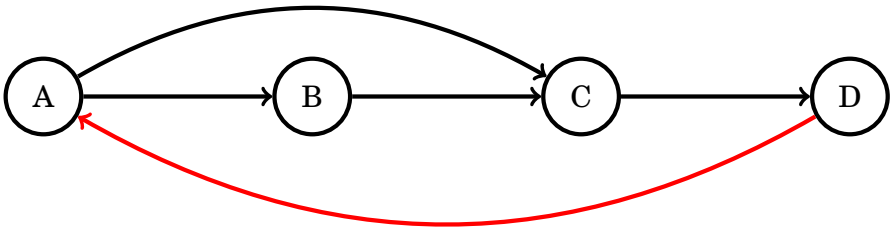

Figure 2.8: *Ordered Graph G''*

2.4 P and NP

A common way problems are classified is by their known solutions. There are two major complexity classes, P and NP. The set of P contains problems that are known to be efficiently solvable in polynomial time. More precisely, they can be solved in time complexity $O(n^c)$, where n is the input size and c is a constant. The set of NP contains

problems for which if a potential solution is presented, it can be verified in polynomial time. NP stands for non-deterministic polynomial. P is a subset of NP, but it is unknown if the two sets are actually equivalent [24]. However, most people believe that $P \neq NP$. The P vs. NP problem is one of seven Millennium Problems, which were proposed in 2000 as some of the most important problems in mathematics. There is a one million dollar prize for solving this problem and proving that P and NP are either equivalent or separate [25]. Therefore, this problem holds significant importance in the world of computer science.

There are two further classes of problems within NP: NP-Complete and NP-Hard. NP-Hard is the set of problems that can be reduced to each other in polynomial time. NP-Complete is the set of problems that are both NP and NP-Hard. Figure 2.9 shows the relationships between the problem classes, both in the case that $P \neq NP$ and in the case that $P = NP$.

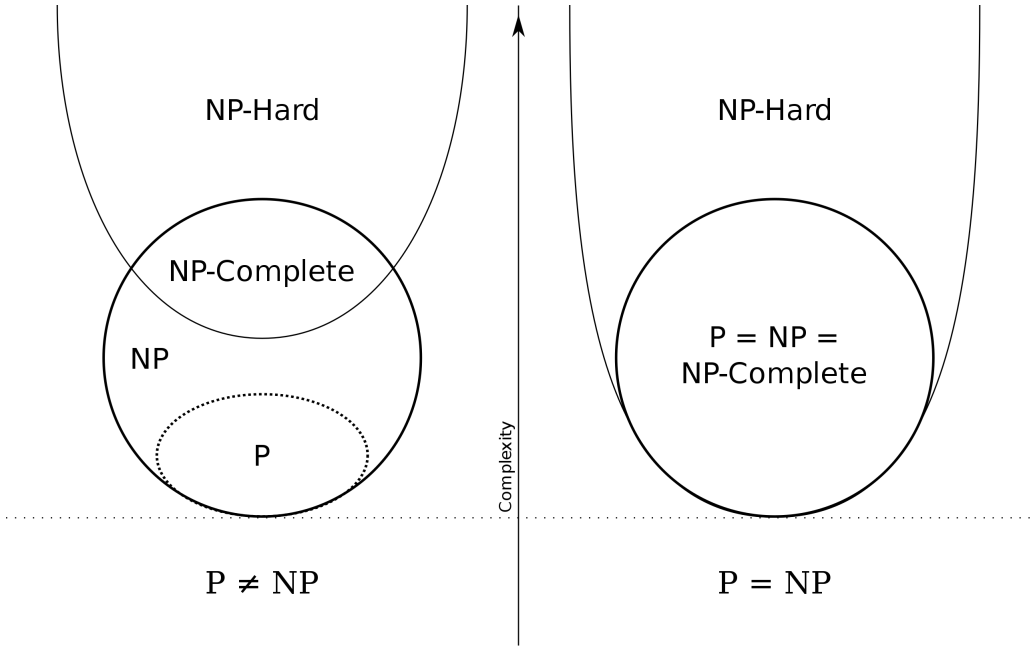

Figure 2.9: *Relationships between Problem Classifications [26]*

2.4.1 Problem Solving Techniques

As an NP-Hard problem grows, there are no known methods to solve it in a reasonable amount of time. In fact, if $P \neq NP$ then there are no methods to solve it in a reasonable amount of time. However there are many methods one can use to solve small NP-Hard problems in a manageable amount of time. These methods use several approaches to

making NP-Hard problems manageable, such as decreasing the time complexity of the problem, or separating the problem into smaller subproblems.

2.4.1.1 Brute Force

A **brute force** approach is a method of solving a problem by trying every single possible solution until a suitable one is found [20]. This approach has the benefits of being simple to understand, as well as being guaranteed to find a solution, if one exists for the given problem. However, brute force solutions are often very computationally expensive, especially for NP-hard problems [20].

2.4.1.2 Dynamic Programming

Dynamic programming is an approach which recursively breaks down a large problem into a series of subproblems [24]. Using the solutions of these subproblems, the larger solution can be found. This approach significantly cuts down on the time requirement to find the optimal solution, since not all possible solutions have to be considered. However, dynamic programming has higher memory consumption from needing to store the solutions to every subproblem, which can become an issue if the number of subproblems is very large [24]. Additionally, not every problem can be divided into the subproblems necessary to utilize a dynamic programming approach.

2.4.1.3 Multithreading

Multithreading is a way of writing code so as to utilize multiple processes at the same time to solve a problem [20]. This concept works especially well with dynamic programming problems, since many subproblems can be solved simultaneously.

Multithreaded algorithms must avoid a **race condition**, or situation in which a system attempts to simultaneously perform more than one operation that must be completed in a specific sequence. To prevent this, multithreaded algorithms utilize **mutual exclusion**, which is essentially a programmatic lock that only allows one process to have access to a shared resource. They also use **atomic operations**, which are operations that other threads understand to be instantaneous and cannot be interrupted [20].

2.5 Previous Work

In the 2017-2018 school year, an MQP was completed that focused on using algorithms to generate sports rankings. This MQP utilized several algorithms to generate rankings, including brute force solutions to the minimum feedback arc set problems, as well as a few approximation algorithms. This MQP included weight scales for its graphs which consider recency (represented as α) and point differential (represented as β). These contributions, as well as the overall structure of the MQP, provided significant background for this project [27].

2.6 Summary

This chapter introduced several key topics relating to our topic. Different sports leagues were discussed, and their variations are important as we wish to test our project against a wide range of data. Graph theory is used extensively throughout this project as a way to model the data for each season. The complexity of problems and ways to solve problems were explained. This is especially important as the problem we solved has no known polynomial time efficient solution.

CHAPTER 3

DESIGN

3.1 Ranking System

Our proposed ranking system is designed to take game results into consideration as much as possible. Therefore, our goal is to create rankings that minimize the amount of occurrences of teams being ranked lower than teams they beat.

By representing teams and the games between them as a digraph, we can consider our goal as minimizing the number of backedges in a ranking. Further, by applying edge weights to represent point differential and game recency, we can minimize the weight of the backedges instead of the number of backedges, which is equivalent to finding the minimum feedback arc set (mfas).

3.1.1 Graph Structure

To create a digraph of a season, we treat the teams as nodes and the games between the teams as directed edges between the nodes. If Team A beats Team B, an edge from Node A to Node B is created. If Team A beats Team B multiple times, we add to the edge's weight instead of creating multiple edges. If Team A and Team B tie, no edge is added.

3.1.2 Edge Weights

Since some game results can have more significance than others, edge weights are given to each game. The two factors in consideration are game recency and point differential. To control the relative importance of each factor we introduced α and β parameters. These parameters control the minimum value of each factor. α creates a linear scale for game recency, from the value of α to 1. Similarly, β creates a linear scale for point differential, from the value of β to 1. If α and β are set to a value of 1, all games will be weighted equally.

3.1.2.1 Game Recency

Game recency is calculated on a linear scale between α and 1. Since each sport has a different scheduling system and number of games, the recency value for different sports is handled slightly differently. For football (NFL and NCAA Football), we treat every game in a distinct week as occurring at the same time. Since each team only plays once per week, there is no difference between playing Sunday afternoon or Monday night. For every other sport, we treat every game that was played on the same day as occurring at the same time. Even though baseball teams can play two games on the same day, this is rare and the time difference between the games is negligible.

The first game of each season is designated as the lowest value game and given a value of α. The last game of each season is designated as the highest value game and given a value of 1. Every game in between is placed linearly between alpha and 1 based on when the game was played. For convenience, even if no games are played on a particular day during the season, we still count that day as being part of the season, there are just no games with a recency value corresponding to that day.

3.1.2.2 Point Differential

Point differential is calculated on a scale between β and 1. The lowest point differential in the season, or the closest game, is given a value of β. The top 25% of the point differentials in the season are given a value of 1. All of the point differentials up to the 75^{th} percentile are placed linearly between β and 1. We chose to give the largest 25% of the point differentials the same value because we do not want blowouts to skew the significance of the wins. For example, in football, winning by 30 points or 35 points is not very different.

To decide where to cap the point differential, we analyzed data on the frequencies of different point differentials in different sports. In every case, the 75^{th} percentile represented a point differential in which one would consider the winning team won by a large margin. Table 3.1 shows the 75^{th} percentile of point differentials for different sports and seasons. Figure 3.1 shows a chart of point differential distributions for different sports and seasons.

Table 3.1: *75^{th} Percentile Point Differential for Different Sports*

Sport and Year	75^{th} Percentile Point Differential
NFL 2016	15
NFL 2017	17
NHL 2017	3
MLB 2018	5
NCAA Basketball 2017	18

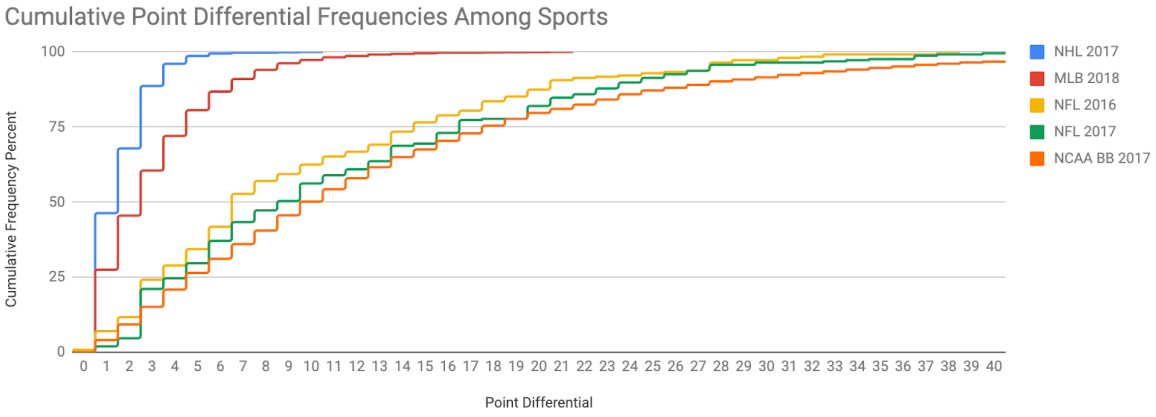

Figure 3.1: *Chart of Point Differential Distributions*

3.2 Data Format

In order for our program to know the results of games between teams, we need to provide season data. We chose to use csv files to store this data and input into our program. Two files are needed by the program: a csv file that contains information about each game, and a csv file that contains the names of all of the teams that we want to rank.

In the csv file containing game information, each row is a game. The only required data are the team names and the score for each team. However, more information can be provided such as the date of the game and which team was the home team. All of our csv files have standardized column headers, so the program knows which information is in each column and the order of the columns does not matter.

The csv of teams is a simple csv file where each row is a different team name. This file tells the program which teams to extract game data for, and which teams to ignore.

3.2.1 Ranking Considerations Among Sports

We chose to compute rankings for the following sports leagues: MLB, NFL, NBA, NHL, NCAA Basketball, and NCAA Football. These sports leagues vary significantly in number of teams, number of games, and average points per game. We chose the first four leagues because they are the most popular professional sports in the United States, and therefore many people will be interested in our results for those sports. We also wanted to include NCAA Basketball and Football because they present interesting ranking challenges due to the large number of teams. Additionally, rankings for these leagues are very important because championship tournament members in NCAA Football and Basketball are determined by a ranking.

The MLB, NFL, NBA, and NHL all provide very dense graphs because of how many teams play each other within the leagues. In the NBA and NHL, all nodes are connected since every team plays every other team at least twice. Additionally, the MLB has only 30 teams, each of which plays 162 games, meaning that two teams may play each other many times.

NCAA Basketball and Football provide very different graphs from those of professional leagues. NCAA Basketball and Football graphs have many more nodes and are sparser than their professional league counterparts. These graphs often have smaller subgraphs within them that are almost completely separate from the rest of the graph.

Overall, the differences between each sport analyzed in this project significantly affect the ranking program design. It must account for both sparse and dense graphs, as well as varying numbers of nodes. It is important to consider these factors when writing algorithms to rank graphs.

3.3 A Dynamic Programming Algorithm

In the context of this project, our problem involves determining which order of nodes gives the smallest possible feedback weight, or backedge weight. This order of nodes represents the best possible ranking in terms of backedge weight; we call this order an **optimal** order. In this section we will present a dynamic programming algorithm that will find an optimal order with a significantly reduced time complexity compared to the brute-force algorithm. The dynamic programming algorithm utilizes the following definition.

Definition 3.3.1. *For a sequential ordering R of the nodes of a directed graph G, a* ***con-***

secutive subgraph *G' of G is any (non-empty) subgraph composed of nodes consecutively ordered by R and the arcs connecting these in G.*

In addition to this definition, the dynamic programming algorithm relies on the following result proved by Younger (Theorem 5.a on page 241 in [21]).

Theorem 3.3.1. *([21]) Assume that we are given an optimal ordering R of a directed graph G and a consecutive subgraph G' of G. Then G' must have as minimum feedback arc set those arcs of G' that are feedback arcs according to R.*

In other words, a consecutive suborder of an optimal order will be also optimal. We can use this idea dynamically to build larger optimal orders from smaller optimal orders.

3.3.1 Dynamic Programming Design

Since each **subproblem** provides optimal accuracy, we can design a dynamic programming algorithm based on the subproblems. For each subproblem, we must find the minimum feedback arc set for a smaller set of nodes. This dynamic programming algorithm is bottom-up, which keeps only an optimal solution for each set of nodes. The algorithm requires the full graph to be broken up into smaller subgraphs. For each subgraph, we can consider separating one node, which will connect to the remaining subgraph. When separating nodes, we must consider every possible split. Then, for each split, we must calculate the added feedback weight from the separated node to the remaining subgraph.

Let $opt(G')$ be the optimal solution for subgraph $G' = (V', E')$ where $V' = \{v'_0, v'_1, \ldots, v'_{k-1}\}$ and E' is the set of the edges in E within V'. Then, using the above splitting idea, we can define a dynamic programming equation as:

$$opt(G') = \begin{cases} 0 & \text{if } G' = \varnothing \\ \min\limits_{v \in G'} (opt(G' - \{v\}) + \sum\limits_{u \in G' - \{v\}} w(v,u)) & \text{if } G' \neq \varnothing \end{cases} \tag{3.1}$$

Additionally, we need to keep track of the order of each subgraph so that we can determine a final ordering of all nodes. We could simply store the suborder for each subgraph, and connect those together, but that is not very space efficient. In order to minimize space complexity, we store only the node that gets separated. Therefore, we can use backtracking from that point to get the full order.

3.3.2 Pseudocode

Algorithm 1: *Dynamic Programming Brute Force*

Data: Graph $G = (V, E)$ and adjacency matrix for weighted edges w
Result: Minimum feedback set of the graph

```
1 begin
2     forall G' is subgraph of G do opt(G') ← ∞;
3     opt(∅) ← 0;
4     // need to be increasing orders
5     forall G' is subgraph of G do
6         G' ← min_{v∈G'}(opt(G' − {v}) + Σ_{u∈G'−{v}} w(v,u));
7     end
8     return opt(G);
9 end
```

3.3.3 Analysis

Our *Dynamic Programming Brute Force* solution is significantly more time efficient than a standard brute force solution to the minimum feedback arc set problem. In our algorithm, we evaluate every subgraph. Considering n as the number of nodes, there are 2^n subgraphs. For each of these subgraphs, we need to consider every possible one-node removal. This requires us to calculate the added feedback weight for a removed node, which takes $O(n)$. Ultimately, this process leaves the algorithm with a time complexity of $O(2^n n^2)$, which is much less than its original brute force complexity of $O(n^{n+2})$. We replace the n^n factor in the original brute force solution with 2^n from the dynamic programming solution, which results in a very large time difference between the two algorithms.

3.3.4 Parallelization

In order to further increase the efficiency of this algorithm, we chose to parallelize it. This process is possible because one subgraph's state can be calculated independently from other subgraph's states, as long as the subgraphs contain the same number of nodes. Because a state can be calculated independently from states whose graphs consists of the same number of nodes as the current node's, we can split the computation among threads.

We define all subgraphs with an equal number of nodes to be on the same **level**. Each level of subgraphs are calculated based upon subgraphs in a previous level. Starting from an empty graph, we can calculate each concurrent level by utilizing data from only the current and previous levels. This ensures that there will not be a race condition. Ultimately, the algorithm is parallelized by giving one state at a time to a thread, parallelizing only within each level.

3.4 Approximation Algorithms

3.4.1 Sliding Window

One approximation algorithm we designed is called the *Sliding Window* approximation algorithm. This algorithm uses our *Dynamic Programming Brute Force* algorithm on separate **windows** or ranges of teams. Given the data of 32 teams and a window size of 20, this algorithm would run dynamic brute force on teams 13 through 32. It would then save the team that was ranked last (32^{nd} place) in the full ranking. Next, the algorithm would run dynamic brute force on teams 12 through 31. The team that was ranked last in this window would then be ranked in 31st place in the full ranking. This process continues until there are only 20 teams remaining, at which point the *Sliding Window* algorithm computes a ranking for those teams and adds the entire result to the full ranking. When the window size is fixed at size w, and there are n teams, this process has a time complexity of $O(2^w * w^2 * (n-w))$. This is significantly less than that of the full brute force algorithm when w is smaller than n, allowing for a much faster solution. If w is constant, then this solution is actually linear, which is very efficient.

The *Sliding Window* algorithm uses windows that start from the bottom of the ranking and move to the top of the ranking. We chose to use this bottom to top process in order to have the greatest possible certainty about the top ranked teams. In the example mentioned above, the *Sliding Window* would begin its first window with 20 teams, leaving 12 unranked and therefore uncertain teams outside of the ranking window. As it moves to the next window, it has a window of 20 teams, plus one ranked or certain team as well as only 11 unranked or uncertain teams. The uncertainty continues to decrease until the algorithm reaches the final window 20 teams at the top of the ranking, at which point there are zero remaining uncertain teams. By using this bottom to top method, we were able to have the greatest certainty about the top ranked teams, which are usually the most important teams in the minds of sports fans, since they would be

the teams ultimately competing for the championship.

3.4.1.1 Preordering

The process of using windows is only able to create a reasonably accurate ranking when it begins with a good preorder. This is because the windows do not include the entirety of the data, and as a result, need to contain the most relevant information within them. For example, when the *Sliding Window* algorithm ranks the bottom 20 input teams, it intends to find the team that should be ranked last in the overall ranking. If the team that should be ranked last is not present in the preordered bottom 20 teams, the final ranking created by the *Sliding Window* algorithm will not be optimal, no matter where the remaining teams are ranked. In order to avoid negatively affecting the performance of the *Sliding Window* algorithm, it is important to create a strong preorder for this algorithm

The strength of the preorder can severely affect the performance of the *Sliding Window* algorithm. Therefore, we decided to preprocess the input data to generate a preorder of teams based on total net edge weight out of nodes. This preorder can be generated quickly by utilizing the game and team data that already exists in the adjacency matrix. The resulting preorder gives a reasonable ordering for teams based on weighted wins and losses that allows the *Sliding Window* algorithm to, in turn, create a reasonable ranking. The algorithm will also accept a manually created preorder, which allows for further experimentation with ordering when desired.

3.4.1.2 Pseudocode

Algorithm 2: Sliding Window

Data: adjacency matrix, numTeams, windowSize, preorder
Result: ranking

```
1 windowPos = numTeams - 1;
2 while windowPos != (windowSize - 1) do
3 |   run brute force on current window of preorder;
4 |   get bottom team from brute force window;
5 |   add bottom team to top of existing final ranking;
6 |   windowPos - -;
7 end
8 run brute force on final window of preorder;
9 add full window to top of existing final ranking;
```

3.4.2 Brute Force Pruning

This approach is similar to the brute force approach to creating a ranking. The important difference is that instead of keeping all possible subrankings at each level, we keep only those rankings which have low backedge weights.

In each level, we keep only a certain number of subrankings, based upon a limiting number. If there are equivalent rankings that would cause us to keep more rankings than the limiting number, then all of those equivalent rankings are discarded to keep the number of rankings below the limiting number. For example, if the limiting number of rankings was 100, and rankings 90 to 110 had the same backedge weight, then the algorithm keeps the 89 subrankings which have the lowest backedge weights in each level.

If many subrankings have the same total backedge weight, it can take much longer to execute this pruning method. Instead of keeping all subrankings with equivalent backedge weights, we keep only a number of subrankings that are below the limiting number. We randomize which subrankings are kept in order to make sure all subrankings with equivalent backedge weights have a similar chance of being chosen.

Although initial orders can have a significant effect on the final results of this algorithm, we use strategies to help decrease the impact of these initial orders. These strategies include randomizing chosen subrankings, permuting ranking orders, and taking the minimum weighted ranking generated by several runs of the algorithm.

3.4.3 Dynamic Brute Force Pruning

This algorithm is an approximation version of the previously discussed Dynamic Programming Algorithm, adapted to take less time to run. In order to decrease run-time, this pruning algorithm removes some subrankings from consideration during computation. Subrankings are removed when they have a high total feedback weight. This approach keeps subrankings according to the limiting number method presented in Section 3.4.2. However, this approach constructs a ranking using the *Dynamic Programming Brute Force* algorithm instead of the traditional brute force algorithm.

3.5 Postprocesses

Rankings returned by our algorithms are just one of potentially many with an equivalent backedge weight. These equivalent rankings pose an additional problem

within this project: finding the best possible ranking amongst those with equivalent backedge weights. Our postprocesses attempt to address this issue.

3.5.1 Range of Correctness

One postprocess that we included in the project is called *Range of Correctness (RoC)*. This postprocess is intended to indicate the certainty of each team's placing within a ranking. The *Range of Correctness* in its most basic form is not intended to choose the best ranking given equivalent backedges. Rather, it is intended to allow a viewer to understand the solidity of each team within a given ranking.

The *Range of Correctness* works based on only the edges in the graph. Given a ranking, this postprocess will calculate each team's upper and lower rank limit. A team's upper rank limit is determined by the rank of the nearest team above it in the ranking that it has lost to, which is indicated by an incoming edge from a team ranked above the current team. *RoC* records the upper rank limit for a team as one position below this team. A team's lower rank limit is determined by the rank of the nearest team below it in the ranking that it has won against, which is indicated by an outgoing edge to a team ranked below the current team. *RoC* records the lower rank limit for a team as one position above this team. For example, if a team lost to the team in first place and beat the team in fifth place, that team's upper rank limit would be two and its lower rank limit would be four. This gives an indication that despite where that team is within the ranking, it could be ranked anywhere from number two to number four based on the team's performance without changing the backedge weight of the ranking.

Range of Correctness is an important postprocess because it shows how much movement is possible within a given ranking. A team can be moved to any ranked position within its range of correctness as long as no other ranges of correctness become impossible as a result of the move. This movement is possible because there are no conflicting edges between teams that can move within the range of correctness, and therefore movement within a team's range of correctness does not affect the total backedge weight of the ranking.

When *Range of Correctness* is run on a ranking, it is designed to print out a full ranking with each team's range of correctness next to its spot in the ranking. For example, if the New England Patriots were ranked number one with an upper limit of one and a lower limit of three, it would look like the following:

1. New England Patriots [1, 3]

This format is intended to clearly indicate to someone looking at the ranking how much movement for each team is possible within the ranking without changing the total backedge weight.

3.5.2 Tree of Correctness (ToC) Method

Often, when a ranking is created, there is flexibility to move teams without changing the backedge weight. A common situation where this could occur is when two teams are next to each other in a ranking, but they have not played each other. Since there is no edge between the two nodes in the graph, no backedge will be introduced by switching the team's places in the ranking. The *Tree of Correctness (ToC)* method was designed to create the "best" ranking that has an equivalent (or better if possible) backedge weight to a given ranking and adjacency matrix.

The first step in the ToC method is to remove all of the backedges from the graph of games. These backedges are known from the input ranking provided. By removing all of the backedges, all of the cycles in the graph are broken, so we get an acyclic graph. An acyclic graph is also known as a forest, or a tree if it is connected. We call it *ToC Structure*.

Once the tree is created, a root node (a node with no incoming edges) is chosen to be the first team in our new ranking. Since there are no incoming edges, choosing this team as the best is safe since no new backedges can be introduced. Once a node is chosen, edges connecting to it are removed from the tree. This process is repeated for each node with no incoming edges, each time adding the team chosen to the next spot in the ranking. Once all nodes have been selected, the new ranking is created. There will always be at least one team with no incoming edges to choose because there will always be at least one root of the tree.

It is important to note that the ToC method cannot generate all possible rankings with an equivalent total backedge weight because of how backedges are removed in cycles. For example, consider a graph of three nodes that contains one cycle (of equivalent weight) as shown in Figure 3.2. Each edge can be considered a backedge depending on the ordering of the nodes. By removing one edge we remove the ability to remove the other edges since they are no longer backedges. Therefore, only one ordering of these three nodes can be considered by the ToC method. We cannot generate all three of the trees shown in Figure 3.3, Figure 3.4, and Figure 3.5 in the same run of ToC.

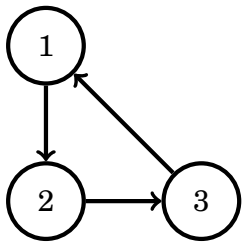

Figure 3.2: *Original Graph*

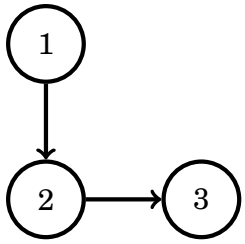

Figure 3.3: *1st Equivalent Graph with Order {1, 2, 3}*

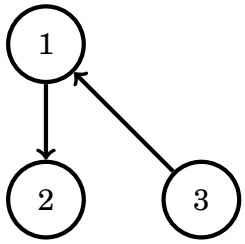

Figure 3.4: *2nd Equivalent Graph with Order {3, 1, 2}*

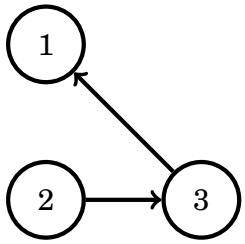

Figure 3.5: *3rd Equivalent Graph with Order {2, 3, 1}*

3.5.2.1 Proof

Lemma 1: *The total feedback weight of a ranking obtained from ToC is less than or equal to the weight of a provided ranking.*

Proof. Edges in a graph can be separated to two parts, forward edges and backward edges. An acyclic graph from the ToC structure consists of all of the forward edges given by a provided ranking. To ensure that no forward edges are turned into backward edges,

we only remove nodes that do not currently have incoming edges. If we removed nodes that have incoming edges, those edges would become backward edges, which would increase the total backedge weight of the ranking. By repeating the ToC process for every node in the acyclic graph, no backedges are ever added, so the backedge weight cannot increase. ■

3.5.2.2 Pseudocode

Algorithm 3: Tree of Correctness (ToC) Structure

Data: Graph $G = (V, E)$ and ranking for each team $rank$
Result: ToC structure (acyclic graph) corresponding to G as ToC_G

```
1 begin
2     E' ← ∅;
3     forall e = (u, v) ∈ E do
4         // add an edge if node u is ranked higher than node v
5         if rank(u) < rank(v) then E' ← E' ⋃ {e};
6     end
7     return ToC_G = (V, E');
8 end
```

3.5.3 Secondary Metric

Within the constraints of the ToC algorithm, we utilize a secondary metric (or score) in order to select the best possible ranking. The idea behind this secondary metric is to give a higher score to teams who defeat teams that are higher in the ranking, and a lower score to teams who lose to teams that are lower in the ranking. Each team gets a score according to the following equation:

$$score(v_i) = \sum_{j=0}^{n-1} (n + 1 - rank(v_j)) \times w(v_i, v_j) - \sum_{j=0}^{n-1} rank(v_j) \times w(v_j, v_i) \tag{3.2}$$

where $rank(v_i)$ is the rank (starting from 1) of node v_i, and n is the number of teams. When this score is maximized, we have the best possible ranking according to our second metric.

3.5.4 Range of Correctness Reorder

The idea behind the *Range of Correctness Reorder* postprocess is to rearrange a ranking depending on the second metric, while limiting a team's mobility to the range of correctness. *Range of Correctness Reorder* runs the *Tree of Correctness* postprocess with the second metric to create a new ranking. Instead of maximizing the second metric score of the entire ranking, we choose the team with the best second metric score from every team whose *Range of Correctness* allows it to be in a certain spot in the ranking. This process repeatedly generates new rankings until an identical ranking is generated twice. When that happens the ranking that occurred twice is kept.

3.6 Comparing Rankings

Comparing our rankings to externally generated rankings, such as those from ESPN or the NCAA, allows us to determine whether our rankings make sense. While our rankings produce the lowest possible backedge weight, some teams may be ranked significantly differently than they would be in a traditional ranking system. Comparing our own system to these rankings allows us to fine-tune different input values to best emulate these external rankings.

Our ranking comparison algorithm takes each team in the external ranking, finds its rank in our ranking, and takes the absolute value of the difference. By taking the summation of the absolute value of this difference for each team, we arrive at a total difference score that shows us how close our ranking is to the external one. This is shown in the following equation, where R_i is team i's rank in our ranking, and E_i is team i's rank in the external ranking:

$$\text{Rank Difference} = \sum_{i=1}^{n} |R_i - E_i| \tag{3.3}$$

By calculating this difference for a series of different α and β values, we can determine which of these values produces the closest ranking to the external one, allowing us to gauge the importance of game recency and point differential in these external rankings.

3.7 Summary

This chapter focused on the overall design of our algorithms and postprocesses. It outlined the structure of the adjacency matrices that store game data. The design of

our optimal *Dynamic Programming Brute Force* algorithm, as well as our approximate *Sliding Window*, *Brute Force Pruning*, and *Dynamic Brute Force Pruning* algorithms were discussed in detail and accompanied by pseudocode. The concept of *Range of Correctness* was introduced and its importance in displaying the flexibility of a ranking was explained. Finally, our postprocessing algorithms *Range of Correctness*, *Tree of Correctness*, and *RoC Reorder* were introduced and explained.

CHAPTER 4

IMPLEMENTATION

4.1 Program Configuration

There were many options when determining how to best implement the algorithms described in the design section above. Firstly, we had to choose a programming language that would allow fast execution time and manageable threading. We also had to consider general program design for flexibility of testing and result gathering.

We chose to use the C++ language to implement our program. It is an efficient language that also allows for the use of objects and other convenient data structures. The boost library has many additional features that make programming complex tasks easier [28].

To implement our program, we decided to have one program that can execute multiple different algorithms on different data. For ease of running the program, the only input argument is a configuration file that specifies which ranking algorithm to run, what data to run it on, what the values of α and β should be, which postprocess algorithm to use (if any), and other convenience measures, such as where to save the results.

To run our program, we used WPI's ACE computing cluster. This allowed us to use up to 40 threads to run the levelled version of our *Dynamic Programming Brute Force* algorithm. Additionally, it provided a common platform where we could all obtain results that were consistent with each other. The majority of the nodes in the ACE computing cluster use two Intel Xeon Gold 6138 Processors. These processors run at 2.00 GHz and can be overclocked to 3.70 GHz [29].

4.2 Data Acquisition

To obtain data, we wrote python scripts to scrape game data from different websites. These scripts read the data for a specified season and put it into the csv format described above. The following table shows where data for each sport was obtained from.

Table 4.1: *Sources for Data Sets*

Data Set	Source
MLB 2018	`www.baseball-reference.com`
NFL 2018-2019	`www.nfl.com`
NBA 2017-2018	`www.basketball-reference.com`
NHL 2017-2018	`https://www.hockey-reference.com`
NCAA Basketball 2017-2018	`www.sports-reference.com/cbb/boxscores/`
NCAA Football 2018-2019	`www.sports-reference.com/cfb/boxscores/`

4.3 Graph Generation

The first step of generating an adjacency matrix from the input data is to create a lookup table that maps team names to integers. These integers are used as indices in the adjacency matrix. To create the lookup table, the program reads through the input csv file of team names. The first team is mapped to 0, the second team to 1, and so on and so forth until all of the teams have been mapped.

Next, the program reads the csv of game data. For each line in the file, the program first determines the winning team and the losing team depending on the score (ties are ignored). It then gets the indices of both teams from the lookup table. If either of the two teams are not in the lookup table, then the program knows that at least one of the teams is not being considered for the ranking. In that case, the game is not added to the adjacency matrix and the next game is looked at. If both teams are in the lookup table, then the edge weight is calculated and stored in the adjacency matrix.

If one team is found in the lookup table but the other team is not, there is a user option to enable a “supernode." This supernode acts as an extra team that represents all of the teams we are not ranking. If a team in the lookup table beats a team not in the lookup table, then it counts as a win over the supernode. Conversely, if a team in the lookup table loses to a team not in the lookup table, then it counts as a loss against the supernode.

To calculate the edge weights, the scale for each factor must be determined. Therefore, the program scans the game data to find when the first and last games occurred, as well as the distribution of the point differentials. When the recency edge weight is being calculated for a specific game, the program knows when in the season the game took place and can assign the appropriate value. Similarly, when the point differential edge weight is being calculated for a specific game, the program knows at what percentile that point differential is, and the appropriate value is assigned.

4.4 Algorithms

4.4.1 Dynamic Programming Algorithm

4.4.1.1 Initial Implementation

Due to memory and time constraints, this algorithm can consider a maximum of approximately 32 nodes. To save on memory and time, we chose to represent the nodes in a subgraph as a bitmask stored as an unsigned long.

Let n be the number of nodes in a graph. The indices of the nodes go from 0 to $n-1$. We define the existence of the i^{th} node as the value of 2^i. When a node is included in a subgraph, it is represented by a 1 in the bitmask. To add a node to the subgraph, the program uses bitwise OR (|) between the existing graph and a bitmask that only has a 1 at the node index we want to add. For example, to add node 2 to a subgraph of nodes 1 and 4 the following operation is performed.

$$1001 \mid 0100 = 1101 \tag{4.1}$$

Note that the algorithm does not need to keep track of the edges in the subgraph because the adjacency matrix still has that information. It is only keeping track of which nodes are included in the subgraph. The maximum value of $2^n - 1$ is needed to represent the entire considered graph. Considering the maximum of 32 nodes, an unsigned long is sufficient to represent all subgraphs.

More precisely, let graph $G' = (V' = \{v_{g_0}, v_{g_1}, \ldots, v_{g_{k-1}}\}, E')$ where $g_0 < g_1 < \cdots < g_{k-1}$. We define the value representing graph G' as:

$$state(G') = \sum_{i=0}^{k-1} 2^{g_i}$$

Then, we represent the optimal ordering, or opt as an array. Because the bigger state values only need smaller state values, the program can directly iterate from 0 to $2^N - 1$ with Equation 3.1. Initially, it sets all values in opt to infinity and changes them when processing. It calculates each of opt in a bottom-up manner; that is, it changes value-greater states from the current state. To calculate the added feedback weight, the program reverses a state value to nodes by using bitwise AND (&). Precisely, to decide if subgraph G' consists of node i, it checks if $state(G')\&2^i > 0$. Then, it uses that information to calculate all weights from one of the excluded nodes to all included nodes and change the value of a new state to lower value.

In addition, the algorithm stores order data in an array. It is then able to backtrack through the array and obtain the best order of nodes. It changes the node number when it changes the state value as explained in the previous paragraph. To retrieve the complete order, it gets the node number, changes it to the node value, and reduces the kept state value until it becomes zero (empty state). Since the algorithm stores the node number during processing, the order is entered in reverse. The last step is to reverse that order and get the correct one.

4.4.1.2 Parallelizable Implementation

It is difficult to separate state values which should be calculated earlier from Section 4.4.1.1. Instead of parallelizing the previous implementation, we completely changed it to better allow for parallelization. For each step in the parallelized implementation, we want to keep and use only the states that are necessary for the calculation of the next states. The algorithm initializes two C++ STL vectors to keep current and previous states; it swaps them when it finishes each level. It allocates space for current states beforehand because it can calculate exactly how much space it needs. Then, we parallelize the process of finding current states from previous states by using a global atomic integer to track a position to be filled. Each thread calculates one previous state at a time to create states which are guaranteed to be different from others.

Similarly to the initial implementation, the algorithm stores the optimal solution for each subgraph and the last node of subgraphs globally. In contrast to the initial implementation, the algorithm starts from a current state by removing one node from a set of nodes of a subgraph, calculates backedge weight from the removed node to remaining subgraph, adds to stored minimum values for subgraph, and takes the minimum value. It, similarly, parallelizes on current states we obtain earlier (explained in previous paragraph).

In order to avoid spending too much time waiting for threads to complete, we decided to use an atomic integer to retrieve the state value from the previous level. This integer will in turn be written to a specific array position in the current level, avoiding the additional time taken by using locking while waiting for the mutex to be released by each thread.

4.5 Approximation Algorithms

4.5.1 Sliding Window

The *Sliding Window* algorithm was implemented according to the logic explained in the *Sliding Window* design section. The algorithm loops through windows of a defined size within a full preorder of teams, running the dynamic brute force algorithm on the ranking, starting from the bottom. Like other algorithms, the *Sliding Window* algorithm allows for α and β values to create different edge weights between games, and therefore is able to consider recency and point differential within each window that it ranks.

4.5.1.1 Preorder

As designed, the *Sliding Window* algorithm requires a preorder. The preorder can either be entered in a file by a user, or it can be generated. The generated preorder takes in an adjacency matrix that has edge weight information in it. The process then creates a preorder based upon the edge weight information.

4.5.2 Brute Force Pruning

As explained in Section 3.4.2, we start the algorithm by shuffling an order of nodes by using C++ STL (std::shuffle). We then duplicate the adjacency matrix and change it so that the first node in the order is index 0, the second node is index 1, and so on and so forth.

For each shuffle, we start by creating a C++ STL priority queue as a maximum heap to keep a pair of total feedback weight and its corresponding subranking. The priority queue will have rankings with a higher feedback weight at the top, so they will be removed before rankings with a lower feedback weight. In addition, we also keep track of the minimum total feedback weight. We start to pop elements that have a higher total feedback weight than the ranking at the limiting number. In our original design concept, our plan was to keep all equivalent feedback weight rankings if that brings us to a total number of rankings less than twice the limiting number. Otherwise, we get rid of all rankings that have an equivalent backedge weight to the ranking at the limiting number. However, in our final implementation, we decided to simply cut off rankings at the limiting number. If there are rankings that cross the limiting number, we randomly choose which ones to keep.

To get a new ranking from each kept subranking, we put a new node to the end of considered subranking, calculate the total feedback weight by adding weights from that new node to all nodes in subranking, and then add it and the newly-created subranking to the heap so that we can reuse that subranking later to save computation. After that, we swap the new node and a node next to it, increase total feedback weight by the weight from the next node to the new node, decrease it by the weight from the other way, and put in heap.

4.5.3 Dynamic Brute Force Pruning

Similarly to *Brute Force Pruning*, *Dynamic Brute Force Pruning* utilizes the same strategy but uses a different weight finding approach. We keep only some states. We pick states one at a time. Then, we choose one of the unranked nodes and try to append it to the end of a state (ranking). Next, we calculate an added feedback weight by summing the weight from the chosen node to the state and store that weight globally as array, which is similar to the optimal *Dynamic Brute Force Algorithm*. Finally, we put it on the heap and repeat the process with different unranked node.

4.6 Postprocesses

4.6.1 Range of Correctness (RoC)

The *Range of Correctness* process utilizes the logic outlined in the design section. This process loops through each team in a ranking and then uses an adjacency matrix to find the closest ranked team above the team in question that was a loss in the team in question's record. Then, it uses the matrix to find the closest ranked team below the team in question that was a win in the team in question's record. After following this process for each team in the ranking, the range of correctness is able to be determined for the full ranking, and is displayed as outlined in the design section.

4.6.2 Tree of Correctness (ToC)

The *Tree of Correctness Method* of postprocessing takes in an adjacency matrix representing games played by all teams. In order to implement the ToC method, we must create an acyclic graph. This acyclic graph can be represented by an adjacency matrix initialized to false boolean values. Once the acyclic graph matrix is created, we

go through game matrix and change the boolean values accordingly. If a position in the game matrix is non-zero and it has an edge from a higher rank to a lower rank, the corresponding boolean will be changed to true in the acyclic graph matrix. After updating the acyclic matrix, we continue calculation with one of three possible processes.

4.6.2.1 Original

The *Original ToC Method* generates equivalent rankings using recursion. Using a function with reference access to a vector, we initialize an empty order globally. Throughout the process, we add and remove elements from this global vector. To add an element, we first check if there is an edge in the acyclic graph going into the element in question. This determines whether or not it can be next in an equivalent ordering. If the element can be next, we keep it and process it recursively until a full equivalent order, or ranking, is complete.

After we obtain every equivalent ranking (or 1 million equivalent rankings are obtained), we use Equation 3.2 to calculate a score for each ranking. For each ranking, we calculate the feedback weight and the score. Then, from rankings with the minimum feedback weight, we pick the ranking with the maximum score.

4.6.2.2 Big Data

For this process, we initialize a global score array to be accessed in future. Then, we iterate through the nodes, using the score calculating function shown below in Equation 4.2 on each node. When the score calculating function is called, it also runs recursively for all of its successors if their scores have not yet been calculated. After retrieving all successors scores, we apply the ToC method. We use a maximum priority queue to determine nodes that can be next in the ranking, taking the one with a maximum score each time. We put any successors which do not have predecessors outside of the ranking.

$$score(v_i) = \gamma|S| + \delta \sum_{u \in S} score(u) \tag{4.2}$$

where S is the set of successors of v_i,
γ is the coefficient of the number of successors,
δ is the coefficient of the total score of successors.

4.6.3 Range of Correctness Reorder

The *Range of Correctness Reorder* process utilizes the logic outlined in the design section. We initialize a set to keep rankings that we have processed. We start each sub-process by generating a second metric value and tree of correctness given graph and ranking. We generate a new ranking using the acyclic graph from the *Tree of Correctness Method*, along with the second metric. We check if the new ranking exists in the set of previously generated rankings. If so, we stop the process and return the ranking; otherwise, we continue the sub-process with the newly generated ranking.

4.7 Summary

This chapter explained how the functionality of the project was implemented. It highlighted the adjustments that were made after the design phase of the project. Code specific details for each algorithm were discussed and reasons for these choices were given. The code implementation of our postprocesses were also discussed for each method.

CHAPTER 5

RESULTS

All code was run using the ACE computing cluster. It is important to note that since this cluster is used by many students and faculty at WPI, we were unable to assure that each run was executed in the exact same circumstances. In some cases, our program was run simultaneously with many other external computations. Therefore, two identical runs could report taking different amounts of time. However, we are still able to identify and analyze trends in the timing of our algorithms.

Additionally, all data was collected at the end of the respective league's regular season. Postseason results are not reflected in any of the rankings - both the ones created by our algorithms and the external rankings.

5.1 Algorithms

The following sections show how our algorithms performed with NFL 2018 data. All rankings were created with $\alpha = 1$ and $\beta = 1$ so all wins are treated equally. Additionally, there is no postprocessing, so each ranking is just one of potentially many with an equivalent backedge weight.

5.1.1 Dynamic Programming Brute Force

The maximum amount of threads we were able to use on the ACE computing cluster was 40. Table 5.1 is a ranking of the 2018 NFL Season computed using *Dynamic Programming Brute Force* with 40 threads.

Using multithreading significantly improved the performance of this algorithm. When using 40 threads, the computation took 1795 seconds (30 minutes) to complete. When using only 1 thread, the computation took 15235 seconds (254 minutes) to complete.

Figure 5.1 is a graph showing how the time performance of our algorithm changes based upon the number of teams (nodes) being ranked. The algorithm was run with 40

Table 5.1: *NFL 2018 Dynamic Programming Brute Force Ranking*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 19.3%

Rank	Team	Range of Correctness
1.	New Orleans Saints	[1, 4]
2.	New England Patriots	[1, 2]
3.	Houston Texans	[3, 8]
4.	Chicago Bears	[3, 4]
5.	Los Angeles Rams	[5, 5]
6.	Kansas City Chiefs	[6, 6]
7.	Baltimore Ravens	[7, 7]
8.	Los Angeles Chargers	[8, 8]
9.	Cleveland Browns	[9, 9]
10.	Denver Broncos	[10, 10]
11.	Pittsburgh Steelers	[11, 12]
12.	Seattle Seahawks	[11, 16]
13.	Cincinnati Bengals	[12, 13]
14.	Indianapolis Colts	[14, 14]
15.	Buffalo Bills	[15, 15]
16.	Tennessee Titans	[16, 16]
17.	Dallas Cowboys	[17, 18]
18.	Minnesota Vikings	[16, 18]
19.	Philadelphia Eagles	[19, 19]
20.	Atlanta Falcons	[20, 20]
21.	Washington Redskins	[21, 21]
22.	Jacksonville Jaguars	[22, 24]
23.	Detroit Lions	[19, 23]
24.	Carolina Panthers	[24, 24]
25.	New York Giants	[25, 25]
26.	Tampa Bay Buccaneers	[26, 30]
27.	Arizona Cardinals	[24, 27]
28.	Green Bay Packers	[28, 28]
29.	Miami Dolphins	[29, 29]
30.	New York Jets	[30, 32]
31.	San Francisco 49ers	[29, 31]
32.	Oakland Raiders	[32, 32]

threads on NFL 2018 data. As expected, the time increases exponentially as more nodes are added.

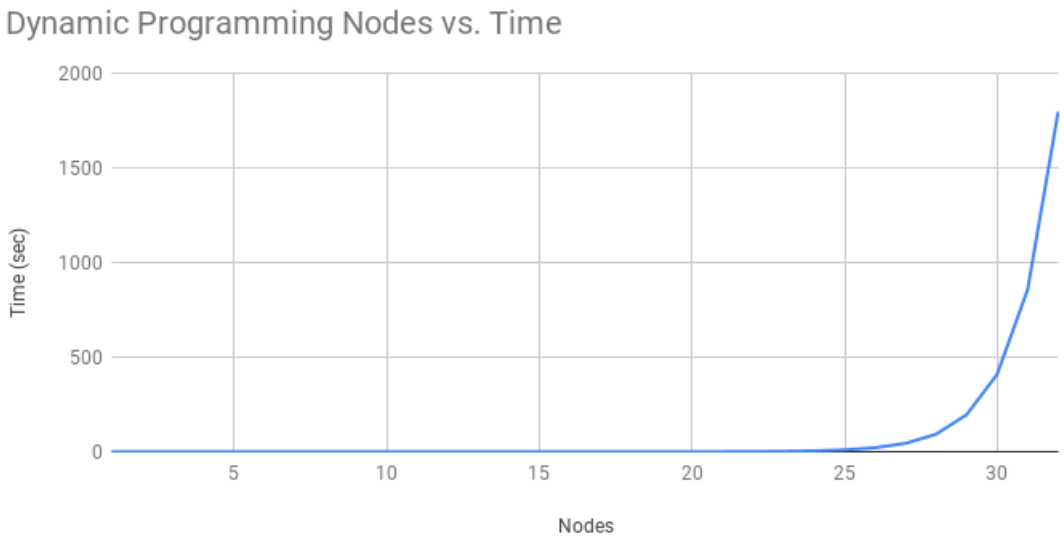

Figure 5.1: *Dynamic Programming Brute Force Nodes vs. Time*

5.1.2 Sliding Window

The accuracy of our *Sliding Window* algorithm depends upon the preorder and the window size. To keep things consistent we automatically generated a preorder based on the net outgoing edge weight of each node. In other words, the teams who won more games were ranked higher in the preorder. Table 5.2 is a preorder generated for the NFL 2018 season where $\alpha = 1$ and $\beta = 1$.

Using this preorder, for the *Sliding Window* algorithm on NFL 2018 data, we obtained the results shown in Table 5.3. Note that the optimal backedge weight found by the *Dynamic Programming Brute Force* algorithm is 49.

With window size 14, we were able to find the ranking with the optimal backedge weight. However, we had to increase to window size 16 to be able to consistently find the optimal ranking. The *Sliding Window* algorithm with window size 16 took 0.89 seconds to find an optimal ranking. The *Dynamic Programming Brute Force* algorithm took 1795 seconds (30 minutes). The ranking for window size 16 is shown in Table 5.3. Even though this ranking has the same backedge weight as *Dynamic Programming Brute Force*, it is a different ranking.

Figure 5.2 shows how the backedge weight changes with changes in the window size. Figure 5.3 shows how the time changes as the window size changes. As expected, this

Table 5.2: *NFL 2018 Sliding Window Preprocess*

$\alpha = 1 \quad \beta = 1$

Preorder Rank	Team
1.	Los Angeles Rams
2.	New Orleans Saints
3.	Chicago Bears
4.	Kansas City Chiefs
5.	Los Angeles Chargers
6.	Houston Texans
7.	New England Patriots
8.	Baltimore Ravens
9.	Dallas Cowboys
10.	Indianapolis Colts
11.	Seattle Seahawks
12.	Pittsburgh Steelers
13.	Philadelphia Eagles
14.	Tennessee Titans
15.	Minnesota Vikings
16.	Cleveland Browns
17.	Atlanta Falcons
18.	Carolina Panthers
19.	Miami Dolphins
20.	Washington Redskins
21.	Green Bay Packers
22.	Buffalo Bills
23.	Cincinnati Bengals
24.	Denver Broncos
25.	Detroit Lions
26.	Jacksonville Jaguars
27.	New York Giants
28.	Tampa Bay Buccaneers
29.	New York Jets
30.	Oakland Raiders
31.	San Francisco 49ers
32.	Arizona Cardinals

is also an exponential increase since sliding window relies on *Dynamic Programming Brute Force*.

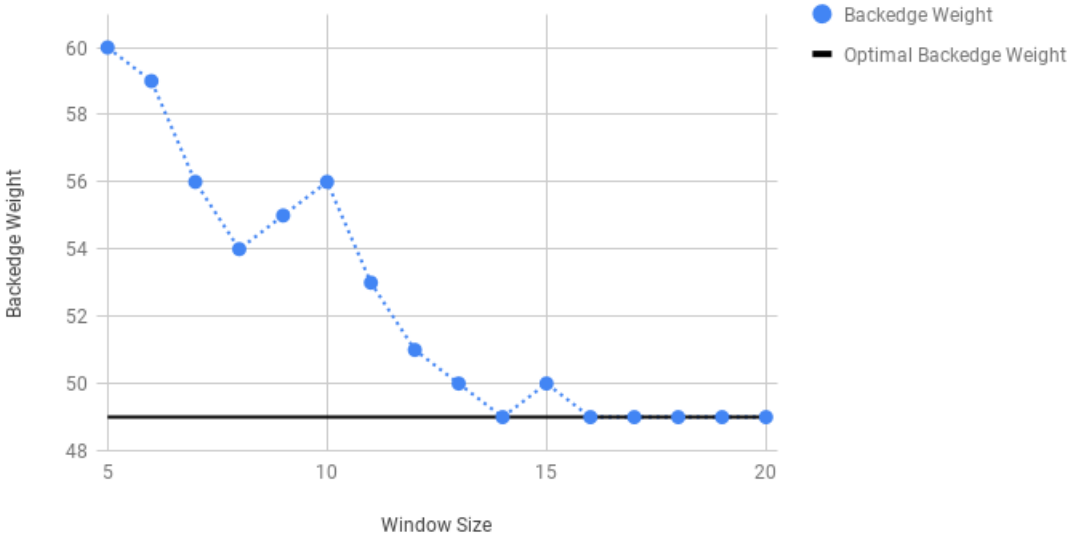

Figure 5.2: *Sliding Window NFL 2018 Window Size vs. Backedge Weight*

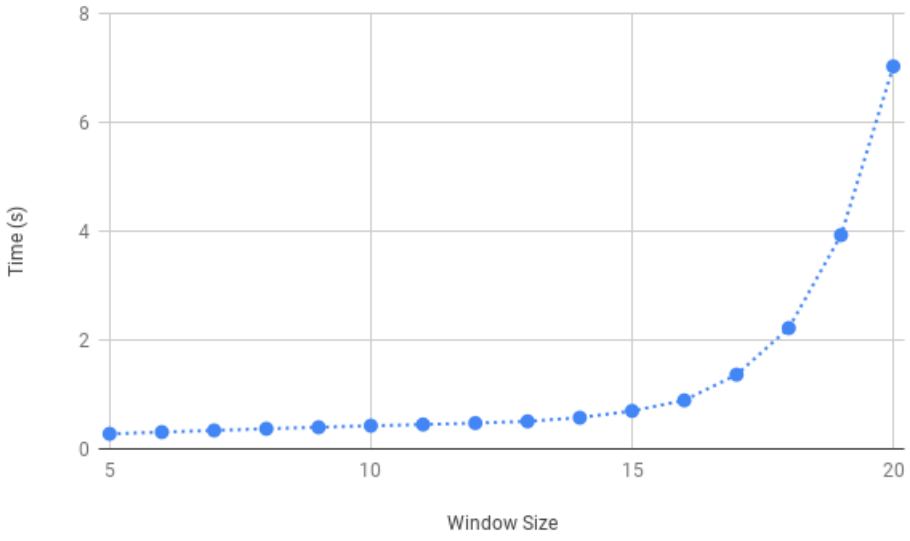

Figure 5.3: *Sliding Window NFL 2018 Window Size vs. Time*

Table 5.3: *NFL 2018 Sliding Window Ranking, Window Size = 14*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 19.3%

Rank	Team	Range of Correctness
1.	New England Patriots	[1, 2]
2.	New Orleans Saints	[1, 3]
3.	Chicago Bears	[2, 3]
4.	Los Angeles Rams	[4, 4]
5.	Kansas City Chiefs	[5, 5]
6.	Baltimore Ravens	[6, 7]
7.	Houston Texans	[2, 7]
8.	Denver Broncos	[8, 8]
9.	Los Angeles Chargers	[9, 9]
10.	Pittsburgh Steelers	[10, 10]
11.	Cleveland Browns	[11, 11]
12.	Cincinnati Bengals	[12, 12]
13.	Indianapolis Colts	[13, 13]
14.	Buffalo Bills	[14, 15]
15.	Seattle Seahawks	[10, 16]
16.	Tennessee Titans	[15, 17]
17.	Minnesota Vikings	[16, 18]
18.	Dallas Cowboys	[17, 18]
19.	Philadelphia Eagles	[19, 19]
20.	Atlanta Falcons	[20, 20]
21.	Washington Redskins	[21, 22]
22.	Detroit Lions	[19, 22]
23.	Arizona Cardinals	[23, 23]
24.	Green Bay Packers	[24, 25]
25.	Jacksonville Jaguars	[22, 25]
26.	Miami Dolphins	[26, 29]
27.	Carolina Panthers	[23, 27]
28.	New York Giants	[28, 28]
29.	Tampa Bay Buccaneers	[29, 30]
30.	New York Jets	[27, 32]
31.	San Francisco 49ers	[30, 31]
32.	Oakland Raiders	[32, 32]

5.1.3 Brute Force Pruning

Using the *Brute Force Pruning* algorithm described earlier, we are able to find good rankings much faster than the optimal *Dynamic Programming Brute Force* algorithm. Table 5.4 shows an optimal ranking found by the *Brute Force Pruning* algorithm for $\alpha = 1$ and $\beta = 1$. Note that although the ranking has the same backedge weight as the ones found using *Dynamic Programming Brute Force* and *Sliding Window*, the teams are in a different order.

Figure 5.4 shows the backedge weights for different combinations of limiting numbers and retries. The black line shows the optimal backedge weight, which was reached by multiple pruning configurations. Due to the randomness involved in choosing whether or not to keep a state, there is a lot of variation among the results. A low limiting number can sometimes give a better result than a higher limiting number. Clearly, there is no obvious benefit to increasing the limiting number as the backedge weight increased and decreased with no apparent pattern. However, there is a benefit to increasing the retry number. Although increases do not always offer an improvement, there was a large improvement between 1 retry and 10 retries. Once the retry number got to 50, *Brute Force Pruning* produced rankings that were all optimal, regardless of the limiting number. Therefore, the best configuration for *Brute Force Pruning* on a connected graph is leaving the limiting number fairly low and retrying the algorithm many times.

The quickest run to create an optimal ranking for this data was 10 retries with limiting number 500, which took 7.6 seconds. However, in order to consistently find the optimal ranking, 50 retries with a limiting number of at least 500 is needed. This took 38 seconds, which is a large improvement over the 1795 seconds (30 minutes) it took the *Dynamic Programming Brute Force* algorithm.

Figure 5.5 shows how the timing increased as the limiting number and number of retries increased. As expected, this was a linear increase.

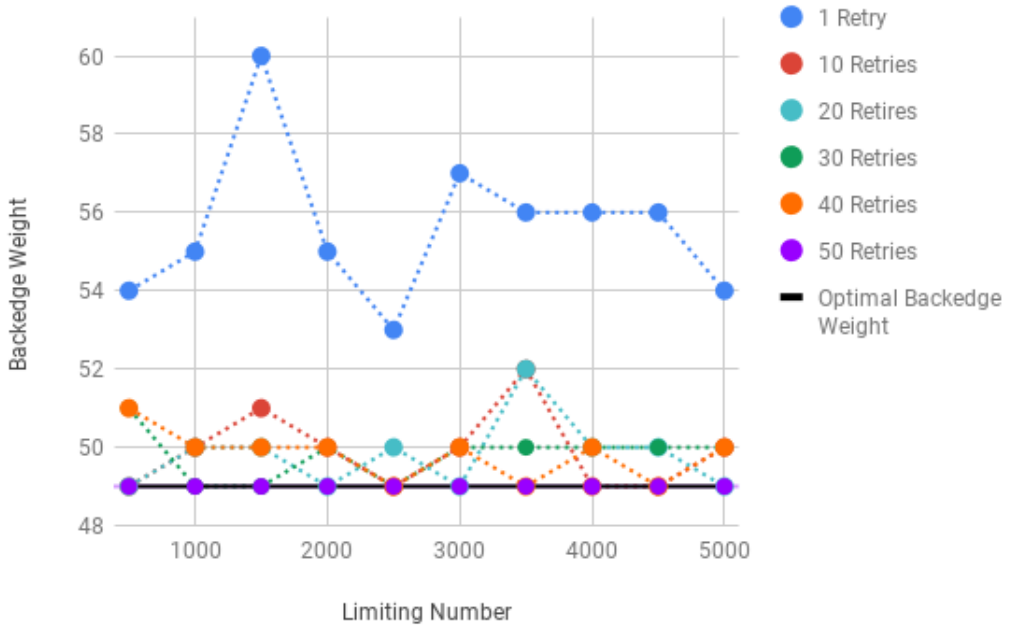

Figure 5.4: *Brute Force Pruning Backedge Weights for Different Configurations*

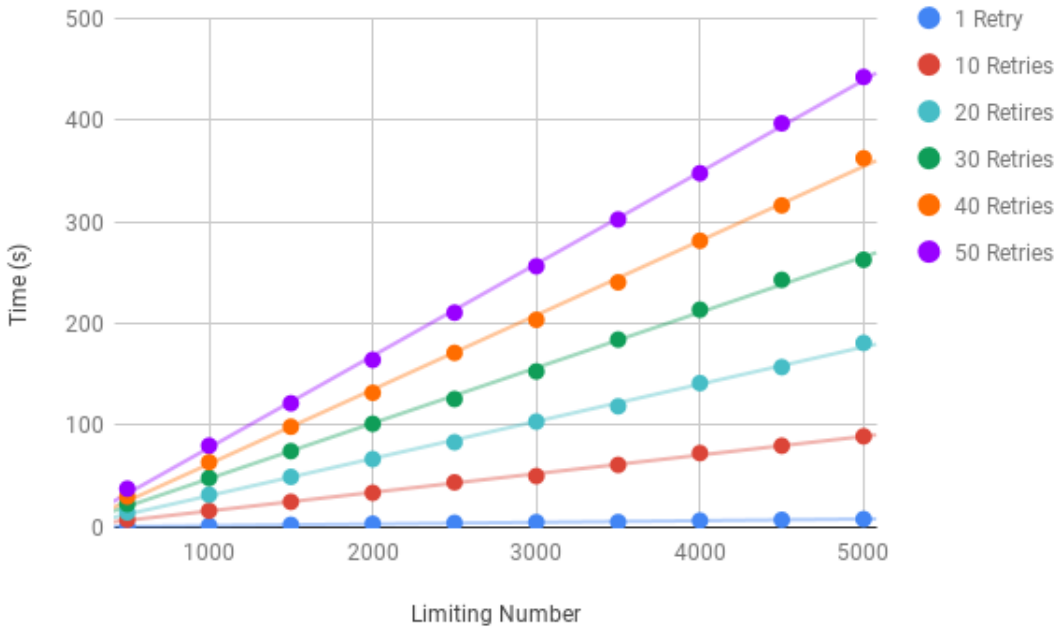

Figure 5.5: *Brute Force Pruning Timing for Different Configurations*

Table 5.4: *NFL 2018 Brute Force Pruning Ranking*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 19.3%

Rank	Team	Range of Correctness
1.	New England Patriots	[1, 2]
2.	New Orleans Saints	[1, 3]
3.	Chicago Bears	[2, 3]
4.	Los Angeles Rams	[4, 5]
5.	Houston Texans	[2, 9]
6.	Kansas City Chiefs	[5, 6]
7.	Baltimore Ravens	[7, 7]
8.	Los Angeles Chargers	[8, 8]
9.	Pittsburgh Steelers	[9, 9]
10.	Cleveland Browns	[10, 10]
11.	Denver Broncos	[11, 11]
12.	Cincinnati Bengals	[12, 12]
13.	Indianapolis Colts	[13, 14]
14.	Seattle Seahawks	[12, 16]
15.	Buffalo Bills	[14, 15]
16.	Tennessee Titans	[16, 16]
17.	Dallas Cowboys	[17, 18]
18.	Minnesota Vikings	[16, 18]
19.	Philadelphia Eagles	[19, 19]
20.	Atlanta Falcons	[20, 21]
21.	Detroit Lions	[19, 22]
22.	Washington Redskins	[21, 22]
23.	Arizona Cardinals	[23, 27]
24.	Carolina Panthers	[23, 25]
25.	Jacksonville Jaguars	[23, 25]
26.	New York Giants	[26, 26]
27.	Tampa Bay Buccaneers	[27, 28]
28.	Green Bay Packers	[24, 28]
29.	San Francisco 49ers	[29, 30]
30.	Miami Dolphins	[29, 30]
31.	Oakland Raiders	[31, 32]
32.	New York Jets	[31, 32]

5.1.4 Dynamic Programming Brute Force Pruning

Our second method of pruning uses the *Dynamic Programming Brute Force* algorithm to prune. Table 5.5 shows the best ranking from *Dynamic Programming Brute Force*, which was achieved with limiting number 500 and 1 retry and took 1 second. This ranking is not optimal.

This method was tested with the same configurations as *Brute Force Pruning* (limiting number up to 5000 in increments of 500, retry number up to 50 in increments of 10). However, for all configurations the same backedge weight of 51 was found. This is likely because this method has difficulty recovering from selecting suboptimal orderings early on. Once two teams have been placed in an order, that order cannot be adjusted later on. Nevertheless, this approximation is still able to find a ranking close to optimal very quickly. For other data sets, we were able to observe an improvement in backedge weight as the limiting number and retry number increased, but the optimal ranking was still not found.

Table 5.5: *NFL 2018 Dynamic Brute Force Pruning Ranking*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 20.1%

Rank	Team	Range of Correctness
1.	New England Patriots	[1, 2]
2.	New Orleans Saints	[1, 3]
3.	Chicago Bears	[2, 3]
4.	Los Angeles Rams	[4, 4]
5.	Los Angeles Chargers	[5, 5]
6.	Seattle Seahawks	[6, 6]
7.	Kansas City Chiefs	[7, 9]
8.	Minnesota Vikings	[7, 17]
9.	Houston Texans	[2, 11]
10.	Baltimore Ravens	[8, 10]
11.	Pittsburgh Steelers	[11, 11]
12.	Cleveland Browns	[12, 12]
13.	Cincinnati Bengals	[13, 13]
14.	Indianapolis Colts	[14, 14]
15.	Buffalo Bills	[15, 15]
16.	Tennessee Titans	[16, 16]
17.	Dallas Cowboys	[17, 17]
18.	Philadelphia Eagles	[18, 18]
19.	Atlanta Falcons	[19, 19]
20.	Washington Redskins	[20, 20]
21.	Jacksonville Jaguars	[21, 23]
22.	Detroit Lions	[18, 22]
23.	Carolina Panthers	[23, 23]
24.	New York Giants	[24, 28]
25.	Arizona Cardinals	[23, 25]
26.	Green Bay Packers	[26, 26]
27.	Miami Dolphins	[27, 27]
28.	New York Jets	[28, 30]
29.	Tampa Bay Buccaneers	[25, 29]
30.	San Francisco 49ers	[30, 30]
31.	Denver Broncos	[31, 31]
32.	Oakland Raiders	[32, 32]

5.2 Postprocesses

5.2.1 Range of Correctness (RoC)

RoC was designed as a simple tool that shows the possible mobility within a given ranking. In practice, *RoC* gives a rough idea of which positions a team can be moved to, however the mobility within the ranking is only accurate for the exact given ranking, not any variation with equivalent backedge weight. For example, in Table 5.1, the output of RoC shows that the Dallas Cowboys and Minnesota Vikings can be swapped. However, the output of RoC does not show all possible moves within the ranking. For instance, in Table 5.1 we can move the New York Jets from 30^{th} to 32^{nd} and move the San Francisco 49ers and Oakland Raiders from 31^{st} and 32^{nd} up by 1 rank to 30^{th} and 31^{st}. This will create an equivalent backedge weighted ranking because the New York Jets do not limit the San Francisco 49ers and the Oakland Raiders. However, RoC shows the Raiders fixed at the bottom because they must remain below the 49ers.

Noticeably, the density of the graphs also limits RoC ranges. For example, the ranges of teams in Table 5.6 are wider than ones in Table 5.1 because the NFL graph is denser than the NCAA graph.

Table 5.6: *NCAA Football 2018 Sliding Window Ranking for Top 25 Teams with Range of Correctness*

$\alpha = 0$ $\beta = 0$ Backedge Percent = 9.9%

Rank	Team	RoC
1.	Oklahoma	[1, 1]
2.	Army	[2, 3]
3.	Ohio	[1, 3]
4.	Buffalo	[4, 8]
5.	Alabama	[1, 5]
6.	Georgia	[6, 6]
7.	Missouri	[7, 9]
8.	Central Florida	[1, 8]
9.	Temple	[9, 10]
10.	Memphis	[9, 10]
11.	Houston	[11, 68]
12.	Clemson	[1, 13]
13.	Middle Tennessee State	[7, 14]
14.	North Carolina State	[13, 14]
15.	Marshall	[15, 39]
16.	Ohio State	[1, 17]
17.	Notre Dame	[1, 17]
18.	Michigan	[18, 18]
19.	Penn State	[19, 23]
20.	Utah State	[1, 47]
21.	Fresno State	[1, 29]
22.	Washington	[1, 22]
23.	Arizona State	[23, 25]
24.	Appalachian State	[20, 32]
25.	Washington State	[23, 25]

5.2.2 Tree of Correctness (ToC) Method

The main goal of all variations of the ToC Method is to select the best possible ranking, given several rankings that have equivalent backedge weights. We used the rankings shown in Table 5.1 and Table 5.6 as the base rankings to run all of the different *ToC Methods* and compared the results of each.

5.2.2.1 Original ToC Method

The *Original ToC Method* can improve the backedge weight of a suboptimal ranking such as Table 5.9 for NCAA 2018. If the provided ranking is already of optimal backedge weight, the *Original ToC Method* will retain its backedge weight.

Although it can improve the backedge weight of rankings, the *Original ToC Method* can be extremely time consuming. For a given ranking, this process will take a proportional amount of time to the number of nodes and the number of existing equivalent rankings. Therefore, the time consumption of the *Original ToC Method* depends on the density of the acyclic graphs it processes. Due to the large difference in number of teams, the NFL and NCAA rankings do not clearly demonstrate this time difference for the density. However, it still showed that the number of nodes affects the computational time as shown in Table 5.9.

To further analyze the *Original ToC method*, we checked the sums of the second metric for equivalent rankings. We found out that when using $\alpha = 1$ and $\beta = 1$ for NFL data, all of the sums were 0, while NCAA ones were not. This happened because all NFL teams play the same number of games, and those games combined are zero-sum. For a league similar to the NFL, the *Original ToC Method* should not be used because equivalent rankings cannot be distinguished by using the second metric. Additionally, in order to run this postprocess on larger data sets, such as NCAA data, we needed to limit the number of equivalent rankings compared, which decreases the accuracy of the results.

Table 5.7: *NFL 2018 Dynamic Programming Brute Force with Original ToC Method*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 19.3%

Rank	**Original ToC Method**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	New England Patriots	[1, 1]	New Orleans Saints	[1, 4]
2.	Chicago Bears	[2, 3]	New England Patriots	[1, 2]
3.	New Orleans Saints	[1, 3]	Houston Texans	[3, 8]
4.	Los Angeles Rams	[4, 5]	Chicago Bears	[3, 4]
5.	Houston Texans	[2, 8]	Los Angeles Rams	[5, 5]
6.	Kansas City Chiefs	[5, 6]	Kansas City Chiefs	[6, 6]
7.	Baltimore Ravens	[7, 7]	Baltimore Ravens	[7, 7]
8.	Los Angeles Chargers	[8, 8]	Los Angeles Chargers	[8, 8]
9.	Cleveland Browns	[9, 9]	Cleveland Browns	[9, 9]
10.	Denver Broncos	[10, 10]	Denver Broncos	[10, 10]
11.	Pittsburgh Steelers	[11, 11]	Pittsburgh Steelers	[11, 12]
12.	Cincinnati Bengals	[12, 12]	Seattle Seahawks	[11, 16]
13.	Indianapolis Colts	[13, 13]	Cincinnati Bengals	[12, 13]
14.	Buffalo Bills	[14, 15]	Indianapolis Colts	[14, 14]
15.	Seattle Seahawks	[11, 15]	Buffalo Bills	[15, 15]
16.	Minnesota Vikings	[16, 18]	Tennessee Titans	[16, 16]
17.	Tennessee Titans	[15, 17]	Dallas Cowboys	[17, 18]
18.	Dallas Cowboys	[18, 18]	Minnesota Vikings	[16, 18]
19.	Detroit Lions	[19, 22]	Philadelphia Eagles	[19, 19]
20.	Philadelphia Eagles	[19, 20]	Atlanta Falcons	[20, 20]
21.	Atlanta Falcons	[21, 21]	Washington Redskins	[21, 21]
22.	Washington Redskins	[22, 22]	Jacksonville Jaguars	[22, 24]
23.	Arizona Cardinals	[23, 24]	Detroit Lions	[19, 23]
24.	Jacksonville Jaguars	[23, 25]	Carolina Panthers	[24, 24]
25.	Green Bay Packers	[24, 25]	New York Giants	[25, 25]
26.	Miami Dolphins	[26, 26]	Tampa Bay Buccaneers	[26, 30]
27.	New York Jets	[27, 32]	Arizona Cardinals	[24, 27]
28.	Carolina Panthers	[23, 28]	Green Bay Packers	[28, 28]
29.	New York Giants	[29, 29]	Miami Dolphins	[29, 29]
30.	Tampa Bay Buccaneers	[30, 30]	New York Jets	[30, 32]
31.	San Francisco 49ers	[31, 31]	San Francisco 49ers	[29, 31]
32.	Oakland Raiders	[32, 32]	Oakland Raiders	[32, 32]

Table 5.8: *NCAA Football 2018 Sliding Window Ranking for Top 25 Teams with Original ToC Method*

$\alpha = 0 \quad \beta = 0$

Rank	**Original ToC Method**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	North Texas	[1, 39]	Oklahoma	[1, 1]
2.	Washington	[1, 2]	Army	[2, 3]
3.	Washington State	[3, 4]	Ohio	[1, 3]
4.	Arizona State	[3, 4]	Buffalo	[4, 8]
5.	Utah	[5, 9]	Alabama	[1, 5]
6.	Fresno State	[1, 6]	Georgia	[6, 6]
7.	Boise State	[7, 7]	Missouri	[7, 9]
8.	Utah State	[8, 47]	Central Florida	[1, 8]
9.	Notre Dame	[1, 9]	Temple	[9, 10]
10.	Stanford	[10, 10]	Memphis	[9, 10]
11.	Oregon	[11, 11]	Houston	[11, 68]
12.	California	[12, 66]	Clemson	[1, 13]
13.	Ohio State	[1, 13]	Middle Tennessee State	[7, 14]
14.	Michigan	[14, 14]	North Carolina State	[13, 14]
15.	Northwestern	[15, 16]	Marshall	[15, 39]
16.	Penn State	[15, 16]	Ohio State	[1, 17]
17.	Iowa	[17, 19]	Notre Dame	[1, 17]
18.	Appalachian State	[17, 18]	Michigan	[18, 18]
19.	Troy	[19, 19]	Penn State	[19, 23]
20.	Nebraska	[20, 20]	Utah State	[1, 47]
21.	Michigan State	[21, 64]	Fresno State	[1, 29]
22.	Clemson	[1, 22]	Washington	[1, 22]
23.	Syracuse	[23, 24]	Arizona State	[23, 25]
24.	Georgia Southern	[23, 33]	Appalachian State	[20, 32]
25.	North Carolina State	[24, 40]	Washington State	[23, 25]

Table 5.9: *Backedge Weight Percentage for Original ToC Method vs No Postprocess*

	Original ToC Method		**No Postprocess**
	Time (s)	Backedge Weight Percentage	Backedge Weight Percentage
NFL 2018 (Table 5.1)	30.6	19.3%	19.3%
NCAA 2018	558.7	7.1%	9.9%

5.2.2.2 Big Data ToC Method

The *Big Data ToC Method* runs *Breath First Search (BFS)* through the graph and *ToC structure*, and therefore it runs quickly. As shown in Table 5.10, it ran in less than 3 milliseconds for both sports; it ran quicker on NFL than on NCAA because the number of teams in NFL is 3-4 times less than one of NCAA.

However, the backedge weight percentage was not significantly changed by running *Big Data ToC Method*. It decreased less than 1 percent on the NCAA dataset. This means that sorting nodes by the density of a ranking's corresponding acyclic graph does not significantly decrease backedge weight.

Table 5.10: *Backedge Weight Percentage for Big Data ToC Method vs No Postprocess*

	Big Data ToC Method		**No Postprocess**
	Time	Backedge Weight Percentage	Backedge Weight Percentage
NFL 2018	< 1 ms	19.3%	19.3%
NCAA 2018	3 ms	9.6%	9.9%

Table 5.11: *NFL 2018 Dynamic Programming Brute Force Ranking with Big Data ToC Method*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percentage = 19.3%

Rank	**Big Data ToC Method**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	New England Patriots	[1, 2]	New Orleans Saints	[1, 4]
2.	New Orleans Saints	[1, 3]	New England Patriots	[1, 2]
3.	Chicago Bears	[2, 3]	Houston Texans	[3, 8]
4.	Los Angeles Rams	[4, 4]	Chicago Bears	[3, 4]
5.	Kansas City Chiefs	[5, 5]	Los Angeles Rams	[5, 5]
6.	Baltimore Ravens	[6, 7]	Kansas City Chiefs	[6, 6]
7.	Houston Texans	[2, 8]	Baltimore Ravens	[7, 7]
8.	Los Angeles Chargers	[7, 8]	Los Angeles Chargers	[8, 8]
9.	Cleveland Browns	[9, 9]	Cleveland Browns	[9, 9]
10.	Denver Broncos	[10, 10]	Denver Broncos	[10, 10]
11.	Seattle Seahawks	[11, 16]	Pittsburgh Steelers	[11, 12]
12.	Pittsburgh Steelers	[11, 12]	Seattle Seahawks	[11, 16]
13.	Cincinnati Bengals	[13, 13]	Cincinnati Bengals	[12, 13]
14.	Indianapolis Colts	[14, 14]	Indianapolis Colts	[14, 14]
15.	Buffalo Bills	[15, 15]	Buffalo Bills	[15, 15]
16.	Tennessee Titans	[16, 16]	Tennessee Titans	[16, 16]
17.	Dallas Cowboys	[17, 18]	Dallas Cowboys	[17, 18]
18.	Minnesota Vikings	[16, 18]	Minnesota Vikings	[16, 18]
19.	Philadelphia Eagles	[19, 19]	Philadelphia Eagles	[19, 19]
20.	Atlanta Falcons	[20, 20]	Atlanta Falcons	[20, 20]
21.	Washington Redskins	[21, 22]	Washington Redskins	[21, 21]
22.	Detroit Lions	[19, 23]	Jacksonville Jaguars	[22, 24]
23.	Jacksonville Jaguars	[22, 26]	Detroit Lions	[19, 23]
24.	Arizona Cardinals	[23, 24]	Carolina Panthers	[24, 24]
25.	Green Bay Packers	[25, 27]	New York Giants	[25, 25]
26.	Carolina Panthers	[23, 26]	Tampa Bay Buccaneers	[26, 30]
27.	New York Giants	[27, 28]	Arizona Cardinals	[24, 27]
28.	Miami Dolphins	[26, 30]	Green Bay Packers	[28, 28]
29.	Tampa Bay Buccaneers	[28, 29]	Miami Dolphins	[29, 29]
30.	San Francisco 49ers	[30, 31]	New York Jets	[30, 32]
31.	New York Jets	[29, 32]	San Francisco 49ers	[29, 31]
32.	Oakland Raiders	[31, 32]	Oakland Raiders	[32, 32]

Table 5.12: *NCAA Football 2018 Sliding Window Ranking for Top 25 Teams with Big Data ToC Method*

$\alpha = 0 \quad \beta = 0$

Rank	**Big Data ToC Method**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	Ohio State	[1, 2]	Oklahoma	[1, 1]
2.	Notre Dame	[1, 2]	Army	[2, 3]
3.	Michigan	[3, 5]	Ohio	[1, 3]
4.	Alabama	[1, 8]	Buffalo	[4, 8]
5.	Oklahoma	[1, 14]	Alabama	[1, 5]
6.	Penn State	[4, 10]	Georgia	[6, 6]
7.	Washington	[1, 13]	Missouri	[7, 9]
8.	Northwestern	[4, 10]	Central Florida	[1, 8]
9.	Georgia	[5, 16]	Temple	[9, 10]
10.	Central Florida	[1, 24]	Memphis	[9, 10]
11.	Iowa	[9, 32]	Houston	[11, 68]
12.	Clemson	[1, 29]	Clemson	[1, 13]
13.	Fresno State	[1, 15]	Middle Tennessee State	[7, 14]
14.	Washington State	[8, 19]	North Carolina State	[13, 14]
15.	Army	[6, 23]	Marshall	[15, 39]
16.	Boise State	[14, 25]	Ohio State	[1, 17]
17.	Middle Tennessee State	[10, 30]	Notre Dame	[1, 17]
18.	West Virginia	[6, 35]	Michigan	[18, 18]
19.	Arizona State	[8, 19]	Penn State	[19, 23]
20.	Utah	[20, 45]	Utah State	[1, 47]
21.	Appalachian State	[7, 25]	Fresno State	[1, 29]
22.	Missouri	[10, 38]	Washington	[1, 22]
23.	Ohio	[1, 23]	Arizona State	[23, 25]
24.	Buffalo	[24, 24]	Appalachian State	[20, 32]
25.	Temple	[25, 28]	Washington State	[23, 25]

5.2.2.3 RoC Reorder

Similarly to the *Original ToC Method*, we want to test how the processing time and reduction of backedge weight changes based on how sparse the graph is. We ran a similar experiment for *RoC Reorder* on NFL 2018 data.

The time complexity to run this method depends on how big the original graph is and how many chains of inputs and outputs of the subprocess there are. From the NFL result in Table 5.15, the method took essentially no time to finish. This indicates that rankings repeat in this process really quickly. In addition, the reduced backedge weight

percentage was close to one from the *Original ToC Method*, which we believe it was the minimum possible reduced backedge weight because it tried every possible equivalent graph.

From the NCAA Football result, this method reduced the backedge weight 15 to 20 percent compared to the input ranking, which is much better than the *Big Data ToC Method*. Additionally, it took only 36 milliseconds to get a new ranking with less backedge weight percentage, which is faster than *Original ToC Method*.

Table 5.13: *NFL 2018 Dynamic Programming Brute Force Ranking with RoC Reorder*

$\alpha = 1 \quad \beta = 1 \quad$ Backedge Percent = 19.3%

Rank	**RoC Reorder**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	New Orleans Saints	[1, 4]	New Orleans Saints	[1, 4]
2.	New England Patriots	[1, 2]	New England Patriots	[1, 2]
3.	Houston Texans	[3, 8]	Houston Texans	[3, 8]
4.	Chicago Bears	[3, 4]	Chicago Bears	[3, 4]
5.	Los Angeles Rams	[5, 5]	Los Angeles Rams	[5, 5]
6.	Kansas City Chiefs	[6, 6]	Kansas City Chiefs	[6, 6]
7.	Baltimore Ravens	[7, 7]	Baltimore Ravens	[7, 7]
8.	Los Angeles Chargers	[8, 8]	Los Angeles Chargers	[8, 8]
9.	Cleveland Browns	[9, 9]	Cleveland Browns	[9, 9]
10.	Denver Broncos	[10, 10]	Denver Broncos	[10, 10]
11.	Pittsburgh Steelers	[11, 12]	Pittsburgh Steelers	[11, 12]
12.	Seattle Seahawks	[11, 16]	Seattle Seahawks	[11, 16]
13.	Cincinnati Bengals	[12, 13]	Cincinnati Bengals	[12, 13]
14.	Indianapolis Colts	[14, 14]	Indianapolis Colts	[14, 14]
15.	Buffalo Bills	[15, 15]	Buffalo Bills	[15, 15]
16.	Tennessee Titans	[16, 16]	Tennessee Titans	[16, 16]
17.	Dallas Cowboys	[17, 18]	Dallas Cowboys	[17, 18]
18.	Minnesota Vikings	[16, 18]	Minnesota Vikings	[16, 18]
19.	Philadelphia Eagles	[19, 19]	Philadelphia Eagles	[19, 19]
20.	Atlanta Falcons	[20, 20]	Atlanta Falcons	[20, 20]
21.	Washington Redskins	[21, 21]	Washington Redskins	[21, 21]
22.	Jacksonville Jaguars	[22, 24]	Jacksonville Jaguars	[22, 24]
23.	Detroit Lions	[19, 23]	Detroit Lions	[19, 23]
24.	Carolina Panthers	[24, 24]	Carolina Panthers	[24, 24]
25.	New York Giants	[25, 25]	New York Giants	[25, 25]
26.	Tampa Bay Buccaneers	[26, 30]	Tampa Bay Buccaneers	[26, 30]
27.	Arizona Cardinals	[24, 27]	Arizona Cardinals	[24, 27]
28.	Green Bay Packers	[28, 28]	Green Bay Packers	[28, 28]
29.	Miami Dolphins	[29, 29]	Miami Dolphins	[29, 29]
30.	New York Jets	[30, 32]	New York Jets	[30, 32]
31.	San Francisco 49ers	[29, 31]	San Francisco 49ers	[29, 31]
32.	Oakland Raiders	[32, 32]	Oakland Raiders	[32, 32]

Table 5.14: *NCAA Football 2018 Sliding Window Ranking for Top 25 Teams with RoC Reorder*

$\alpha = 0 \quad \beta = 0$

Rank	**RoC Reorder**		**No Postprocess**	
	Team	RoC	Team	RoC
1.	Alabama	[1, 2]	Oklahoma	[1, 1]
2.	Clemson	[1, 18]	Army	[2, 3]
3.	Georgia	[2, 4]	Ohio	[1, 3]
4.	Central Florida	[1, 19]	Buffalo	[4, 8]
5.	Missouri	[4, 19]	Alabama	[1, 5]
6.	Ohio State	[1, 7]	Georgia	[6, 6]
7.	Notre Dame	[1, 7]	Missouri	[7, 9]
8.	Michigan	[8, 12]	Central Florida	[1, 8]
9.	Fresno State	[1, 10]	Temple	[9, 10]
10.	Oklahoma	[1, 17]	Memphis	[9, 10]
11.	Boise State	[10, 11]	Houston	[11, 68]
12.	Utah State	[12, 35]	Clemson	[1, 13]
13.	Penn State	[9, 13]	Middle Tennessee State	[7, 14]
14.	Appalachian State	[14, 30]	North Carolina State	[13, 14]
15.	Louisiana State	[2, 15]	Marshall	[15, 39]
16.	Mississippi State	[16, 18]	Ohio State	[1, 17]
17.	Washington	[1, 21]	Notre Dame	[1, 17]
18.	West Virginia	[11, 45]	Michigan	[18, 18]
19.	Texas A&M	[17, 24]	Penn State	[19, 23]
20.	Memphis	[6, 39]	Utah State	[1, 47]
21.	Ohio	[1, 36]	Fresno State	[1, 29]
22.	Washington State	[18, 35]	Washington	[1, 22]
23.	Middle Tennessee State	[4, 49]	Arizona State	[23, 25]
24.	Syracuse	[8, 28]	Appalachian State	[20, 32]
25.	Kentucky	[20, 25]	Washington State	[23, 25]

Table 5.15: *Backedge Weight Percentage for RoC Reorder Method vs No Postprocess*

	RoC Reorder		**No Postprocess**
	Time	Backedge Weight Percentage	Backedge Weight Percentage
NFL 2018 (Table 5.1)	< 1 ms	19.3%	19.3%
NCAA 2018	36 ms	8.3%	9.9%

5.3 Ranking Comparisons

To give our rankings an objective version of the “eye test," we used external rankings from ESPN, NBC Sports, the AP Poll, NFL.com, and NBA.com as a comparison. While these rankings are not specifically created to minimize backedge weight, they are an indicator of approximately where teams should fall in a “good" ranking. Additionally, this method of comparing rankings allows us to look at these external rankings and estimate how influential game recency and point differential are in these rankings.

For all professional sports, we ran these tests using the *Dynamic Programming Brute Force* algorithm, with the *RoC Reorder* postprocess. For collegiate sports, the algorithm used was *Brute Force Pruning* with the *RoC Reorder* postprocess.

5.3.1 National Football League

Table 5.16: *NFL 2018 Ranking Comparison Scores for ESPN*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	190	188	196	192	198
0.25	194	188	192	202	200
0.5	252	134	144	144	138
0.75	140	**128**	**128**	130	134
1.0	140	**128**	**128**	130	134

Table 5.17: *NFL 2018 Ranking Comparison Scores for NFL.com*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	130	134	146	138	152
0.25	138	134	134	148	160
0.5	138	**96**	114	114	112
0.75	136	118	118	128	136
1.0	138	118	118	134	136

Our two external rankings for the NFL (ESPN [30] and NFL.com [31]) both provided a comparison score of 140. This means that on average, a team would be placed in one ranking 4.1875 ranks away from where they were placed in the other ranking. Putting our own rankings against these external ones, we find that our lowest comparison score when comparing against ESPN is 128, which occurs when $\alpha = 0.75$ or $\alpha = 1.0$ and $\beta = 0.25$ or $\beta = 0.5$. This 128 score means that our ranking placed teams an average of 4.0 ranks

away form their placement in the ESPN ranking. Against NFL.com, we find that our closest ranking, produced when $\alpha = 0.5$ and $\beta = 0.25$, has an even lower score of 96, meaning that our ranking placed teams an average of 3.0 ranks away from NFL.com's placement in their ranking.

When comparing against the ESPN ranking, our highest difference in rank for a single team is 13 for the Buffalo Bills. Most teams in this ranking are either ranked within 1 to 3 ranks from their placement by ESPN, or are much further away, usually 9 to 13 ranks.

Table 5.18: *NFL 2018 Ranking Comparison between ESPN and our Ranking*

$\alpha = 0.75 \quad \beta = 0.25 \quad$ Backedge Percent = 16.2%

Rank	Our Ranking	ESPN	Rank Difference
1.	New Orleans Saints	Los Angeles Rams	[+3]
2.	New England Patriots	New Orleans Saints	[-1]
3.	Chicago Bears	Kansas City Chiefs	[+2]
4.	Los Angeles Rams	New England Patriots	[-2]
5.	Kansas City Chiefs	Los Angeles Chargers	[+2]
6.	Baltimore Ravens	Houston Texans	[+2]
7.	Los Angeles Chargers	Pittsburgh Steelers	[+3]
8.	Houston Texans	Chicago Bears	[-5]
9.	Denver Broncos	Seattle Seahawks	[+2]
10.	Pittsburgh Steelers	Baltimore Ravens	[-4]
11.	Seattle Seahawks	Dallas Cowboys	[+7]
12.	Cleveland Browns	Minnesota Vikings	[+5]
13.	Cincinnati Bengals	Indianapolis Colts	[+1]
14.	Indianapolis Colts	Carolina Panthers	[+9]
15.	Buffalo Bills	Denver Broncos	[-6]
16.	Tennessee Titans	Tennessee Titans	[0]
17.	Minnesota Vikings	Philadelphia Eagles	[+2]
18.	Dallas Cowboys	Washington Redskins	[+4]
19.	Philadelphia Eagles	Green Bay Packers	[+9]
20.	Atlanta Falcons	Miami Dolphins	[+9]
21.	Detroit Lions	Atlanta Falcons	[-1]
22.	Washington Redskins	Cleveland Browns	[-10]
23.	Carolina Panthers	Tampa Bay Buccaneers	[+3]
24.	Jacksonville Jaguars	Jacksonville Jaguars	[0]
25.	New York Giants	Cincinnati Bengals	[-12]
26.	Tampa Bay Buccaneers	New York Giants	[-1]
27.	Arizona Cardinals	Detroit Lions	[-6]
28.	Green Bay Packers	Buffalo Bills	[-13]
29.	Miami Dolphins	New York Jets	[+1]
30.	New York Jets	Arizona Cardinals	[-3]
31.	San Francisco 49ers	San Francisco 49ers	[0]
32.	Oakland Raiders	Oakland Raiders	[0]

Compared against NFL.com, our highest single team difference is 14 with the Cincinnati Bengals. However, unlike the ESPN comparison, this appears to be an outlier, as all other teams are ranked within 6 places of their ranking from NFL.com.

Table 5.19: *NFL 2018 Ranking Comparison between NFL.com and our Ranking*

$\alpha = 0.5 \quad \beta = 0.25 \quad$ Backedge Percent = 15.8%

Rank	Our Ranking	NFL.com	Rank Difference
1.	New Orleans Saints	New Orleans Saints	[0]
2.	New England Patriots	Chicago Bears	[+1]
3.	Chicago Bears	Los Angeles Rams	[+1]
4.	Los Angeles Rams	Kansas City Chiefs	[+5]
5.	Houston Texans	New England Patriots	[-3]
6.	Baltimore Ravens	Baltimore Ravens	[0]
7.	Los Angeles Chargers	Los Angeles Chargers	[0]
8.	Seattle Seahawks	Indianapolis Colts	[+5]
9.	Kansas City Chiefs	Houston Texans	[-4]
10.	Pittsburgh Steelers	Seattle Seahawks	[-2]
11.	Cleveland Browns	Dallas Cowboys	[+6]
12.	Cincinnati Bengals	Philadelphia Eagles	[+6]
13.	Indianapolis Colts	Pittsburgh Steelers	[-3]
14.	Buffalo Bills	Minnesota Vikings	[+2]
15.	Tennessee Titans	Tennessee Titans	[0]
16.	Minnesota Vikings	Cleveland Browns	[-5]
17.	Dallas Cowboys	Atlanta Falcons	[+4]
18.	Philadelphia Eagles	Buffalo Bills	[-4]
19.	Detroit Lions	Detroit Lions	[0]
20.	Green Bay Packers	New York Giants	[+5]
21.	Atlanta Falcons	Green Bay Packers	[-1]
22.	Washington Redskins	Washington Redskins	[0]
23.	Carolina Panthers	Carolina Panthers	[0]
24.	Jacksonville Jaguars	San Francisco 49ers	[+5]
25.	New York Giants	Oakland Raiders	[+5]
26.	Tampa Bay Buccaneers	Cincinnati Bengals	[-14]
27.	Miami Dolphins	Tampa Bay Buccaneers	[-1]
28.	New York Jets	Denver Broncos	[+3]
29.	San Francisco 49ers	Miami Dolphins	[-2]
30.	Oakland Raiders	Jacksonville Jaguars	[-6]
31.	Denver Broncos	New York Jets	[+3]
32.	Arizona Cardinals	Arizona Cardinals	[0]

Additionally, looking at the backedge weight of these rankings provides further insight. By minimizing the backedge weight of our rankings, we are effectively minimizing the number and size of upsets in our ranking over the course of the season. The ESPN ranking, using $\alpha = 0.75$ and $\beta = 0.25$, has a backedge weight percentage of 25.73%, while our own ranking with the same α and β has a backedge percentage of 16.20%. Comparing

to NFL.com, using $\alpha = 0.5$ and $\beta = 0.25$ gives a backedge weight percentage of 22.56%, while our own rankings has as 15.84% backedge weight. This means that our rankings contain a lower number of upsets, and that the upsets that did occur were closer scoring games that happened earlier in the season compared to the ESPN and NFL.com rankings. In Table 5.20, we see that this trend is repeated across a wide range of α and β values.

Table 5.20: *NFL 2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	ESPN	NFL.com	ESPN	NFL.com
0.0	0.0	9.89%	23.79%	20.39%	190	130
0.5	0.5	17.18%	27.16%	24.16%	144	114
1.0	1.0	19.29%	29.13%	26.38%	134	136
0.75	0.25	15.84%	25.73%	22.56%	128	118
0.5	0.25	16.20%	25.73%	22.56%	192	96

For both external rankings, our closest ranking was produced with $\beta = 0.25$, signifying that these rankings consider point differential as a significant factor. Meanwhile, α in these rankings was 0.5 for NFL.com and 0.75 or 1.0 for ESPN. This means that ESPN's rankings likely give little or no consideration to the recency of a game, while the NFL.com analysts likely consider it, but it may not be a major factor in their rankings.

5.3.2 Major League Baseball

Table 5.21: *MLB 2018 Ranking Comparison Scores for ESPN*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	118	120	130	136	136
0.25	122	120	120	136	134
0.5	112	118	**98**	116	130
0.75	128	114	104	124	130
1.0	130	110	104	100	114

Table 5.22: *MLB 2018 Ranking Comparison Scores for NBC Sports*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	118	122	130	128	128
0.25	128	116	116	128	122
0.5	110	114	**94**	110	116
0.75	138	114	106	116	116
1.0	140	118	108	**94**	104

Against the ESPN [32] and NBC Sports [33] rankings for the MLB 2018 season, we found that our closest rankings gave comparison scores of 98 when compared to ESPN, and 94 when compared to NBC Sports. This led to an average displacement of teams by 3.267 and 3.133 ranks, respectively. This was significantly higher than the comparison between the two external rankings, which gave a comparison score of 28, and an average team displacement of 0.933 ranks.

Comparing ourselves against ESPN, we find that our highest single team displacement was by 8 ranks, which was the case for several teams. Most of the different displacements were by either 1 to 3 ranks, or by about 6 to 8 ranks.

Table 5.23: *MLB 2018 Ranking Comparison between ESPN and our Ranking*

$\alpha = 0.5 \quad \beta = 0.5 \quad$ Backedge Percent = 36.9%

Rank	Our Ranking	ESPN	Rank Difference
1.	New York Yankees	Houston Astros	[+1]
2.	Houston Astros	Boston Red Sox	[+1]
3.	Boston Red Sox	New York Yankees	[-2]
4.	Chicago Cubs	Oakland Athletics	[+8]
5.	Los Angeles Dodgers	Cleveland Indians	[+8]
6.	Milwaukee Brewers	Chicago Cubs	[-2]
7.	St. Louis Cardinals	Milwaukee Brewers	[-1]
8.	Colorado Rockies	Los Angeles Dodgers	[-3]
9.	Seattle Mariners	Atlanta Braves	[+1]
10.	Atlanta Braves	Colorado Rockies	[-2]
11.	Tampa Bay Rays	Tampa Bay Rays	[0]
12.	Oakland Athletics	St. Louis Cardinals	[-5]
13.	Cleveland Indians	Washington Nationals	[+8]
14.	New York Mets	Arizona Diamondbacks	[+2]
15.	San Francisco Giants	Seattle Mariners	[-6]
16.	Arizona Diamondbacks	Pittsburgh Pirates	[+8]
17.	Los Angeles Angels	Los Angeles Angels	[0]
18.	Texas Rangers	Philadelphia Phillies	[+4]
19.	Minnesota Twins	New York Mets	[-5]
20.	Toronto Blue Jays	Minnesota Twins	[-1]
21.	Washington Nationals	San Francisco Giants	[-6]
22.	Philadelphia Phillies	Toronto Blue Jays	[-2]
23.	San Diego Padres	Texas Rangers	[-5]
24.	Pittsburgh Pirates	Cincinnati Reds	[+4]
25.	Miami Marlins	Detroit Tigers	[+2]
26.	Kansas City Royals	San Diego Padres	[-3]
27.	Detroit Tigers	Chicago White Sox	[+2]
28.	Cincinnati Reds	Miami Marlins	[-3]
29.	Chicago White Sox	Kansas City Royals	[-3]
30.	Baltimore Orioles	Baltimore Orioles	[0]

When we compare our rankings to the NBCSports ranking, we find that most teams are displaced by 2 to 7 ranks.

Table 5.24: *MLB 2018 Ranking Comparison between NBC Sports and our Ranking*

$\alpha = 0.5 \quad \beta = 0.5 \quad$ Backedge Percent = 36.9%

Rank	Our Ranking	NBC Sports	Rank Difference
1.	New York Yankees	Boston Red Sox	[+2]
2.	Houston Astros	Houston Astros	[0]
3.	Boston Red Sox	New York Yankees	[-2]
4.	Chicago Cubs	Milwaukee Brewers	[+2]
5.	Los Angeles Dodgers	Oakland Athletics	[+7]
6.	Milwaukee Brewers	Cleveland Indians	[+7]
7.	St. Louis Cardinals	Los Angeles Dodgers	[-2]
8.	Colorado Rockies	Chicago Cubs	[-4]
9.	Seattle Mariners	Atlanta Braves	[+1]
10.	Atlanta Braves	Colorado Rockies	[-2]
11.	Tampa Bay Rays	Tampa Bay Rays	[0]
12.	Oakland Athletics	St. Louis Cardinals	[-5]
13.	Cleveland Indians	Seattle Mariners	[-4]
14.	New York Mets	Pittsburgh Pirates	[+10]
15.	San Francisco Giants	Washington Nationals	[+6]
16.	Arizona Diamondbacks	Arizona Diamondbacks	[0]
17.	Los Angeles Angels	Philadelphia Phillies	[+5]
18.	Texas Rangers	Los Angeles Angels	[-1]
19.	Minnesota Twins	New York Mets	[-5]
20.	Toronto Blue Jays	Minnesota Twins	[-1]
21.	Washington Nationals	Toronto Blue Jays	[-1]
22.	Philadelphia Phillies	San Francisco Giants	[-7]
23.	San Diego Padres	Cincinnati Reds	[+5]
24.	Pittsburgh Pirates	Texas Rangers	[-6]
25.	Miami Marlins	San Diego Padres	[-2]
26.	Kansas City Royals	Detroit Tigers	[+1]
27.	Detroit Tigers	Miami Marlins	[-2]
28.	Cincinnati Reds	Chicago White Sox	[+1]
29.	Chicago White Sox	Kansas City Royals	[-3]
30.	Baltimore Orioles	Baltimore Orioles	[0]

Looking at the backedge weight of each ranking for $\alpha = 0.5$ and $\beta = 0.5$, which was the point where our rankings were closest to both the ESPN and NBC Sports rankings, we find that our backedge weight percentage is lower, producing 36.93% backedge weight, compared to 38.83% for ESPN and 39.49% for NBC Sports.

Table 5.25: *MLB 2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	ESPN	NBCSports	ESPN	NBCSports
0.0	0.0	34.74%	37.37%	38.24%	118	118
0.5	0.5	36.93%	38.83%	39.49%	98	94
1.0	1.0	37.56%	39.70%	40.23%	114	104
1.0	0.75	37.33%	39.31%	39.90%	100	94

The α and β values that produced these lowest comparison scores suggest that both the ESPN and NBC Sports rankings consider both game recency and point differential as somewhat significant factors in their rankings.

5.3.3 National Basketball Association

Table 5.26: *NBA 2017-2018 Ranking Comparison Scores for ESPN*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	104	112	108	108	108
0.25	104	110	**80**	86	86
0.5	104	96	**80**	86	88
0.75	104	96	94	86	88
1.0	104	94	94	86	92

Table 5.27: *NBA 2017-2018 Ranking Comparison Scores for NBA.com*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	108	114	100	104	104
0.25	108	114	**86**	**86**	**86**
0.5	108	94	**86**	**86**	90
0.75	100	94	94	**86**	90
1.0	100	94	94	**86**	94

Using the 2017-2018 NBA season, we compared our rankings to those from ESPN [34] and NBA.com [35]. Against these two rankings, we found that our closest rankings gave comparison scores of 80 when compared to ESPN, and 86 when compared to NBA.com, giving respective average team displacements of 2.667 and 2.867 ranks. Similar to our MLB rankings, these are significantly higher than the 28 comparison score and 0.933 average team displacement given by comparing the two external rankings.

When we compared our rankings to those from NBA.com, we found that there were multiple α and β pairs which produced rankings with equivalent comparison scores to the external rankings. We used $\alpha = 0.5$ and $\beta = 0.5$ for the following rankings, since this α and β pair was one that produced the lowest comparison score for both rankings.

Table 5.28: *NBA 2017-2018 Ranking Comparison between ESPN and our Ranking*

$\alpha = 0.5$ $\beta = 0.5$ Backedge Percent = 27.7%

	Our Ranking	ESPN	Rank Difference
1.	Toronto Raptors	Houston Rockets	[+3]
2.	Boston Celtics	Toronto Raptors	[-1]
3.	Philadelphia 76ers	Golden State Warriors	[+8]
4.	Houston Rockets	Philadelphia 76ers	[-1]
5.	Minnesota Timberwolves	Cleveland Cavaliers	[+9]
6.	Denver Nuggets	Boston Celtics	[-4]
7.	Portland Trail Blazers	Portland Trail Blazers	[0]
8.	Oklahoma City Thunder	Utah Jazz	[+2]
9.	Indiana Pacers	Indiana Pacers	[0]
10.	Utah Jazz	New Orleans Pelicans	[+2]
11.	Golden State Warriors	San Antonio Spurs	[+2]
12.	New Orleans Pelicans	Oklahoma City Thunder	[-4]
13.	San Antonio Spurs	Minnesota Timberwolves	[-8]
14.	Cleveland Cavaliers	Denver Nuggets	[-8]
15.	Miami Heat	Miami Heat	[0]
16.	Charlotte Hornets	Milwaukee Bucks	[+4]
17.	Washington Wizards	Washington Wizards	[0]
18.	Los Angeles Clippers	Detroit Pistons	[+1]
19.	Detroit Pistons	Los Angeles Clippers	[-1]
20.	Milwaukee Bucks	Charlotte Hornets	[-4]
21.	Los Angeles Lakers	Los Angeles Lakers	[0]
22.	New York Knicks	Brooklyn Nets	[+3]
23.	Phoenix Suns	Sacramento Kings	[+1]
24.	Sacramento Kings	New York Knicks	[-2]
25.	Brooklyn Nets	Chicago Bulls	[+1]
26.	Chicago Bulls	Dallas Mavericks	[+2]
27.	Atlanta Hawks	Atlanta Hawks	[0]
28.	Dallas Mavericks	Orlando Magic	[+1]
29.	Orlando Magic	Memphis Grizzlies	[+1]
30.	Memphis Grizzlies	Phoenix Suns	[-7]

Table 5.29: *NBA 2017-2018 Ranking Comparison between NBA.com and our Ranking*

$\alpha = 0.5 \quad \beta = 0.5 \quad$ Backedge Percent = 27.7%

	Our Ranking	NBA.com	Rank Difference
1.	Toronto Raptors	Houston Rockets	[+3]
2.	Boston Celtics	Toronto Raptors	[-1]
3.	Philadelphia 76ers	Golden State Warriors	[+8]
4.	Houston Rockets	Utah Jazz	[+6]
5.	Minnesota Timberwolves	Philadelphia 76ers	[-2]
6.	Denver Nuggets	Portland Trail Blazers	[+1]
7.	Portland Trail Blazers	Cleveland Cavaliers	[+7]
8.	Oklahoma City Thunder	Boston Celtics	[-6]
9.	Indiana Pacers	Indiana Pacers	[0]
10.	Utah Jazz	Oklahoma City Thunder	[-2]
11.	Golden State Warriors	New Orleans Pelicans	[+1]
12.	New Orleans Pelicans	San Antonio Spurs	[+1]
13.	San Antonio Spurs	Denver Nuggets	[-7]
14.	Cleveland Cavaliers	Minnesota Timberwolves	[-9]
15.	Miami Heat	Miami Heat	[0]
16.	Charlotte Hornets	Milwaukee Bucks	[+4]
17.	Washington Wizards	Washington Wizards	[0]
18.	Los Angeles Clippers	Los Angeles Clippers	[0]
19.	Detroit Pistons	Detroit Pistons	[0]
20.	Milwaukee Bucks	Los Angeles Lakers	[+1]
21.	Los Angeles Lakers	Charlotte Hornets	[-5]
22.	New York Knicks	Brooklyn Nets	[+3]
23.	Phoenix Suns	Atlanta Hawks	[+4]
24.	Sacramento Kings	Sacramento Kings	[0]
25.	Brooklyn Nets	New York Knicks	[-3]
26.	Chicago Bulls	Dallas Mavericks	[+2]
27.	Atlanta Hawks	Chicago Bulls	[-1]
28.	Dallas Mavericks	Orlando Magic	[+1]
29.	Orlando Magic	Memphis Grizzlies	[+1]
30.	Memphis Grizzlies	Phoenix Suns	[-7]

Looking at backedge weight percentage, we found that for $\alpha = 0.5$ and $\beta = 0.5$ gave us 27.7% backedge weight for our own rankings. For the external rankings, this α and β pair gave a backedge weight percentage of 31.6% for ESPN and 32.2% for the NBA.com ranking. This is significantly higher than the backedge weight our own ranking system produces for this data, meaning our rankings create a lower number of upsets, and that those upsets were closer scoring games that happened earlier in the season compared to those in the ESPN and NBA.com rankings.

Table 5.30: *NBA 2017-2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	ESPN	NBA.com	ESPN	NBA.com
0.0	0.0	24.40%	29.70%	30.31%	104	108
0.5	0.5	27.74%	31.55%	32.23%	80	86
1.0	1.0	29.02%	32.68%	33.41%	92	94

Because of the number of equivalent comparison scores for the NBA.com ranking, it is difficult to determine exactly how much game recency and point differential factor into their rankings. The best we can determine based on these results is that neither is likely to be a significant factor, since the equivalent comparison scores do form a cluster around $\alpha = 0.5$ and $\beta = 0.75$ However, for the ESPN ranking, we can determine that both factors are fairly significant in their rankings, since the lowest score was produced with $\alpha = 0.25$ or $\alpha = 0.5$ and $\beta = 0.5$.

5.3.4 National Hockey League

Table 5.31: *NHL 2017-2018 Ranking Comparison Scores for ESPN*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	130	144	142	136	136
0.25	134	144	144	134	132
0.5	132	140	140	130	**108**
0.75	114	120	122	122	116
1.0	116	122	120	120	124

Table 5.32: *NHL 2017-2018 Ranking Comparison Scores for Sports Illustrated*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	120	132	130	140	140
0.25	122	132	132	138	136
0.5	**116**	126	126	132	130
0.75	**116**	120	124	124	120
1.0	118	122	126	126	134

Using the 2017-2018 NHL season, we compared our rankings to those from ESPN [36] and Sports Illustrated (SI) [37]. We found that our closest rankings gave comparison scores of 108 against the ESPN ranking, and 116 when compared to Sports Illustrated, with respective average team displacements of 3.484 and 3.742 ranks. This was higher

than the 68 comparison score and 2.194 average team displacement between these two external rankings. Notably, the α, β pair that produced the lowest comparison score was significantly different for each of the two external rankings.

Table 5.33: *NHL 2017-2018 Comparison between ESPN and our Ranking*

$\alpha = 0.5 \quad \beta = 1 \quad$ Backedge Percent = 33.5%

Rank	Our Ranking	ESPN	Rank Difference
1.	Vegas Golden Knights	Nashville Predators	[+1]
2.	Nashville Predators	Tampa Bay Lightning	[+2]
3.	Winnipeg Jets	Boston Bruins	[+12]
4.	Tampa Bay Lightning	Vegas Golden Knights	[-3]
5.	Anaheim Ducks	Winnipeg Jets	[-2]
6.	Pittsburgh Penguins	Pittsburgh Penguins	[0]
7.	Los Angeles Kings	Toronto Maple Leafs	[+5]
8.	Philadelphia Flyers	Washington Capitals	[+1]
9.	Washington Capitals	Philadelphia Flyers	[-1]
10.	Columbus Blue Jackets	San Jose Sharks	[+7]
11.	Colorado Avalanche	Los Angeles Kings	[-4]
12.	Toronto Maple Leafs	Columbus Blue Jackets	[-2]
13.	Dallas Stars	Minnesota Wild	[+3]
14.	Florida Panthers	New Jersey Devils	[+7]
15.	Boston Bruins	Colorado Avalanche	[-4]
16.	Minnesota Wild	Dallas Stars	[-3]
17.	San Jose Sharks	St. Louis Blues	[+2]
18.	Calgary Flames	Calgary Flames	[0]
19.	St. Louis Blues	Anaheim Ducks	[-14]
20.	Edmonton Oilers	Florida Panthers	[-6]
21.	New Jersey Devils	New York Islanders	[+2]
22.	Montreal Canadiens	Chicago Blackhawks	[+6]
23.	New York Islanders	Carolina Hurricanes	[+3]
24.	Detroit Red Wings	New York Rangers	[+1]
25.	New York Rangers	Montreal Canadiens	[-3]
26.	Carolina Hurricanes	Edmonton Oilers	[-6]
27.	Vancouver Canucks	Vancouver Canucks	[0]
28.	Chicago Blackhawks	Detroit Red Wings	[-4]
29.	Arizona Coyotes	Buffalo Sabres	[+2]
30.	Ottawa Senators	Arizona Coyotes	[-1]
31.	Buffalo Sabres	Ottawa Senators	[-1]

Table 5.34: *NHL 2017-2018 Comparison between Sports Illustrated and our Ranking*

$\alpha = 0.5 \quad \beta = 0 \quad$ Backedge Percent = 31.4%

Rank	Our Ranking	Sports Illustrated	Rank Difference
1.	Vegas Golden Knights	Nashville Predators	[+1]
2.	Nashville Predators	Boston Bruins	[+8]
3.	Winnipeg Jets	Tampa Bay Lightning	[+1]
4.	Tampa Bay Lightning	Winnipeg Jets	[-1]
5.	Los Angeles Kings	Vegas Golden Knights	[-4]
6.	Washington Capitals	Washington Capitals	[0]
7.	Colorado Avalanche	Toronto Maple Leafs	[+1]
8.	Toronto Maple Leafs	San Jose Sharks	[+8]
9.	Anaheim Ducks	Minnesota Wild	[+5]
10.	Boston Bruins	Columbus Blue Jackets	[+2]
11.	Pittsburgh Penguins	Pittsburgh Penguins	[0]
12.	Columbus Blue Jackets	Los Angeles Kings	[-7]
13.	Dallas Stars	Philadelphia Flyers	[+4]
14.	Minnesota Wild	St. Louis Blues	[+5]
15.	Florida Panthers	Anaheim Ducks	[-6]
16.	San Jose Sharks	Colorado Avalanche	[-9]
17.	Philadelphia Flyers	New Jersey Devils	[+4]
18.	Arizona Coyotes	Florida Panthers	[-3]
19.	St. Louis Blues	Dallas Stars	[-6]
20.	Edmonton Oilers	Calgary Flames	[+6]
21.	New Jersey Devils	Arizona Coyotes	[-3]
22.	Montreal Canadiens	Carolina Hurricanes	[+5]
23.	Detroit Red Wings	New York Rangers	[+2]
24.	New York Islanders	New York Islanders	[0]
25.	New York Rangers	Chicago Blackhawks	[+5]
26.	Calgary Flames	Edmonton Oilers	[-6]
27.	Carolina Hurricanes	Detroit Red Wings	[-4]
28.	Buffalo Sabres	Montreal Canadiens	[-6]
29.	Vancouver Canucks	Vancouver Canucks	[0]
30.	Chicago Blackhawks	Ottawa Senators	[+1]
31.	Ottawa Senators	Buffalo Sabres	[-3]

For the ESPN ranking, setting $\alpha = 0.5$ and $\beta = 0.0$ gives us a backedge weight percentage of 37.84%, while our own ranking with this α and β has a backedge weight percentage of 33.5%. Against the Sports Illustrated ranking, we find that using the $\alpha = 0.5$ and $\beta = 1.0$ which produced the lowest comparison score gives the Sports Illustrated ranking a backedge weight percentage of 38.12%, while our own ranking has a backedge weight percentage of 31.40%.

Table 5.35: *NHL 2017-2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	ESPN	SI	ESPN	SI
0.0	0.0	29.52%	36.90%	38.12%	130	120
0.5	0.5	32.81%	37.47%	38.67%	140	126
1.0	1.0	33.99%	37.84%	39.02%	124	134
0.5	1.0	33.54%	37.84%	39.02%	108	130
0.5	0.0	31.40%	36.90%	38.12%	132	116

For these external rankings, we found that $\alpha = 0.5$ produced the lowest comparison scores, meaning that both consider game recency as a somewhat significant factor. However, their β values were extremely different, with ESPN being most closely matched with $\beta = 1.0$, while Sports Illustrated's ranking was best matched by $\beta = 0.0$. This means that while ESPN likely does not consider point differential at all for NHL rankings, Sports Illustrated uses it as a highly significant part of their ranking. This large difference may be explained by the low scoring nature of NHL games, since our cutoff for the 75^{th} percentile when weighing point differential was only 3 goals, much lower than any other sport.

5.3.5 NCAA Football

Table 5.36: *NCAA Football 2018 Ranking Comparison Scores for the AP Poll*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	461	293	209	172	231
0.25	205	260	278	305	196
0.5	287	243	287	294	355
0.75	292	295	262	302	218
1.0	259	257	**135**	214	341

Table 5.37: *NCAA Football 2018 Ranking Comparison Scores for the College Football Playoff (CFP)*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	436	305	197	175	299
0.25	210	265	241	311	198
0.5	312	192	274	305	355
0.75	340	267	259	301	211
1.0	275	225	**140**	219	339

For both the AP [38] and CFP [39] rankings, $\alpha = 1.0$ and $\beta = 0.5$ produced the closest ranking. With a lowest comparison score of 135, our ranking placed teams approximately 5.40 ranks away from where the AP Poll had placed them. For the CFP comparison, our lowest comparison score was 140, placing teams 5.60 ranks away from where they were ranked by the CFP system. As a baseline, the AP and CFP rankings had a comparison score of 24, placing teams an average of 0.96 ranks away from the other. However, since not every team is ranked in either of these external rankings, teams in one of the external rankings but not the other (Army, Iowa State, Northwestern, and Syracuse) are not counted in this baseline comparison score.

Table 5.38: *NCAA Football 2018 Comparison between the AP Poll and our Ranking*

$\alpha = 1 \quad \beta = 0.5 \quad$ Backedge Percent = 10.4%

Rank	Our Ranking	AP Poll	Rank Difference
1.	Alabama	Alabama	[0]
2.	Clemson	Clemson	[0]
3.	Notre Dame	Notre Dame	[0]
4.	Oklahoma	Oklahoma	[0]
5.	Central Florida	Ohio State	[+1]
6.	Ohio State	Georgia	[+3]
7.	Michigan	Central Florida	[-2]
8.	Louisiana State	Michigan	[-1]
9.	Georgia	Washington	[+7]
10.	Kentucky	Florida	[+10]
11.	West Virginia	Louisiana State	[-3]
12.	Boise State	Washington State	[+5]
13.	Fresno State	Penn State	[+1]
14.	Penn State	Texas	[+1]
15.	Texas	West Virginia	[-4]
16.	Washington	Kentucky	[-6]
17.	Washington State	Syracuse	[+19]
18.	South Carolina	Mississippi State	[+3]
19.	Missouri	Fresno State	[-6]
20.	Florida	Utah	[+5]
21.	Mississippi State	Texas A&M	[+1]
22.	Texas A&M	Army	[+10]
23.	Vanderbilt	Boise State	[-11]
24.	North Carolina State	Missouri	[-5]
25.	Utah	Iowa State	[+31]

Table 5.39: *NCAA Football 2018 Comparison between the CFP and our Ranking*

$\alpha = 1 \quad \beta = 0.5 \quad$ Backedge Percent = 10.4%

Rank	Our Ranking	College Football Playoff	Rank Difference
1.	Alabama	Alabama	[0]
2.	Clemson	Clemson	[0]
3.	Notre Dame	Notre Dame	[0]
4.	Oklahoma	Oklahoma	[0]
5.	Central Florida	Georgia	[+4]
6.	Ohio State	Ohio State	[0]
7.	Michigan	Michigan	[0]
8.	Louisiana State	Central Florida	[-3]
9.	Georgia	Washington	[+7]
10.	Kentucky	Florida	[+10]
11.	West Virginia	Louisiana State	[-3]
12.	Boise State	Penn State	[+2]
13.	Fresno State	Washington State	[+4]
14.	Penn State	Kentucky	[-4]
15.	Texas	Texas	[0]
16.	Washington	West Virginia	[-5]
17.	Washington State	Utah	[+8]
18.	South Carolina	Mississippi State	[+3]
19.	Missouri	Texas A&M	[+3]
20.	Florida	Syracuse	[+16]
21.	Mississippi State	Fresno State	[-8]
22.	Texas A&M	Northwestern	[+11]
23.	Vanderbilt	Missouri	[-4]
24.	North Carolina State	Iowa State	[+32]
25.	Utah	Boise State	[-13]

Looking at backedge weight, the external rankings give us a backedge weight of 20.99% from the AP Poll with $\alpha = 1.0$ and $\beta = 0.5$, while our own ranking gives 10.38%. For the CFP system, this $\alpha = 1.0$ and $\beta = 0.5$ gives a backedge weight percentage of 19.97%.

Table 5.40: *NCAA Football 2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	AP Poll	CFP	AP Poll	CFP
0.0	0.0	6.33%	15.69%	14.43%	461	436
0.5	0.5	10.10%	20.68%	19.49%	287	274
1.0	1.0	11.34%	22.50%	21.43%	341	339
1.0	0.5	10.38%	20.99%	19.97%	135	140

For both of these rankings, the lowest comparison score happened at $\alpha = 1.0$ and $\beta = 0.5$. This suggests that the recency of a particular game has no effect on its weight in these external rankings, while the point differential of the game is a somewhat significant factor.

5.3.6 NCAA Basketball

Table 5.41: *NCAA Basketball 2017-2018 Ranking Comparison Scores for the AP Poll*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	640	611	780	489	566
0.25	402	750	537	588	430
0.5	**346**	518	596	590	402
0.75	378	421	415	440	705
1.0	347	377	669	412	383

Table 5.42: *NCAA Basketball 2017-2018 Ranking Comparison Scores for the Coaches Poll*

$\alpha \backslash \beta$	0.0	0.25	0.5	0.75	1.0
0.0	630	560	576	494	566
0.25	392	745	540	594	457
0.5	**351**	537	564	471	417
0.75	391	403	441	496	714
1.0	391	382	692	417	379

For both the AP Poll [40] and Coaches Poll [41], $\alpha = 0.5$ and $\beta = 0.0$ produced the closest ranking. With a lowest comparison score of 346, our ranking placed teams approximately 13.84 ranks away from where the AP Poll had placed them. Comparing ourselves to the Coaches Poll, our lowest comparison score was 351, placing teams an average of 14.04 ranks away. As a baseline, the AP Poll and Coaches Poll rankings had a comparison score of 44, placing teams an average of 1.76 ranks away from the other. Like the NCAA Football rankings, however, teams in one of the external rankings but not the other are not counted in this baseline comparison score.

Looking at backedge weight, the AP Poll give us a backedge weight of 32.91% with $\alpha = 0.5$ and $\beta = 0.0$, while our own ranking gives 15.49%. For the Coaches Poll, $\alpha = 1.0$ and $\beta = 0.5$ gives a backedge weight percentage of 37.28%.

Table 5.43: *NCAA Basketball 2017-2018 Comparison between the AP Poll and our Ranking*

$\alpha = 0.5 \quad \beta = 0.0 \quad$ Backedge Percent = 15.5%

Rank	Our Ranking	AP Poll	Rank Difference
1.	Villanova	Virginia	[+1]
2.	Virginia	Villanova	[-1]
3.	Duke	Xavier	[+49]
4.	UNC	Michigan State	[+28]
5.	Texas Tech	Duke	[-2]
6.	Kansas	Gonzaga	[+9]
7.	TCU	Michigan	[+23]
8.	Nevada	Cincinnati	[+45]
9.	South Dakota	Kansas	[-3]
10.	Syracuse	Purdue	[+23]
11.	Clemson	Wichita State	[+9]
12.	Florida	UNC	[-8]
13.	Auburn	Tennessee	[+1]
14.	Tennessee	Texas Tech	[-9]
15.	Gonzaga	Arizona	[+25]
16.	Murray State	Auburn	[-3]
17.	Saint Mary's	Ohio State	[+14]
18.	Miami (FL)	West Virginia	[+18]
19.	Notre Dame	Clemson	[-8]
20.	Wichita State	Saint Mary's	[-3]
21.	Fresno State	Houston	[+33]
22.	Jacksonville State	Nevada	[-14]
23.	Belmont	Florida	[-11]
24.	St. Joseph's	Miami (FL)	[-6]
25.	Rhode Island	Rhode Island	[0]

Table 5.45: *NCAA Basketball 2017-2018 Backedge Weight Comparisons*

		Backedge Percentage			Comparison Score	
α	β	Our Rankings	AP Poll	Coaches Poll	AP Poll	Coaches Poll
0.0	0.0	16.34%	32.19%	37.82%	640	630
0.5	0.5	20.26%	38.06%	42.06%	596	574
1.0	1.0	21.45%	40.58%	44.00%	383	379
0.5	0.0	15.49%	32.91%	37.28%	346	351

From the closest scores to our external rankings, we find that these rankings are most closely matched when $\alpha = 1.0$ and $\beta = 0.5$. This means that the AP Poll and Coaches Poll see game recency as a somewhat significant factor in their rankings. Additionally,

Table 5.44: *NCAA Basketball 2017-2018 Comparison between the Coaches Poll and our Ranking*

$\alpha = 0.5 \quad \beta = 0.0 \quad$ Backedge Percent = 15.5%

Rank	Our Ranking	Coaches Poll	Rank Difference
1.	Villanova	Virginia	[+1]
2.	Virginia	Villanova	[-1]
3.	Duke	Kansas	[+3]
4.	UNC	Xavier	[+48]
5.	Texas Tech	Michigan State	[+27]
6.	Kansas	Duke	[-3]
7.	TCU	Michigan	[+23]
8.	Nevada	Gonzaga	[+7]
9.	South Dakota	UNC	[-5]
10.	Syracuse	Cincinnati	[+43]
11.	Clemson	Purdue	[+22]
12.	Florida	Tennessee	[+2]
13.	Auburn	Texas Tech	[-8]
14.	Tennessee	West Virginia	[+22]
15.	Gonzaga	Arizona	[+25]
16.	Murray State	Wichita State	[+4]
17.	Saint Mary's	Ohio State	[+14]
18.	Miami (FL)	Clemson	[-7]
19.	Notre Dame	Houston	[+35]
20.	Wichita State	Kentucky	[+15]
21.	Fresno State	Auburn	[-8]
22.	Jacksonville State	Rhode Island	[+3]
23.	Belmont	Saint Mary's	[-6]
24.	St. Joseph's	Florida	[-12]
25.	Rhode Island	Miami (FL)	[-7]

this means that point differential carries very significant weight in their rankings.

5.3.7 Rankings Summary

In the end, our algorithms generated a “best" ranking for each data set. Based upon our results, the “best" rankings come from various combinations of our ordering algorithms. For professional sports, rankings with the lowest total backedge weight were generated by using the *Dynamic Programming Brute Force* algorithm, which was possible due to the relatively low number of teams in those leagues. For NCAA sports, we used *Sliding Window* with window size 20 since the NCAA leagues had too many teams to run *Dynamic Programming Brute Force* in a reasonable amount of time. For all data sets, we ran the *RoC Reorder* postprocess to choose the best equivalent ranking based on our second metric. Finally, α and β were chosen so that our rankings best match external rankings.

5.3.7.1 NFL 2018

Below is our best generated ranking for the NFL 2018 season where $\alpha = 0.5$ and $\beta = 0.25$, which was closest to the existing NFL.com ranking.

Table 5.46: *NFL 2018 Best Ranking*

$\alpha = 0.5$ $\beta = 0.25$ Backedge Percent = 15.8%

Rank	Team	Range of Correctness
1.	New Orleans Saints	[1, 3]
2.	New England Patriots	[1, 2]
3.	Chicago Bears	[3, 3]
4.	Los Angeles Rams	[4, 6]
5.	Houston Texans	[3, 10]
6.	Baltimore Ravens	[2, 6]
7.	Los Angeles Chargers	[7, 7]
8.	Seattle Seahawks	[8, 8]
9.	Kansas City Chiefs	[9, 9]
10.	Pittsburgh Steelers	[10, 10]
11.	Cleveland Browns	[11, 11]
12.	Cincinnati Bengals	[12, 12]
13.	Indianapolis Colts	[13, 13]
14.	Buffalo Bills	[14, 14]
15.	Tennessee Titans	[15, 16]
16.	Minnesota Vikings	[15, 17]
17.	Dallas Cowboys	[16, 17]
18.	Philadelphia Eagles	[18, 20]
19.	Detroit Lions	[18, 19]
20.	Green Bay Packers	[20, 20]
21.	Atlanta Falcons	[21, 21]
22.	Washington Redskins	[22, 22]
23.	Carolina Panthers	[23, 24]
24.	Jacksonville Jaguars	[23, 24]
25.	New York Giants	[25, 25]
26.	Tampa Bay Buccaneers	[26, 28]
27.	Miami Dolphins	[25, 27]
28.	New York Jets	[28, 30]
29.	San Francisco 49ers	[27, 29]
30.	Oakland Raiders	[30, 30]
31.	Denver Broncos	[31, 31]
32.	Arizona Cardinals	[32, 32]

5.3.7.2 MLB 2018

Below is our best generated ranking for the MLB 2018 season where $\alpha = 0.5$ and $\beta = 0.5$, which was closest to the existing NBC Sports ranking.

Table 5.47: *MLB 2018 Best Ranking*

$\alpha = 0.5$ $\beta = 0.5$ Backedge Percent = 36.9%

Rank	Team	Range of Correctness
1.	New York Yankees	[1, 1]
2.	Houston Astros	[2, 2]
3.	Boston Red Sox	[3, 8]
4.	Chicago Cubs	[1, 4]
5.	Los Angeles Dodgers	[5, 5]
6.	Milwaukee Brewers	[6, 6]
7.	St. Louis Cardinals	[7, 7]
8.	Colorado Rockies	[8, 8]
9.	Seattle Mariners	[9, 10]
10.	Atlanta Braves	[9, 10]
11.	Tampa Bay Rays	[11, 11]
12.	Oakland Athletics	[12, 12]
13.	Cleveland Indians	[13, 16]
14.	New York Mets	[12, 14]
15.	San Francisco Giants	[15, 15]
16.	Arizona Diamondbacks	[16, 16]
17.	Los Angeles Angels	[17, 17]
18.	Texas Rangers	[18, 18]
19.	Minnesota Twins	[19, 19]
20.	Toronto Blue Jays	[20, 20]
21.	Washington Nationals	[21, 21]
22.	Philadelphia Phillies	[22, 22]
23.	San Diego Padres	[23, 23]
24.	Pittsburgh Pirates	[24, 24]
25.	Miami Marlins	[25, 27]
26.	Kansas City Royals	[25, 26]
27.	Detroit Tigers	[27, 27]
28.	Cincinnati Reds	[28, 28]
29.	Chicago White Sox	[29, 29]
30.	Baltimore Orioles	[30, 30]

5.3.7.3 NBA 2017-2018

Below is our best generated ranking for the NBA 2017-2018 season where $\alpha = 0.5$ and $\beta = 0.5$, which was closest to the existing ESPN ranking.

Table 5.48: *NBA 2017-2018 Best Ranking*

$\alpha = 0.5$ $\beta = 0.5$ Backedge Percent = 27.7%

Rank	Team	Range of Correctness
1.	Toronto Raptors	[1, 1]
2.	Boston Celtics	[2, 2]
3.	Philadelphia 76ers	[3, 3]
4.	Houston Rockets	[4, 4]
5.	Minnesota Timberwolves	[5, 5]
6.	Denver Nuggets	[6, 6]
7.	Portland Trail Blazers	[7, 7]
8.	Oklahoma City Thunder	[8, 8]
9.	Indiana Pacers	[9, 9]
10.	Utah Jazz	[10, 10]
11.	Golden State Warriors	[11, 11]
12.	New Orleans Pelicans	[12, 12]
13.	San Antonio Spurs	[13, 13]
14.	Cleveland Cavaliers	[14, 14]
15.	Miami Heat	[15, 15]
16.	Charlotte Hornets	[16, 16]
17.	Washington Wizards	[17, 17]
18.	Los Angeles Clippers	[18, 18]
19.	Detroit Pistons	[19, 19]
20.	Milwaukee Bucks	[20, 20]
21.	Los Angeles Lakers	[21, 21]
22.	New York Knicks	[22, 22]
23.	Phoenix Suns	[23, 23]
24.	Sacramento Kings	[24, 24]
25.	Brooklyn Nets	[25, 25]
26.	Chicago Bulls	[26, 26]
27.	Atlanta Hawks	[27, 27]
28.	Dallas Mavericks	[28, 28]
29.	Orlando Magic	[29, 29]
30.	Memphis Grizzlies	[30, 30]

5.3.7.4 NHL 2017-2018

Below is our best generated ranking for the NHL 2017-2018 season where $\alpha = 0.5$ and $\beta = 1.0$, which was closest to the existing ESPN ranking.

Table 5.49: *NHL 2017-2018 Best Ranking*

$\alpha = 0.5 \quad \beta = 1 \quad$ Backedge Percent = 33.5%

Rank	Team	Range of Correctness
1.	Vegas Golden Knights	[1, 1]
2.	Nashville Predators	[2, 2]
3.	Winnipeg Jets	[3, 3]
4.	Tampa Bay Lightning	[4, 4]
5.	Anaheim Ducks	[5, 5]
6.	Pittsburgh Penguins	[6, 6]
7.	Los Angeles Kings	[7, 7]
8.	Philadelphia Flyers	[8, 8]
9.	Washington Capitals	[9, 9]
10.	Columbus Blue Jackets	[10, 10]
11.	Colorado Avalanche	[11, 11]
12.	Toronto Maple Leafs	[12, 12]
13.	Dallas Stars	[13, 13]
14.	Florida Panthers	[14, 14]
15.	Boston Bruins	[15, 15]
16.	Minnesota Wild	[16, 16]
17.	San Jose Sharks	[17, 17]
18.	Calgary Flames	[18, 18]
19.	St. Louis Blues	[19, 19]
20.	Edmonton Oilers	[20, 20]
21.	New Jersey Devils	[21, 21]
22.	Montreal Canadiens	[22, 22]
23.	New York Islanders	[23, 23]
24.	Detroit Red Wings	[24, 24]
25.	New York Rangers	[25, 25]
26.	Carolina Hurricanes	[26, 26]
27.	Vancouver Canucks	[27, 27]
28.	Chicago Blackhawks	[28, 28]
29.	Arizona Coyotes	[29, 29]
30.	Ottawa Senators	[30, 30]
31.	Buffalo Sabres	[31, 31]

5.3.7.5 NCAA Football 2018

Below is our best generated ranking for the NCAA Football 2018 season where $\alpha = 1$ and $\beta = 0.5$, which was closest to the existing College Football Playoff and AP Poll top 25 ranking.

Table 5.50: *NCAA Football 2018 Best Ranking*

$\alpha = 1 \quad \beta = 0.5 \quad$ Backedge Percent = 10.4%

Rank	Team	RoC	Rank	Team	RoC	Rank	Team	RoC
1.	Alabama	[1, 7]	45.	Temple	[45, 45]	89.	Florida International	[88, 105]
2.	Clemson	[1, 17]	46.	Cincinnati	[46, 50]	90.	Miami (OH)	[87, 90]
3.	Notre Dame	[1, 6]	47.	San Diego State	[28, 52]	91.	Northern Illinois	[91, 91]
4.	Oklahoma	[1, 10]	48.	Mississippi	[40, 59]	92.	Brigham Young	[92, 92]
5.	Central Florida	[1, 34]	49.	Marshall	[42, 81]	93.	Hawaii	[93, 99]
6.	Ohio State	[1, 6]	50.	Wyoming	[27, 83]	94.	Ohio	[92, 105]
7.	Michigan	[7, 13]	51.	South Florida	[47, 51]	95.	Arizona	[93, 95]
8.	Louisiana State	[2, 8]	52.	Georgia Tech	[52, 61]	96.	Oregon	[96, 96]
9.	Georgia	[9, 9]	53.	Eastern Michigan	[48, 53]	97.	UCLA	[97, 97]
10.	Kentucky	[10, 17]	54.	Purdue	[54, 54]	98.	California	[98, 98]
11.	West Virginia	[5, 14]	55.	Iowa	[55, 55]	99.	Southern California	[99, 109]
12.	Boise State	[1, 12]	56.	Iowa State	[56, 57]	100.	Colorado State	[94, 100]
13.	Fresno State	[13, 46]	57.	Boston College	[55, 61]	101.	Arkansas	[101, 104]
14.	Penn State	[8, 29]	58.	Baylor	[57, 58]	102.	North Carolina	[99, 130]
15.	Texas	[12, 36]	59.	Kansas State	[59, 59]	103.	Navy	[94, 104]
16.	Washington	[1, 16]	60.	Texas Tech	[60, 60]	104.	Louisville	[89, 110]
17.	Washington State	[17, 24]	61.	Oklahoma State	[61, 79]	105.	Tulsa	[104, 119]
18.	South Carolina	[11, 18]	62.	Virginia Tech	[58, 62]	106.	Massachusetts	[95, 106]
19.	Missouri	[19, 19]	63.	Florida State	[63, 63]	107.	Liberty	[107, 107]
20.	Florida	[20, 20]	64.	Wake Forest	[64, 73]	108.	New Mexico	[108, 109]
21.	Mississippi State	[21, 21]	65.	Toledo	[54, 65]	109.	San Jose State	[101, 109]
22.	Texas A&M	[22, 47]	66.	Nevada	[66, 83]	110.	Nevada-Las Vegas	[110, 127]
23.	Vanderbilt	[21, 37]	67.	Coastal Carolina	[44, 67]	111.	Western Kentucky	[105, 116]
24.	North Carolina State	[3, 48]	68.	Alabama-Birmingham	[68, 69]	112.	Georgia State	[70, 112]
25.	Utah	[18, 26]	69.	Louisiana	[68, 111]	113.	Louisiana-Monroe	[113, 113]
26.	Utah State	[13, 49]	70.	Louisiana Tech	[69, 70]	114.	Southern Mississippi	[114, 120]
27.	Stanford	[26, 46]	71.	North Texas	[71, 71]	115.	South Alabama	[114, 121]
28.	Troy	[13, 28]	72.	Southern Methodist	[72, 72]	116.	Rutgers	[88, 121]
29.	Georgia Southern	[29, 29]	73.	Houston	[73, 73]	117.	Ball State	[112, 117]
30.	Appalachian State	[30, 42]	74.	Tulane	[74, 80]	118.	Western Michigan	[118, 123]
31.	Duke	[3, 31]	75.	Colorado	[26, 75]	119.	Oregon State	[100, 130]
32.	Army	[32, 43]	76.	Arizona State	[76, 77]	120.	Connecticut	[107, 130]
33.	Northwestern	[32, 39]	77.	Nebraska	[76, 77]	121.	Texas-San Antonio	[115, 121]
34.	Miami (FL)	[32, 34]	78.	Michigan State	[78, 78]	122.	Texas State	[122, 122]
35.	Pittsburgh	[35, 35]	79.	Maryland	[79, 84]	123.	New Mexico State	[123, 127]
36.	Syracuse	[36, 56]	80.	Kansas	[62, 115]	124.	Bowling Green State	[119, 124]
37.	Texas Christian	[16, 55]	81.	East Carolina	[75, 101]	125.	Akron	[125, 125]
38.	Tennessee	[24, 38]	82.	Charlotte	[69, 82]	126.	Kent State	[126, 130]
39.	Auburn	[39, 47]	83.	Florida Atlantic	[83, 83]	127.	Central Michigan	[126, 130]
40.	Wisconsin	[34, 53]	84.	Air Force	[84, 99]	128.	Texas-El Paso	[124, 128]
41.	Middle Tennessee State	[24, 48]	85.	Illinois	[80, 85]	129.	Rice	[129, 129]
42.	Memphis	[20, 71]	86.	Minnesota	[86, 86]	130.	Old Dominion	[130, 130]
43.	Arkansas State	[31, 66]	87.	Indiana	[87, 87]			
44.	Buffalo	[33, 44]	88.	Virginia	[88, 93]			

5.3.7.6 NCAA Basketball 2017-2018

Below is our best generated ranking for the NCAA Basketball 2017-2018 season where $\alpha = 0.5$ and $\beta = 0$, which was closest to the existing AP Poll and Coaches Poll.

Table 5.51: *NCAA Basketball 2017-2018 Best Ranking*

$\alpha = 0.5 \quad \beta = 0 \quad$ Backedge Percent = 15.5%

Rank	Team	RoC	Rank	Team	RoC	Rank	Team	RoC
1.	Villanova	[1, 13]	54.	Houston	[54, 63]	107.	Wofford	[104, 107]
2.	Virginia	[1, 2]	55.	San Diego State	[52, 60]	108.	Georgia Tech	[108, 109]
3.	Duke	[3, 3]	56.	Kansas State	[52, 56]	109.	Iowa	[103, 109]
4.	UNC	[4, 9]	57.	Baylor	[57, 59]	110.	Northwestern	[110, 110]
5.	Texas Tech	[1, 5]	58.	Seton Hall	[53, 72]	111.	Valparaiso	[111, 111]
6.	Kansas	[6, 6]	59.	Mississippi State	[54, 62]	112.	Utah State	[112, 112]
7.	TCU	[7, 7]	60.	Texas	[58, 64]	113.	Northeastern	[113, 114]
8.	Nevada	[8, 20]	61.	Boise State	[56, 61]	114.	UNLV	[113, 141]
9.	South Dakota	[8, 84]	62.	Loyola (IL)	[62, 70]	115.	Hofstra	[114, 115]
10.	Syracuse	[7, 10]	63.	Missouri	[60, 63]	116.	William & Mary	[116, 142]
11.	Clemson	[11, 11]	64.	Arkansas	[64, 64]	117.	DePaul	[111, 117]
12.	Florida	[12, 12]	65.	Oklahoma	[65, 65]	118.	Providence	[118, 118]
13.	Auburn	[13, 13]	66.	Oklahoma State	[66, 67]	119.	Washington	[119, 155]
14.	Tennessee	[14, 28]	67.	Georgia	[65, 72]	120.	St. John's (NY)	[119, 120]
15.	Gonzaga	[13, 16]	68.	Iowa State	[67, 71]	121.	Georgetown	[121, 122]
16.	Murray State	[14, 21]	69.	Wake Forest	[55, 69]	122.	Cal State Fullerton	[76, 147]
17.	Saint Mary's	[16, 45]	70.	Florida State	[70, 107]	123.	Richmond	[122, 123]
18.	Miami (FL)	[12, 18]	71.	Illinois State	[63, 71]	124.	Davidson	[124, 125]
19.	Notre Dame	[19, 19]	72.	Tulsa	[72, 131]	125.	Alabama-Birmingham	[124, 135]
20.	Wichita State	[20, 25]	73.	Vanderbilt	[68, 73]	126.	VCU	[125, 126]
21.	Fresno State	[9, 54]	74.	Radford	[74, 76]	127.	Bucknell	[127, 127]
22.	Jacksonville State	[17, 22]	75.	UCSB	[44, 121]	128.	Vermont	[128, 147]
23.	Belmont	[23, 43]	76.	Sam Houston State	[58, 158]	129.	Ball State	[128, 143]
24.	St. Joseph's	[2, 24]	77.	UNC Asheville	[75, 77]	130.	Pitt	[109, 171]
25.	Rhode Island	[25, 25]	78.	UNC Greensboro	[78, 78]	131.	La Salle	[128, 131]
26.	College of Charleston	[26, 75]	79.	ETSU	[79, 97]	132.	Temple	[132, 132]
27.	Lamar	[1, 27]	80.	Dayton	[60, 80]	133.	South Carolina	[133, 168]
28.	Stephen F. Austin	[28, 28]	81.	St. Bonaventure	[81, 85]	134.	UCF	[133, 135]
29.	LSU	[29, 29]	82.	Tennessee Tech	[81, 82]	135.	Wisconsin	[133, 137]
30.	Michigan	[30, 30]	83.	New Mexico	[83, 83]	136.	Memphis	[135, 136]
31.	Ohio State	[31, 31]	84.	Wyoming	[84, 84]	137.	Mercer	[137, 152]
32.	Michigan State	[32, 32]	85.	South Dakota State	[85, 85]	138.	Minnesota	[136, 138]
33.	Purdue	[33, 36]	86.	Buffalo	[86, 95]	139.	Drake	[139, 139]
34.	Texas A&M	[30, 34]	87.	Lipscomb	[83, 94]	140.	Missouri State	[140, 140]
35.	Kentucky	[35, 35]	88.	Evansville	[84, 88]	141.	Wright State	[141, 143]
36.	West Virginia	[36, 55]	89.	Southern Illinois	[89, 92]	142.	Illinois	[139, 142]
37.	Louisville	[36, 37]	90.	Ole Miss	[86, 90]	143.	Marshall	[143, 143]
38.	Virginia Tech	[38, 38]	91.	Alabama	[91, 91]	144.	Toledo	[144, 159]
39.	NC State	[39, 39]	92.	BYU	[92, 105]	145.	Austin Peay	[143, 202]
40.	Arizona	[40, 41]	93.	Bradley	[91, 93]	146.	Grand Canyon	[143, 184]
41.	Nebraska	[34, 47]	94.	Northern Iowa	[94, 110]	147.	UTSA	[144, 156]
42.	UCLA	[41, 42]	95.	Tennessee State	[88, 95]	148.	Harvard	[139, 219]
43.	USC	[43, 43]	96.	Canisius	[96, 163]	149.	North Dakota State	[141, 168]
44.	Middle Tennessee	[44, 44]	97.	Central Michigan	[87, 97]	150.	Rutgers	[143, 199]
45.	Western Kentucky	[45, 46]	98.	Purdue-Fort Wayne	[98, 98]	151.	UConn	[137, 151]
46.	New Mexico State	[44, 82]	99.	Indiana	[99, 99]	152.	SMU	[152, 157]
47.	Old Dominion	[46, 79]	100.	Maryland	[100, 100]	153.	Liberty	[138, 153]
48.	Boston College	[42, 68]	101.	Butler	[101, 102]	154.	Georgia State	[154, 154]
49.	Stanford	[44, 49]	102.	Penn State	[101, 108]	155.	Louisiana	[155, 156]
50.	Oregon	[50, 50]	103.	Furman	[102, 106]	156.	Montana	[155, 174]
51.	Arizona State	[51, 51]	104.	Marquette	[102, 104]	157.	Louisiana Tech	[156, 168]
52.	Xavier	[52, 52]	105.	Creighton	[105, 109]	158.	Tulane	[155, 158]
53.	Cincinnati	[53, 53]	106.	Utah	[102, 111]	159.	Southeastern Louisiana	[159, 176]

Rank	Team	RoC	Rank	Team	RoC	Rank	Team	RoC
160.	Oakland	[145, 160]	224.	Navy	[224, 225]	288.	Northwestern State	[281, 288]
161.	Northern Kentucky	[161, 164]	225.	NJIT	[224, 239]	289.	Louisiana-Monroe	[289, 289]
162.	Texas-Arlington	[156, 162]	226.	Delaware	[225, 226]	290.	South Alabama	[290, 290]
163.	Georgia Southern	[163, 180]	227.	UNC Wilmington	[227, 227]	291.	Coastal Carolina	[291, 304]
164.	Monmouth	[152, 164]	228.	Campbell	[228, 251]	292.	St. Peter's	[291, 292]
165.	Iona	[165, 165]	229.	Boston University	[225, 233]	293.	Niagara	[293, 293]
166.	Rider	[166, 195]	230.	Grambling	[212, 235]	294.	Cornell	[294, 300]
167.	Albany (NY)	[165, 167]	231.	Hartford	[218, 231]	295.	Incarnate Word	[289, 351]
168.	Colgate	[168, 216]	232.	Stony Brook	[232, 232]	296.	Sacred Heart	[293, 296]
169.	UTEP	[158, 185]	233.	Saint Francis (PA)	[233, 233]	297.	St. Francis (NY)	[297, 297]
170.	Colorado State	[159, 170]	234.	Lehigh	[234, 234]	298.	Fairleigh Dickinson	[298, 298]
171.	Winthrop	[171, 227]	235.	Princeton	[235, 255]	299.	Central Connecticut	[299, 299]
172.	Duquesne	[131, 172]	236.	Texas-Rio Grande Valley	[231, 236]	300.	North Carolina A&T	[300, 306]
173.	San Francisco	[173, 173]	237.	UC-Irvine	[237, 268]	301.	Brown	[300, 301]
174.	UC-Davis	[174, 174]	238.	North Texas	[237, 238]	302.	UMass-Lowell	[302, 314]
175.	Northern Colorado	[175, 175]	239.	Indiana State	[239, 272]	303.	Quinnipiac	[302, 313]
176.	Idaho	[176, 176]	240.	Jacksonville	[226, 240]	304.	Cleveland State	[300, 304]
177.	Nicholls State	[177, 182]	241.	North Florida	[241, 241]	305.	Arkansas State	[305, 305]
178.	Denver	[176, 181]	242.	Eastern Michigan	[242, 242]	306.	Little Rock	[306, 306]
179.	Western Michigan	[177, 179]	243.	Miami (OH)	[243, 243]	307.	Norfolk State	[307, 307]
180.	Ohio	[180, 192]	244.	Kent State	[244, 244]	308.	Bethune-Cookman	[308, 308]
181.	George Mason	[173, 201]	245.	Oregon State	[245, 245]	309.	Savannah State	[309, 339]
182.	Oral Roberts	[179, 186]	246.	Colorado	[246, 247]	310.	Detroit	[305, 310]
183.	New Orleans	[178, 190]	247.	Western Illinois	[244, 260]	311.	Saint Louis	[311, 311]
184.	Hawaii	[175, 184]	248.	Washington State	[247, 248]	312.	George Washington	[312, 312]
185.	Utah Valley	[185, 209]	249.	Texas Southern	[249, 249]	313.	Fordham	[313, 313]
186.	Southern Miss	[176, 198]	250.	Prairie View	[250, 250]	314.	UMass	[314, 338]
187.	Omaha	[183, 246]	251.	Eastern Kentucky	[251, 251]	315.	New Hampshire	[313, 315]
188.	Texas State	[171, 188]	252.	Charleston Southern	[252, 252]	316.	Binghamton	[316, 317]
189.	Pacific	[189, 189]	253.	Gardner-Webb	[253, 253]	317.	Dartmouth	[316, 317]
190.	San Diego	[190, 230]	254.	Hampton	[254, 257]	318.	Loyola (MD)	[318, 351]
191.	Texas A&M-Corpus Christi	[189, 191]	255.	California	[249, 332]	319.	Marist	[304, 319]
192.	Abilene Christian	[192, 192]	256.	Akron	[245, 256]	320.	Citadel	[320, 320]
193.	Bowling Green State	[193, 222]	257.	UT-Martin	[257, 304]	321.	Chattanooga	[321, 321]
194.	Western Carolina	[163, 197]	258.	Appalachian State	[257, 258]	322.	Charlotte	[322, 322]
195.	LIU-Brooklyn	[1, 195]	259.	Troy	[259, 259]	323.	High Point	[323, 324]
196.	Wagner	[196, 196]	260.	UIC	[260, 260]	324.	Siena	[320, 345]
197.	Mount St. Mary's	[197, 232]	261.	Milwaukee	[261, 261]	325.	Presbyterian	[324, 336]
198.	Samford	[195, 200]	262.	Elon	[262, 262]	326.	Alcorn State	[251, 326]
199.	Rice	[189, 221]	263.	South Florida	[263, 282]	327.	Arkansas-Pine Bluff	[327, 347]
200.	East Carolina	[159, 226]	264.	Holy Cross	[235, 270]	328.	Youngstown State	[311, 328]
201.	VMI	[199, 251]	265.	Robert Morris	[234, 265]	329.	IUPUI	[329, 351]
202.	North Carolina Central	[182, 202]	266.	Drexel	[266, 270]	330.	Northern Arizona	[283, 330]
203.	Southeast Missouri State	[203, 203]	267.	Northern Illinois	[262, 285]	331.	Cal State Bakersfield	[331, 336]
204.	Eastern Illinois	[204, 206]	268.	Southern Utah	[215, 268]	332.	Sacramento State	[331, 332]
205.	McNeese State	[203, 205]	269.	Long Beach State	[269, 269]	333.	Cal State Northridge	[333, 333]
206.	Central Arkansas	[206, 207]	270.	Pepperdine	[270, 275]	334.	UC-Riverside	[334, 334]
207.	SIU-Edwardsville	[205, 207]	271.	Lafayette	[267, 271]	335.	Cal Poly	[335, 351]
208.	Morehead State	[208, 250]	272.	Army	[272, 272]	336.	USC Upstate	[286, 351]
209.	Southern	[203, 229]	273.	Air Force	[273, 274]	337.	Chicago State	[332, 351]
210.	UMKC	[204, 210]	274.	Columbia	[273, 293]	338.	Longwood	[326, 351]
211.	Seattle	[211, 211]	275.	San Jose State	[274, 275]	339.	Maine	[318, 351]
212.	Eastern Washington	[212, 212]	276.	Santa Clara	[276, 276]	340.	Coppin State	[310, 340]
213.	Weber State	[213, 213]	277.	Portland	[277, 277]	341.	South Carolina State	[341, 341]
214.	Portland State	[214, 254]	278.	Loyola Marymount	[278, 294]	342.	Morgan State	[342, 342]
215.	James Madison	[214, 215]	279.	Idaho State	[276, 280]	343.	Florida A&M	[343, 343]
216.	Towson	[216, 216]	280.	Houston Baptist	[207, 287]	344.	Howard	[344, 344]
217.	UMBC	[217, 230]	281.	Montana State	[280, 281]	345.	Maryland-Eastern Shore	[345, 346]
218.	Manhattan	[217, 218]	282.	North Dakota	[282, 326]	346.	Bryant	[325, 351]
219.	Fairfield	[219, 219]	283.	Stetson	[264, 283]	347.	Delaware State	[346, 351]
220.	Penn	[220, 220]	284.	Florida International	[284, 289]	348.	Alabama A&M	[328, 348]
221.	Yale	[221, 225]	285.	Kennesaw State	[284, 307]	349.	Jackson State	[349, 349]
222.	Florida Atlantic	[217, 222]	286.	Green Bay	[284, 303]	350.	Alabama State	[350, 350]
223.	Florida Gulf Coast	[223, 223]	287.	American	[273, 314]	351.	Mississippi Valley State	[351, 351]

5.4 Comparing Backedge Weight Across Sports

Table 5.52: *Backedge Weight Comparisons across Different Sports*

α	β	NFL	MLB	NBA	NHL	NCAA FB	NCAA BB
0.0	0.0	9.89%	34.74%	24.40%	29.52%	7.60%	16.3%
0.5	0.5	17.18%	36.93%	27.74%	32.81%	11.87%	20.2%
1.0	1.0	19.29%	37.56%	29.02%	33.99%	13.25%	21.5%

Because different sports leagues have different season lengths and games played per season, the backedge percentage of a ranking for one sport may be significantly higher or lower than one for another sport. In Table 5.52, we see the backedge percentages for our own rankings at different α and β values across the different sports we analyzed. One significant trend is that leagues which play fewer games, such as the NFL and NCAA Football, tend to have lower backedge weight percentages. This makes sense since in a scenario like this, the graph is going to be significantly sparser, leading to less edges needing to be considered as backedges.

In the MLB, we found our highest backedge weight percentages. Unlike the other sports we analyzed, the MLB season is made up of many series in which teams will play each other 2 to 4 times in succession. Often times, one teams will not win every game in the series, leading to at least one of the resulting games becoming a backedge. This phenomenon is the most likely reason for the high backedge weight percentages we found.

5.5 Summary

This chapter detailed our results from our algorithms and postprocesses. First, the raw results (no postprocess) from each algorithm for NFL 2018 were presented. Then each postprocess method was introduced and shown to improve the rankings. Our rankings were then compared against external rankings to show validity as well as figure out what factors may have been considered when creating these external rankings. Finally, our overall best ranking for each data set was presented.

CHAPTER 6

FUTURE WORK

Although this project explored several different approaches to sports rankings, there is still more work that can be done to improve upon and expand its results. One area that would benefit from further work relates to our pruning approximation algorithms. Currently, our *Dynamic Programming Brute Force Pruning* algorithm estimates backedge weight by calculating backedges within ranked teams and calculating backedges from ranked teams to unranked teams, which dictates how the pruning is executed. The *Dynamic Programming Brute Force Pruning* does not perform as well as our other approximation algorithms, likely because of the way it calculates weight. In particular, if it estimates poorly in the beginning, the ranking cannot be adjusted in latter steps, and it does not estimate backedge weights amongst unranked teams. If a better heuristic for estimating weight could be applied to this algorithm, it would greatly improve its performance.

Another area within this project that could be explored further involves further exploration of the ToC Structure. In the project, we were able to demonstrate that the *ToC Method* generates new rankings that have less than or equal backedge weight to the original ranking. However, we did not focus on improving a ranking's total backedge weight using the *ToC Method*. In order to achieve this, we must implement an algorithm that chooses edges that can be moved from the backedge set to the forward edge set, ultimately reducing the total backedge weight. By implementing this, we think that it would significantly improve our approximation algorithms, getting them closer to an optimal solution.

When it comes to ranking sports teams, this project takes into account game recency and point differential. These factors have a significant effect on final rankings. However, recency and point differential are not the only factors that affect game outcomes. Rankings might improve if we researched other factors affecting games such as location (home vs. away) and weather conditions.

It would also be interesting to look into expanding the scope of sports that this

project attempts to rank. This project ranks teams within several professional and collegiate sports leagues. However, it never addresses ranking sports that involve direct competition between individuals, as opposed to teams. It would be interesting to explore ranking sports such as swimming, track, and ski racing to figure out how to handle sports where several individuals compete in a single outing. It would also be interesting to explore ranking sports like boxing, where two individuals compete directly against each other.

Outside of the direct context of this project, it might be useful to explore other np-complete problems. In this project, we utilized several strategies to reduce the time complexity of solutions to the minimum feedback arc set problem. If we apply these strategies and algorithms to other np-complete problems, we would likely be able to reduce the time complexity of those problems as well. Overall, this project could expand in several different directions in the future, all of which would produce interesting results.

CHAPTER 7

CONCLUSION

We were able to successfully complete the goals of this project. Through the algorithms we developed, we generated rankings with minimal backedge weight. For professional leagues (32 or less teams), we were able to find the optimal backedge weight for a ranking. Previous attempts to find optimal rankings could only complete a ranking for 10 teams in a reasonable amount of time. However, by using dynamic programming, we were able to significantly improve the number of teams that can be ranked using brute force.

Our approximation algorithms also performed very well. *Sliding window* reliably found the optimal ranking for professional leagues when using a window size of 23, which provided immense time savings. While we cannot compare its performance to optimal rankings for NCAA sports, the rankings produced were close to real world rankings, and the backedge percentage was similar to what was found for professional leagues. *Brute Force Pruning* was able to find rankings with the minimum backedge weight even faster than *Sliding Window*, although it provided inconsistent results until many trials were conducted. *Dynamic Brute Force Pruning* gave results close to optimal, but was less successful than the other two approximations.

Using postprocesses, we were able to further improve the backedge scores for suboptimal rankings. Given several rankings with minimum backedge weights, we were able to choose the “best" one based on a second metric. Our *RoC Reorder* method produced the best postprocess results based on speed and backedge weight reduction.

Comparing our rankings to external rankings, we found that ours differed slightly more than external rankings differed from each other. However, this is to be expected as many external rankings do not focus exclusively on backedge weight, which was the main problem we were trying to solve. Our rankings are still comparable, but have a much better backedge weight. Ultimately, our ranking generation processes were able to create rankings that appear to reflect the true strength of each team by taking into account important game information and results.

BIBLIOGRAPHY

[1] Week 8 Power Rankings. http://www.espn.com/nfl/story/_/id/25055742/nfl-week-8-power-rankings-2018-players-need-step-all-32-teams. [Online; accessed 8 Feb. 2019].

[2] dunkelindex.com. About Us: The Dunkel Index. http://dunkelindex.com/the-index/. [Online; accessed 2 Feb. 2019].

[3] www.nbastuffer.com. Point Differential. https://www.nbastuffer.com/analytics101/point-differential/, 2007. [Online; accessed 31 Jan. 2019].

[4] Richard Sandomir. College Football; Margin of Victory Falls in Bowl Rating. https://www.nytimes.com/2002/06/26/sports/college-football-margin-of-victory-falls-in-bowl-rating.html, 2002. [Online; accessed 21 Jan. 2019].

[5] Arpad Elo. *The Rating of Chessplayers, Past and Present*. 1978.

[6] Nate Silver and Reuben Fischer-Baum. How We Calculate NBA Elo Ratings. https://fivethirtyeight.com/features/how-we-calculate-nba-elo-ratings/, May 2015. [Online; accessed 28 Jan. 2019].

[7] metinmediamath.wordpress.com. How to Calculate the Elo-Rating (Including Examples). https://metinmediamath.wordpress.com/2013/11/27/how-to-calculate-the-elo-rating-including-example/, 2013. [Online; accessed 21 Jan. 2019].

[8] College Football Playoff. Selection Day. https://collegefootballplayoff.com/sports/2016/10/24/_131504730172490753.aspx. [Online; accessed 2 Feb. 2019].

[9] NCAA. How the field of 68 teams is picked for March Madness. https://www.ncaa.com/news/basketball-men/article/2018-10-19/how-field-68-teams-picked-march-madness. [Online; accessed 2 Feb. 2019].

[10] NCAA.com. NCAA men's college basketball scores, news, rankings | NCAA.com. https://www.ncaa.com/sports/basketball-men/d1, 2019. [Online; accessed 14 Jan. 2019].

[11] NCAA.com. NCAA men's college football scores, news, rankings | NCAA.com. https://www.ncaa.com/sports/football/fbs, 2019. [Online; accessed 2 Feb. 2019].

[12] NFL.com. NFL.com - Official Site of the National Football League. https://www.nfl.com/, 2019. [Online; accessed 14 Jan. 2019].

[13] Read American Football. How Far Up Is NFL Scoring From Last Season? - Read American Football. https://readamericanfootball.com/2018/10/25/how-far-up-is-nfl-scoring-from-last-season/, 2019. [Online; accessed 14 Jan. 2019].

[14] MLB.com. The Official Site of Major League Baseball. https://www.mlb.com/, 2019. [Online; accessed 14 Jan. 2019].

[15] Teamrankings.com. MLB Stats - MLB Team Runs per Game on TeamRankings.com. https://www.teamrankings.com/mlb/stat/runs-per-game, 2019. [Online; accessed 14 Jan. 2019].

[16] Sky Sports. NBA News, Highlights and Videos | Sky Sports. https://www.nba.com/, 2019. [Online; accessed 14 Jan. 2019].

[17] Teamrankings.com. NBA Stats - NBA Team Points per Game on TeamRankings.com. https://www.teamrankings.com/nba/stat/points-per-game, 2019. [Online; accessed 14 Jan. 2019].

[18] Guide to 2013-14 NHL Realignment. https://www.nhl.com/news/guide-to-2013-14-nhl-realignment/c-685005. [Online; accessed 27 Feb. 2019].

[19] NHL League Averages. https://www.hockey-reference.com/leagues/stats.html. [Online; accessed 27 Feb. 2019].

[20] Andrew Butterfield and Gerard Ekembe Ngondi. *A Dictionary of Computer Science*. January 2016.

[21] D. Younger. Minimum feedback arc sets for a directed graph. *IEEE Transactions on Circuit Theory*, 10(2):238–245, June 1963.

[22] Michael R. Garey. *Computers and intractability : a guide to the theory of NP-completeness*. A Series of books in the mathematical sciences. W. H. Freeman, San Francisco, c1979.

[23] Vreda Pieterse and Paul E. Black. Algorithms and Theory of Computation Handbook. https://www.nist.gov/dads/HTML/nphard.html, September 2013. [Online; accessed 2 Feb. 2019].

[24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[25] Austin Jesse Mitchell. The Millennium Problems Are Seven Math Problems Worth 1 Million Dollars. https://curiosity.com/topics/the-millennium-problems-are-seven-math-problems-worth-dollar1-million-each-curiosity/. [Online; accessed 14 Feb. 2019].

[26] Behnam Esfahbod. https://commons.wikimedia.org/w/index.php?curid=3532181. [Online; accessed 3 Feb. 2019].

[27] Alfred, Daniel; Beader, Matthew; Jackman, Matthew; Walsh, Ryan. MQP-CEW-1801: Graph-Based Sports Rankings, 2018.

[28] C vs. C++: Comparing Two Foundations of Modern Programming. https://www.upwork.com/hiring/development/c-vs-c-plus-plus/. [Online; accessed 27 Feb. 2019].

[29] Intel Xeon Gold 6138 Processor. https://ark.intel.com/content/www/us/en/ark/products/120476/intel-xeon-gold-6138-processor-27-5m-cache-2-00-ghz.html. [Online; accessed 24 Feb. 2019].

[30] NFL Power Rankings: New Year's resolutions for every team. http://www.espn.com/nfl/story/_/id/25656646/2018-nfl-week-18-power-rankings-new-year-resolutions-every-team. [Online; accessed 7 Feb. 2019].

[31] NFL Power Rankings: Pats up, Seahawks down entering playoffs. http://www.nfl.com/news/story/0ap3000001006802/article/nfl-power-rankings-pats-up-seahawks-down-entering-playoffs. [Online; accessed 7 Feb. 2019].

[32] Power Rankings: Astros fight off their superteam rivals. http://www.espn.com/mlb/story/_/id/24811499/ranking-mlb-teams-week-26. [Online; accessed 7 Feb. 2019].

[33] Final MLB Power Rankings. http://www.rotoworld.com/articles/mlb/82577/522/final-mlb-power-rankings?pg=2. [Online; accessed 7 Feb. 2019].

[34] NBA Power Rankings: Ready for a wild West finish? http://www.espn.com/nba/story/_/id/23083860/nba-power-rankings-our-expert-panel-unveils-rankings-week-26. [Online; accessed 7 Feb. 2019].

[35] Week 26 Power Rankings: Cleveland Cavaliers and Golden State Warriors look vulnerable as regular season closes. http://www.nba.com/powerrankings/2017-18-week-26. [Online; accessed 7 Feb. 2019].

[36] Power Rankings: The Farewell Tour edition. http://www.espn.com/nhl/story/_/id/22759910/nhl-2017-18-power-rankings-greg-wyshynski-farewell-tour-edition. [Online; accessed 7 Feb. 2019].

[37] Power Rankings: Teams on the Bubble Battling to Make the Playoffs. https://www.si.com/nhl/2018/04/02/power-rankings-playoff-bubble-teams-stars-blues-panthers. [Online; accessed 7 Feb. 2019].

[38] AP Top 25 Poll - Week 15. https://collegefootball.ap.org/poll/2018/15. [Online; accessed 7 Feb. 2019].

[39] FBS Football Rankings - College Football Playoff. https://www.ncaa.com/rankings/football/fbs/college-football-playoff. [Online; accessed 7 Feb. 2019].

[40] 2017-18 College Basketball Polls. https://www.sports-reference.com/cbb/seasons/2018-polls-old.html. [Online; accessed 7 Feb. 2019].

[41] Men's College Basketball Rankings - Week 18. http://www.espn.com/mens-college-basketball/rankings/_/week/18/year/2018/seasontype/2. [Online; accessed 24 Feb. 2019].