NFL Survivor Optimization

Marco I. Bornstein **Ragib Mostofa** **Temitayo Ajayi** **Andrew J. Schaefer**

November 23, 2018

1 Problem Statement

The National Football League (NFL) has undoubtedly become the most popular sport in the U.S. In the past two years, the average TV audience for an NFL game was 17.9 million and 16.5 million respectively[1]. This extremely large interest in the NFL has created a huge market for football-related games. The market for all fantasy football games has been approximated at 70 billion dollars[2]. One major component of this market is the very popular Survivor Football competition.

On espn.com, the Survivor Football game mode is named the "Eliminator Challenge". In only the first 34 largest public groups (there are thousands of groups), there are 470,000 members. Thus, it is without a doubt that there are well over one million people playing Survivor Football on ESPN alone.

The large audience for Survivor Football coupled with the large market for fantasy football games has created many competitions with substantial monetary prizes for those successful at Survivor Football. Thus, discovering an algorithm to boost one's likelihood of winning a competition would be greatly sought after.

Survivor Football is a game where a player picks one NFL team per week to win its current matchup. The catch, however, is that over the course of the entire season, participants are not allowed to choose the same NFL team twice. While different competitions have different elimination settings, the goal in all competitions is to perfectly select a winning NFL team for all 17 weeks of the regular season.

With as many as 32 teams in the National Football League (NFL) and 17 weeks in the regular season, there are exactly 32!/(32 - 17)! possible pathways that a player can choose to follow, even when the aforementioned constraints have been taken into account. Due to these constraints and the goal of maximizing the number of winning picks while restricting the number of losing picks to a minimum, an optimization problem arises. There must now be an optimal path (with variables including strength of opponents in a matchup, location of the game, etc.) in selecting the correct team every week such that all constraints are satisfied.

2 Literature Review

A significant amount of literature with regards to both this particular problem and similar problems in the context of other sports leagues and associations, such as the National Hockey League, Major League Baseball, and NCAA Men's Basketball, exist. The following literature review will examine methods to optimize a survivor-style sports league competition as well as how the optimization changes in a pool competition format.

Bergman & Imbrogno (2017) optimized the NFL Survivor Football competition for a single-entry model as well as a multiple-entry model. In the single-entry model, the optimization problem was a linear binary IP. The optimized model roughly doubled the likelihood of surviving all 17 weeks, 1.76% vs. 0.86%, compared to the greedy strategy of selecting the highest remaining win probability each week. (Bergman & Imbrogno, 2017). The paper also concluded that 8 weeks is the optimal number of look-ahead weeks.

Clair & Letscher (2006) analyzed the optimal strategies for sports betting pools, including NFL weekly competitions. The authors used ESPN pool competitions to test whether an increasing pool size should effect their betting strategy. They found that they were correct in their initial hypothesis that as a pool becomes so large, it becomes increasingly beneficial to stray from the favored teams each week (which is greatly correlated with the pool consensus).

Breiter & Carlin (1997) discuss optimization strategies for NCAA Men's Basketball Tournament bracket competitions. The two common strategies, selecting the highest seed and selecting the highest 4 seeds but picking upsets on the remainder, are compared to a Monte Carlo approach of simulating the tournament and choosing the result (bracket) which has the greatest expected value. Their results show that the Monte Carlo approach was much more successful than the alternatives.

In the book *Mathletics* (Winston, 2009), Winston describes the "Winston Equations". These equations were created by Winston to relate the final margin of victory for an NFL team to the Vegas line of the

game. It is explained that the Vegas line can act as the mean of a normal distribution curve, with a standard deviation between 13-14, and the margin of victory can accurately be approximated as a random number taken from the normal distribution curve. This result shows that the Vegas line can accurately predict an NFL match-up.

3 Probability Matrix Creation

The probability matrix we create for our NFL Survivor Football simulations implements an Elo rating system. The Elo rating system has been utilized in chess for decades, accurately computing the winning percentage of chess matches. The system works by assigning each new player an initial rating (usually around 1200 for chess). As this player plays more chess matches, their rating will change depending upon the result of each match (win, loss, or tie) as well as the rating, or skill-level, of their opponent (ex. a win versus a higher rated opponent will raise one's score more than a win versus a lower ranked opponent). Thus, the rating system is dynamic and adjusts after each match. In this system, determining the winning percentage of chess match between two players is simple. All the information needed to determine the winning percentage of a match is the Elo rating of both players. The way that winning percentage is calculated from an Elo rating is as follows: If Team 1 has an Elo rating of X, and Team 2 has an Elo rating of Y, then the winning percentage of Team 1, E_1, is as follows,

$$E_1 = \frac{1}{1 + 10^{(\frac{Y-X}{400})}}$$

Therefore, the winning percentage of Team 2, E_2, is,

$$E_2 = 1 - E_1$$

Modifying the Elo rating system to work in the NFL is rather straightforward. The main difference is that the initial, or average, rating for each NFL team is 1500. In fact, the website FiveThirtyEight, a website that meshes statistics with sports, has already created an Elo rating model for the NFL. Their Elo ratings are extensive, even having ratings for each team as far back as the creation of the NFL. For our project, we use the Elo ratings from FiveThirtyEight to power our NFL simulations. FiveThirtyEight updates the NFL Elo ratings after every week, accounting for the results of each NFL matchup from the previous week.

Building the entire probability matrix is slightly more complicated. To start, we create a NFL season schedule matrix as well as a binary home-field matrix. The NFL season schedule matrix is a $i \times j$ matrix where i is the number of NFL teams and j is the number of regular season weeks. The team index entered into the $i, j-th$ entry corresponds to the opponent whom team i will play in week j. Thus, we know which teams will play each other for each week of the NFL season. The binary home-field matrix is a $i \times j$ matrix where once again i is the number of NFL teams and j is the number of regular season weeks. If the $i, j-th$ entry is 1, then team i plays at home during week j. If the $i, j-th$ entry is 0, then team i plays on the road during week j.

The probability matrix is also a $i \times j$ matrix where i is the number of NFL teams and j is the number of regular season weeks. To complete the probability matrix, take the current week Elo from FiveThirtyEight for both team i and its opponent, the corresponding $i, j-th$ entry of the NFL season schedule matrix, and apply the Elo winning percentage formula written above. Repeat this process for each team, i, and each week, j. The final component of this process is the addition of the home-field advantage factor into the team rankings. To incorporate this, if the $i, j-th$ entry of the binary home-field matrix is 1, then a designated home-field Elo point total is added to team i. Therefore, the Elo winning percentages are calculated with the addition of the home-field advantage Elo rating boost.

4 Linear Programming Model

Let M be a set containing indices 1 to 32, representing all of the National Football League (NFL) teams. Let N be another set containing indices 1 to 17 representing the ordered regular season weeks.

$$M = \{1, 2, ..., 32\} \tag{1}$$

$$N = \{1, 2, ..., 17\} \tag{2}$$

Let n be the number of initial weeks during which the model selects the team with the highest win probability each week. Furthermore, let N_0 be the set containing the indices of these initial n weeks.

$$N_0 = [n] \tag{3}$$

The year to year performance of many NFL teams changes drastically. Each team in the NFL plays 16 games per regular season. Compared to the National Basketball Association (NBA), National Hockey League (NHL) and Major League Baseball (MLB), which have 82, 82 and 162 regular season games per team respectively, the NFL has a much smaller slate of regular season games. In other words, the sample size for each team is much smaller than for the 3 aforementioned leagues. Statistically, smaller sample sizes have greater variance, which means there is considerable randomness in the end-of-season win-loss record for each team. On top of this, every NFL team experiences a high rate of roster turnover in the off-season. For example, the average NFL team in 2017 (as of

June 19th) experienced a roster turnover of 22.09% (or approximately 11.7 new players on each team's 53-man roster)[1]. With a high roster turnover rate and uncertainty over true team performance due to randomness from the year before, it is common to find multiple teams each season that greatly exceed their initially expected performance (and vice versa).

To ensure this expected change in team performance does not affect the model, we implement a sliding window. In essence, a window is an interval of $(\tau+1)$ weeks for which the optimization program below is solved. The number of windows is denoted by k, where,

$$k = |N| - \max N_0 - \tau \tag{4}$$

For each window, $i \in [k]$, we recursively define a set, N_i of relevant week indices.

Base Case: $N_1 = \{\max N_0, 1+\max N_0, ..., \tau+\max N_0\}$ (5)

Recursive Case: $N_i = \{1 + \min N_{i-1}, 2 + \min N_{i-1}, ..., (1+\tau) + \min N_{i-1}\}$

The collection of windows, N_i, constitute the sliding windows. As a consequence,

$$N = \bigcup_{i=0}^{k} N_i \tag{6}$$

4.1 Formulation of Model

For each window, $i \in [k]$, the following optimization program is solved.

Let $x_{i,j} \in \{0, 1\}$ be a binary decision variable which denotes whether team i is selected by the model on week j such that,

$$x_{i,j} = \begin{cases} 1 & \text{if team i is predicted to win on week j} \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $i \in M$ and $j \in N_i$

The objective of the model is to maximize the probability of correctly selecting a single NFL team to win each week of the regular season. Let $P_{i,j}$ denote the probability of team i winning in week j, where $i \in M$ and $j \in N_i$. Hence, the objective function that the optimization model should aim to maximize is,

$$Maximize\ W = \prod_{j \in N_i} \mathbf{P}_{j,\cdot}^{T} \mathbf{x}_{\cdot,j} \tag{8}$$

Every team is selected at most once during the course of the 17-week regular season.

$$\sum_{j \in N} x_{i,j} \leq 1, \forall i \in M \tag{9}$$

Only 1 team can be selected by the model each week.

$$\sum_{i \in M} x_{i,j} = 1, \forall j \in N \tag{10}$$

4.2 Linearization Of Objective Function

Notice that the objective function provided above (Equation 8) is non-linear since it includes the product of the decision variables $x_{i,j}$. Before initiating an attempt at linearization, we establish a couple of important statements that should help with the process of linearization.

Lemma 1: Let $f(x) = log_b x, x > 0$. Then f is a monotonically increasing function.

Proof: Computing the derivative of $f(x)$,

$$f(x) = log_b x$$

$$f'(x) = \frac{d}{dx} log_b x$$

$$f'(x) = \frac{d}{dx}\left(\frac{ln\ x}{ln\ b}\right) = \frac{1}{ln\ b} \times \frac{d}{dx}(ln\ x)$$

$$f'(x) = \frac{1}{xln\ b}$$

Assuming that only logarithms with bases greater than 1 are being considered, i.e. $b > 1$, it can be inferred that $ln\ b > 0$. Using the fact that $x > 0$ (by definition), we may conclude that $f'(x) = \frac{1}{xln\ b} > 0$. Since the derivative of f is positive, f must be a monotonically increasing function. ■

Proposition 1: Let $A = \{f(x_1, x_2, ..., x_n)\}$ be an ordered set (ascending), where f is a function. Then,

$$x_n \in argmax\{g(f(x_1)), g(f(x_2)), ..., g(f(x_n))\}$$

if g is a monotonic mapping.

Proof: Since g is a monotonic function, the elements $\{g(f(x_i))\}_{i=1}^{n}$, are ordered, i.e. mapping $f(x_i)$ to $g(f(x_i))$ preserves the ordering of the elements in A. Hence, as $x_n \in argmax A$, it must also be true that $x_n \in argmax\{g(f(x_i))\}_{i=1}^{n}$. ■

Using lemma 1 and proposition 1, one may argue that applying a logarithmic transformation to the objective function of our optimization model above should not alter the optimal solution, i.e. the set of team selections. As such, we apply the transformation, with

the end goal of linearizing the objective function, as follows,

$$\begin{aligned} W' = log_b W &= log_b \prod_{j \in N_i} \mathbf{P}^T_{j,\cdot} \mathbf{x}_{\cdot,j} \\ &= \sum_{j \in N_i} log_b \mathbf{P}^T_{j,\cdot} \mathbf{x}_{\cdot,j}, b \in \mathcal{Z}_{>1} \\ &= \sum_{j \in N_i} log_b \Big(\sum_{k=1}^{m} P^T_{j,k} x_{k,j} \Big) \end{aligned}$$

Despite the logarithmic transformation, the objective function above remains non-linear as the logarithm function is being applied to the decision variables, $x_{k,j}$. However, using constraint (10), it is possible to further simplify the function to arrive at the desired form. Notice that since a single team is picked every week, only one $x_{k,j}$ term in the inner summation may equal 1 while others evaluate to 0. As a result, it is now possible to manipulate the function to move the logarithm inside the inner summation, where it is applied to only the probability term, $P^T_{j,k}$ and not the decision variables. Denoting the winner of week j by k' and following the process outlined above, we derive,

$$\begin{aligned} W' &= \sum_{j \in N_i} log_b \Big(\sum_{k=1}^{m} P^T_{j,k} x_{k,j} \Big) \\ &= \sum_{j \in N_i} log_b P^T_{j,k'} \quad (*) \\ &= \sum_{j \in N_i} \sum_{k=1}^{m} (log_b P^T_{j,k}) x_{k,j}, \end{aligned}$$

which is the desired linearized form of the objective function for our model.

(*) since $x_{j,k} = \begin{cases} 1 & \text{if k = k'} \\ 0 & \text{otherwise} \end{cases}$

5 Shortest Path Model

In this section we establish that the binary integer program presented in Section 4 is equivalent to a shortest path problem. We first provide a formal definition of the directed graph on which the shortest path model is defined and follow up with a simple proof of the equivalence of the two models.

5.1 Formal Definition of Graph

Assuming there are n teams and m weeks in the regular season of the National Football League, let $N = [n]$ be the set of indices ranging from 1 to n inclusive. Furthermore, define $\mathcal{P}(N)$ to be the powerset of N. Finally, define $\mathcal{S}_{\mathcal{P}(N)}$ as,

$$\mathcal{S}_{\mathcal{P}(N)} = \bigcup_{x \in \mathcal{P}(N) \setminus \{\emptyset\}} Perm(x),$$

where $Perm(x)$ denotes the set of all permutations on the elements of set x. Note that since the order of the elements in the sets contained in $\mathcal{S}_{\mathcal{P}(N)}$ matters, these sets will be referred to as ordered tuples from here onwards.

Let G = (V, E, w) be the weighted directed graph associated with the shortest path model, where V is the set of vertices, E is the set of edges and w is the weight function.

5.1.1 Vertex Set, V

The vertex set, V for the graph, G is defined as follows.

$$V = \{s, t\} \cup \{v_l \mid \forall l \in \mathcal{S}_{\mathcal{P}(N)}\},$$

where s and t are the start and terminal nodes respectively.

5.1.2 Edge Set, E

The edge set E for the graph, G, consists of ordered pairs of the form (u, v), which denotes that there exists a directed edge leading from node u to node v, where $u, v \in V$. In order to define the edge set, we consider 3 different cases.

Firstly, the start node must be connected to every team node for the beginning week of the season. Denoting the set of all such edges as E_1,

$$E_1 = \{(s, v_l) \mid |l| = 1\},$$

where $|l|$ is the cardinality of the tuple, l

Next, all inner nodes (i.e. the nodes excluding the start and terminal nodes) are connected by an edge if their index tuples contain a sequence of team selection that is acceptable according to previously discussed constraints. Let E_2 denote the set of all such edges. Then, E_2 is defined as,

$$E_2 = \{(v_l, v_{l'}) \mid l' = l \cdot (x)\ \forall x \in N \setminus l, \forall l \in \mathcal{S}_{\mathcal{P}(N)}\},$$

where $l \cdot (x)$ denotes the concatenation of the tuple (x) to the end of tuple l.

Finally, all nodes that have an out-degree of zero must be connected to a common terminal node via an outgoing edge. Hence, the set of all such edges, E_3 is defined as,

$$E_3 = \{(v, t) \mid \nexists (v, u) \in E_1 \cup E_2,\ \forall u, v \in V \setminus \{t\}\}$$

Now that all three cases have been clarified, we define E as follows.

$$E = E_1 \cup E_2 \cup E_3$$

5.1.3 Weight Function, w

Notice that since every edge in the graph has a cost associated with it, there exists a mapping $w : E \mapsto \mathcal{R}$ which, given a certain edge $e \in E$, maps the edge to its corresponding cost from the matrix, $C \in \mathcal{R}^{m \times n}$ (this matrix is defined in Section 1.1.4). Assuming that the tuple, l and matrix C used below are zero-indexed, the function w is defined as follows.

$$w(e, C) = \begin{cases} C_{0,l_0} & \text{if } e = (s, v_l) \in E_1 \\ C_{|l|,l'_{|l|}-1} & \text{if } e = (v_l, v_{l'}) \in E_2 \\ \delta & \text{if } e = (v_l, t) \in E_3 \end{cases}$$

where $\delta \in \mathcal{R}_{[0,1]}$ and $C_{i,j}$ is the entry on the i^{th} row and the j^{th} column of the matrix C.

5.1.4 Cost Matrix, C

The cost matrix C, used by the weight function above is a derivation of the probability matrix, P defined in Section 4.1 during the formulation of the original binary model. One fundamental difference between the two models lies in the objective functions - while we wish to maximize the sum of the logged probabilities of the selected teams in the first model, the shortest path model, by definition, aims to minimize the sum of the costs associated with the selected edges. Hence, for the two models to be equivalent, the cost matrix must be defined as some sort of inverse of the logged probabilities. As such, we define the cost matrix C, as follows.

$$C_{i,j} = -log_b P_{i,j}, b \in \mathcal{Z}_{>1},$$

where $C_{i,j}$ and $P_{i,j}$ are the entries on the i^{th} row and the j^{th} column of the matrices C and P respectively.

One potential edge case that may arise when taking the logarithm of the original probabilities is in the case of bye weeks. Statistically, the probability of a team i winning during a bye week, j is 0. However, for the purposes of computing the cost matrix, C, when we are expected to take the logarithm of all entries in P (including bye-weeks), we define all such $P_{i,j} = \epsilon$, where ϵ is a very small positive number. As a result, the corresponding entry in C for bye weeks should be the negative logarithm of ϵ, which yields a large positive number. This is in perfect accordance with the objective of the shortest path model, since edges with high costs are not included in the optimal path, i.e. bye weeks are ignored, as desired.

An example graph for 3 teams and 3 weeks of play has been provided below.

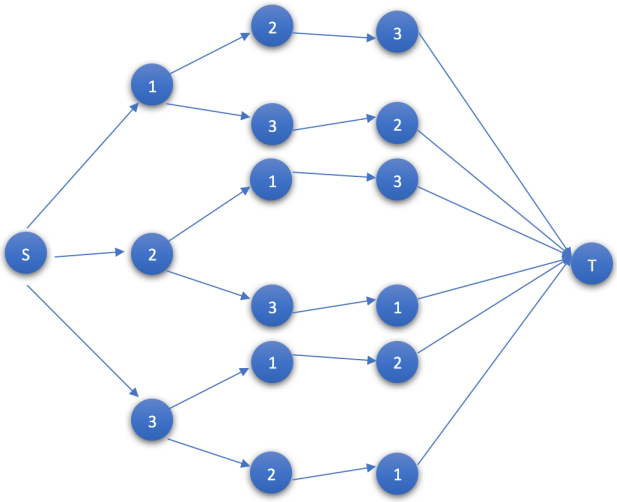


Figure: Example Graph with 3 teams and 3 weeks of play

5.2 Proof of Equivalence of Models

The goal of this section is to prove that the original binary integer program that has been proposed as a solution to this problem in Section 4.1 is equivalent to the shortest path model on graph G defined above. For ease of reference, we provide the formulation of the shortest path model below.

Given a directed graph (V, E) with source node s, sink node t, and cost w_{ij} for each edge (i, j) in E, consider the program with variables $x_{i,j}$,

$$Minimize \sum_{i,j \in E} w_{i,j} x_{i,j}$$

subject to

$$\sum_j x_{i,j} - \sum_j x_{j,i} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise} \end{cases} \quad , \forall i$$

$$x \in \{0, 1\}$$

5.2.1 Equivalence of Feasible Regions

Let F_1 denote the feasible region of the optimization model in Section 4.1 and F_2 denote the shortest path model in Section 5.2.

It is apparent from the definition of the graph, G, that any feasible path from the source node, s to the sink node, t is also feasible under the constraints presented in the optimization model, i.e. a single team is selected every week and that every team is selected at most once throughout the course of the entire season. Moreover, every selection of teams by the optimization model can be represented as a unique path on graph,

G. Since any arbitrary element in F_1 and its corresponding element in F_2 point to the same solution, i.e. set of selected teams, the two feasible regions, F_1 and F_2 are equivalent.

5.2.2 Equivalence of Objective Functions

The objective of the optimization model in Section 4.1 is to maximize $\sum_{j \in N_i} \sum_{k=1}^{m} (log_b P_{j,k}^T) x_{k,j}$ while the objective of the shortest path model is to minimize $\sum_{i,j \in E} w_{i,j} x_{i,j}$. Since the weights on the directed edges are taken from the cost matrix C, one can rewrite the objective as minimizing $\sum_{i,j \in E} C_{w,j}^T x_{i,j}$, where w is the week during which team j is chosen. Moreover, as the cost matrix is derived from the probability matrix, we may use the definition in Section 5.1.4 to simplify the objective to minimizing $\sum_{i,j \in E} -\log_b P_{w,j}^T x_{i,j}$ which can be further simplified to maximizing $\sum_{i,j \in E} \log_b P_{w,j}^T x_{i,j}$. Notice that following from the discussion in the subsubsection above, it is apparent that this objective is equivalent to the objective of the original optimization and hence, we may conclude that the two objective functions are equivalent.

6 Algorithmic Runtime Complexity

6.1 Runtime Complexity of Dijkstra

A crude implementation of Dijkstra's algorithm using adjacency matrices has a wort-case runtime complexity of $O(|V|^2)$, where V is the vertex set as above. Using such an implementation it becomes impossible to run simulations for $\tau = 3$ or a moving window of 4 weeks, in a reasonable amount of time. The quadratic runtime may be improved by employing a slightly variant implementation that utilizes priority queues to keep track of unvisited nodes. This reduces the runtime to $O(|E| + |V| log_2 |V|)$, where E is the edge set as above. The improvement from a quadratic runtime complexity to a loglinear one allows us to run simulations with a window of 4 weeks but not longer.

6.2 Top k teams

Even with the improvement in algorithmic runtime discussed above, it is difficult to run simulations beyond a moving window of 4 weeks. The main bottleneck lies with the size of the input graphs. With as many as 32 teams, building the graph alone takes long periods of time. In order to avoid this problem we propose a pruning mechanism that considers only the best teams in any given week without interfering with the optimal solution. An obvious idea is to build the graph using the union of top k teams from every week of the season when $\tau = k - 1$, i.e. when a moving window of k weeks is used. However, we propose a different pruning method where we include just the top k teams every week that have not been selected yet. We shall now prove that using this method and a lower bound of $k \geq \tau + 1$ on the number of teams considered we should still be able to reach the same optimal path when solving the shortest path problem on our graph G.

Proposition 2: Define G' to be the edge-induced subgraph of G obtained through the pruning method outlined above. Then, the shortest path model on the graph G yields the same optimal path as the shortest path model on the subgraph G', given $k \geq \tau + 1$.

Proof: We assume for the sake of contradiction that the optimal path in subgraph G' and the optimal path in the original graph G are different. Suppose that the two paths differ on week k, i.e. the teams selected by the two paths in week k are different, where $1 \leq k \leq K$. It follows from this, that, given an ordered array of the probabilities of teams that can be considered for week k, the optimal path in G chooses a team with a probability lower than those with the first K probabilities in said array. Even if all the teams selected for the ensuing $K - k$ weeks form a subset of the top K teams in week k, there exist at least one team with a higher probability than the team selected for the optimal path in graph G. This is a contradiction, since it leads to the conclusion that there exists a path more optimal than the one being considered. Moreover, the team selected in week k must be among the top K teams for that week. Therefore, the shortest path model on the graph G yields the same optimal path as the shortest path model on the subgraph G'. ■

7 Simulations

The NFL simulation process utilizes the probability matrix as well as the Week 1 Elo ratings from FiveThirtyEight. The basis from the simulation process comes from Wayne Winston, and his book Mathletics. The website Pro Football Reference ($https://www.pro-football-reference.com/about/win_prob.htm$), describes that "Winston posited that the final margin of victory for an NFL team in a given game can be approximated as a normal random variable with a mean of the Vegas line and a standard deviation between 13-14". From Winston's observation, we can simulate any NFL game by randomly selecting a number from a normal distribution with the mean of the normal distribution being the spread of the game. The spread is simply the expected margin of victory by the favored NFL team.

One can easily calculate the spread of an NFL game from the Elo ratings. To do this, one must find the difference in the Elo ratings of the two teams (accounting for home-field advantage) and divide by 25. The team with the higher Elo rating is expected to win by this

amount. Therefore, to simulate an NFL regular season week, all one must do is derive the spreads of each game, and randomly generate a number from its respective normal distribution. The randomly generated number is the resulting point differential of the simulated game. If the number is positive, then the favored team won, and vice versa. This allows us to simulate a match-up between two NFL teams. Furthermore, the point differential is also important in determining how much a team's Elo rating changes by. FiveThirtyEight determines how much a team's Elo rating changes by implementing the following sets of equations.

$$ {}_nR_i = {}_oR_i + K * M_{oV}M_{ij} * (S_{i,j} - \mu_{i,j}) $$

$$ M_{oV}M_{ij} = ln(|Pd| + 1) \times \frac{2.2}{[0.001 \times (R_W - R_L)] + 2.2} $$

$$ \mu_{ij} = \frac{1}{1 + 10^{\frac{({}_oR_i - {}_oR_j)}{400}}} $$

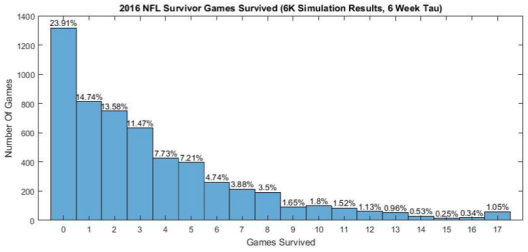
$$ S_{ij} = \begin{cases} 1, & \text{if Team i wins against Team j} \\ \frac{1}{2} & \text{if Team i ties against Team j} \\ 0, & \text{if Team i loses against Team j} \end{cases} $$

K is a scalar which assists in the updating process. FiveThirtyEight determined K to equal 20. The Actual Result is binary, 1 if the team won or 0 if it lost. μ_{ij}, or expected result, is the winning probability originally determined before the match-up occurred. Pd is the point differential determined from the randomly generated number.

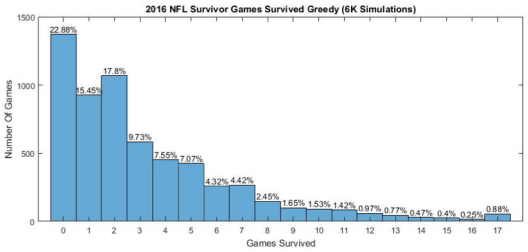
From the set of equations above, we can then determine the changes in Elo which will become the new Elo ratings for the next regular season week. The NFL simulation repeats the processes above for each week until the season is over. For each week, we run our algorithm to select our optimal team for the current week. We keep track of whether the optimal team selected each week for the optimization problem won, and thus whether we survived the week. To do so, we determine what the point differential of the simulated game was for the selected team. If the point differential is positive in favor of our team, then we continue to run the simulation and mark that we successfully survived the current week. If not, then we lose the competition and stop the simulation. We also ensure that the optimal team we selected can no longer be selected in the future. We repeat this process until either we lose or win all 17 regular season week match-ups.

8 Simulations Results

Results from 6000 NFL Simulations with $\tau = 6$.



Results from 6000 NFL Simulations with $\tau = 0$, otherwise known as the "greedy" method.



The results from the two methods show that our optimized method was superior than the greedy method in the NFL Survivor Football competition. The optimized method led to a perfect record (each team selected over the course of the NFL season won) in 1.05% of the simulations. The greedy method led to a perfect record in only 0.88% of the simulations. Therefore, optimizing the competition yielded an increase in the success rate of a perfect record by approximately 19.3% over the standard greedy method.

9 Conclusion

This report examined the NFL Survivor Football competition and proposed a possible strategy to optimize the chances of surviving the competition without a single loss. We used Elo rankings to model the probability distribution of win percentages of teams throughout an NFL season. We then modeled the problem as a linear optimization program which was shown to be a shortest path problem on a directed acyclic graph (DAG) that we define. Due to the large size of our self-defined DAGs, a crude implementation of Dijkstra's Algorithm failed to determine the optimal solution in a reasonable amount of time. As a result, we proposed a pruning method to reduce the size of the DAGs and proved that this method does not alter the optimal solution. Using this new edge-induced subgraph of our original DAG, we ran 6000 simulations of the 2016 NFL season and compared the results with the greedy method (i.e. select the team with the greatest winning percentage each week). It was determined that optimizing an NFL Survivor Football competition is an effective way of boosting ones probability of attaining a perfect record in the competition. Instead of utilizing the greedy method, using Dijkstra's algorithm to optimize the competition increases the likelihood of a perfect record by 19.3%.

References

1. Rovell, D. (2017, January 05). NFL TV viewership dropped an average of 8 percent this season. Retrieved September 2, 2017, from $http://www.espn.com/nfl/story/_/id/18412873/nfl-tv-viewership-drops-average-8-percent-season$

2. Roster Turnover in 2017: Number 20-11. (n.d.). Retrieved September 2, 2017, from https://overthecap.com/roster-turnover-2017-number-20-11

3. Bergman, D, and Imbrogno,D. "Surviving a National Football League Survivor Pool." Operations Research, Feb. 2017, doi:10.1287/opre.2017.1633.

4. Winston, W. L. (2012). Mathletics: How Gamblers, Managers, and Sports Enthusiasts Use Mathematics in Baseball, Basketball, and Football. Princeton, NJ: Princeton Univ. Press.

5. Breiter, D. J., & Carlin, B. P. (1997). How to Play Office Pools If You Must. Chance, 10(1), 5-11. doi : 10.1080/09332480.1997.10554789

6. Clair, B., & Letscher, D. (2007). Optimal Strategies for Sports Betting Pools. Operations Research, 55(6), 1163-1177. doi : 10.1287/$opre$.1070.0448