Analyzing Elo Ratings For NCAA Men’s Division 1 Hockey

Alex Tidd

Advisors: Matt Higham and Robin Lock

2025-05-05

Table of contents

## 1 Abstract

There are 64 NCAA Division 1 Men’s Hockey teams. In NCAA Division 1 Men’s Hockey, the U.S. College Hockey Online (USCHO) provides the official rankings on the 64 total teams using a system that relies on expert votes. However, this method is not perfect as there is no formal quantitative analytics involved in the voting. To improve on this, a chess ratings system, called Elo, is commonly used in many sports to quantitatively rate players, weighing strength of opponent and recency of match. In this project, we modify the Elo system for NCAA Division 1 Men’s Hockey by adding in weights for game goal differential (so that games in which the score differential was large potentially result in a larger bump in Elo for the winning team) and home-ice advantage (so that the team playing on home-ice has an adjusted probability of winning the game).

## 2 Introduction

NCAA Men’s Division 1 Ice Hockey has 64 teams. This inherently makes ranking teams extremely hard. It is impossible to rank teams in the same manner as pro leagues like the NHL and junior leagues like the 3 CHL leagues, where teams are ranked based off of a record-point system. This is due to the fact that many teams end up with similar records, but with a league so large and full of different conferences, the strength of a victory/loss is not the same for every game. Some conferences are weaker than others, so if a weak-conference playing team has a great record, it would not be fair to rank them higher than a team in a strong conference with a mediocre record.

To rank teams, the current method is an “expert vote” system done by US College Hockey Online (USCHO). This method takes votes by “experts” and ranks teams based on how many votes each team receives. In general, these rankings are pretty accurate. However, it lacks any true quantitative analysis, instead relying on the opinions of voters.

This project aims to solve the issue presented by using a ratings system used in chess to accurately rate and rank players, called Elo, for use in collegiate hockey. In chess, Elo ratings calculate an “expected outcome” using 2 players’ ratings. This expected outcome is essentially a “win probability” for each player. After the expected outcome is calculated, it is compared to the actual outcome. Depending on the outcome, a player’s new rating will either be increased (if the player won) or decreased (if the player lost). The benefit to using Elo, is that the rating system takes into consideration your strength, the opponent’s strength, and the recency of the match. Big wins/losses garner big adjustments in rating, whereas an expected win/loss won’t inflate/tarnish a rating very much. The equation for expected outcome is:

Where:

* is the player’s expected outcome.
* is the player’s pregame rating.
* is the opponent’s pregame rating.

To update the rating after the match is played, the update equation is:

Where:

* is the player’s postgame rating.
* is the player’s pregame rating
* is an update factor that scales the amount of points a player changes by.
* is the outcome of the match, 1 for a win, 0 for a loss.
* is the player’s expected outcome.

Example:

Player A has a pregame rating of 2000 and beats Player B with a rating of 2200. To find Player A’s new rating (k is set to 100 for this example):

We see that Player A’s new rating is 2075.975. It grew by almost 76 points due to them beating a player they were not expecting to beat. Had Player A had a pregame rating off 2200 and Player B had a pregame rating of 2000, with Player A still winning, Player A’s new rating would be 2224.025, only a 24.025 point increase, as Player A was expected to win.

The goal for this project is to use this base Elo ratings system, incorporate factors such as home ice advantage and goal differential, optimize k, and make a ratings model that can accurately predict outcome and rank teams accordingly. This paper shows the steps of finding data, creating a ratings function, and optimizing our constants.

## 3 Data

There were two main data sets used for this project. The full 2023-2024 NCAA Men’s Division 1 Ice Hockey schedule, which after wrangling, included 1166 games with variables: date, game\_type, away\_team, away\_score, home\_team, home\_score, overtime, neutral\_site, score\_diff, and outcome. The score\_diff and outcome variables are in reference to the home team, with outcome being either 1: win, 0.5: tie, or 0: loss. The next data frame was the entire 2024-2025 NCAA Men’s Division 1 Ice Hockey schedule. The schedule contains 1153 games which had the same variables and setup as the 2023-2024 season. Both data frames were scraped from the website: College Hockey News.

2023-2024 Season:

schedule2324 |> slice(1:4)

# A tibble: 4 × 10  
 date game\_type away\_team away\_score home\_team home\_score overtime  
 <date> <chr> <chr> <dbl> <chr> <dbl> <lgl>   
1 2023-10-07 Non-Conference Lake Super… 2 Michigan… 5 FALSE   
2 2023-10-07 Non-Conference Mass.-Lowe… 2 Alaska-A… 3 TRUE   
3 2023-10-07 Non-Conference Denver 7 Alaska 3 FALSE   
4 2023-10-07 Non-Conference Lindenwood 1 Air Force 4 FALSE   
# ℹ 3 more variables: neutral\_site <lgl>, score\_diff <dbl>, outcome <dbl>

2024-2025 Season:

schedule |> slice(1:4)

# A tibble: 4 × 10  
 date game\_type away\_team away\_score home\_team home\_score overtime  
 <date> <chr> <chr> <dbl> <chr> <dbl> <lgl>   
1 2024-10-04 Non-Conference Michigan S… 2 Lake Sup… 1 TRUE   
2 2024-10-04 Non-Conference Minnesota … 5 Michigan 2 FALSE   
3 2024-10-04 Non-Conference Arizona St… 8 Air Force 1 FALSE   
4 2024-10-04 Non-Conference Bowling Gr… 2 Mercyhur… 1 FALSE   
# ℹ 3 more variables: neutral\_site <lgl>, score\_diff <dbl>, outcome <dbl>

A third data frame for initial rankings was created from scratch using knowledge from the final 2022-2023 season. Initial rankings were created by ranking teams from 2000 to 1300. The top 8 teams from the 2022-2023 season were assigned the rating of 2000, every 8 teams, the assigned rating would decrease by 100. This initial ratings file was used as the initial ratings file to run with the 2023-2024 season to get starting ratings for the 2024-2025 season.

To get these “accurate” ratings for the 2024-2025 schedule, the 2023-2024 schedule was run through the Elo function with k = 100, a value used based off a general exploration which gave relatively accurate rankings compared to the final 2023-2024 season. After, the ratings were scaled using the equation below:

Where:

* is the preseason ratings.
* is the final ratings from the previous season.

This adjustment of rankings is used to adjust for player turnover, new coaching hires, and off season improvement. The formula used is based off of the formula used by Fivethirtyeight in their NHL Elo ratings. These final adjusted ratings are what was used for the intial rating in the 2024-2025 season.

## 4 Elo Model

To add goal differential into the Elo system, it was determined that it should be placed in the “rating update” portion of the Elo system, due to goal differential being a “post-game” statistic. On the other hand, home ice advantage affects the predicted probability of a home team winning. Therefore, it was added to the “expected outcome calculation” portion of the Elo system. Home ice advantage was also incorporated as a simple point “boost” for the home team instead of a multiplier, based off of FiveThirtyEight’s NHL Elo model.

In order to obtain the “best” values for each parameter, a grid search method was used to optimize each of the three parameters to lower the mean absolute residual of the season. In this case, the “absolute residual” is calculated by the absolute value of the difference of the game outcome and the expected outcome of the home team. Essentially we want the smallest difference between actual outcome and expected outcome. The smaller the difference, the more accurate our function is predicting actual outcomes of games.

The un-optimized Elo function is as follows:

Where:

* is the expected outcome of the home team.
* is the expected outcome of the away team.
* is the pregame rating of the home team.
* is the pregame rating of the away team.
* is the home ice advantage factor.

Where:

* is the post game rating of the home team.
* is the post game rating of the away team.
* is the pregame rating of the home team.
* is the pregame rating of the away team.
* is the update factor, scaling the amount of points a team gains from a win or loss.
* is the goal differential factor.
* is the score differential in relation to the home team.
* is the outcome of a game, 1 for a win, 0.5 for a tie, 0, for a loss.
* is the expected outcome of the home team.
* is the expected outcome of the away team.

Using the above model a general exploration into possible values of , , and could render a lower mean absolute residual. Based off the exploration, is found at roughly 100, at roughly 50, and at roughly 50. It should be noted that was explored first with , and . Next was , which was found at varying weights of and . was then explored with , and . The only reason is held in all three cases, is that if , the function would not be able to update rating as the update portion would always be or .

These explorations provided essential knowledge for the grid search as initial optimization could be found using smaller ranges of values, significantly decreasing the run time of each optimization. This background knowledge cut run time of one optimization dawn from 16.3 days to roughly 30 minutes as each optimization grid search used 1000 combinations of the three variables instead of 4.5 million.

Using this method, was found to be optimized at 37.22222, optimized at 53.33333, and optimized at 40.55556. This provided a mean absolute residual of 0.3515844. This looked very promising at first glance. However, upon deeper inspection, two major concerns were found. First, the bottom four teams had end of season ratings in the negatives. Second, there were many residuals of 1, -1, and 0. This raised the most concern as expected outcome is bounded by 0 < expected outcome < 1, meaning we should never see residuals of exactly 1 or 0, unless the predicted outcome is exactly 1 or 0. Upon deeper inspection, it was found that there were many games were expected outcome was to the effect of 0.9 repeating or 1e-20. While technically within our bounds, are highly unrealistic due to the incredibly high and low probabilities. A good model that predicts win probability for college hockey should never give a pregame win probability of nearly 100% or 0%. Something needed to be changed with the model. Goal differential was looked at first.

Upon inspecting and researching different ways to incorporate goal differential in an Elo function, FiveThirtyEight was looked at again. In their NHL model, goal differential, , was incorporated as shown below:

This approach is much more stable for a few reasons. First, a logarithmic function levels out at the upper a lower bounds. This mimics real life as there is much of a difference in blowout games. A 7-0 game and a 12-0 game are both considered relatively equal games as they’re both blowouts. This stops team from gaining huge amounts of points by really blowing out a team well after it is already “blown out”. The function in addition to downplaying the effects of extremes, also contains all the ratings in the positive ratings. This stabilizes the ratings so we don’t see extreme jumps in rating update from an upset win, and won’t tank a team for being upset.

The new update function now looks as follows:

0.8048 is added into the factor to account for games where since and is undefined. This means that any game where is equal to .

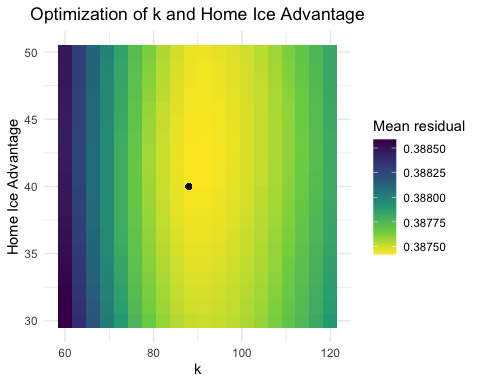
Now that goal factor is optimized, all that was left was to rerun the grid search to optimize and . Using the grid search method, is optimized at 88 and optimized at 40, which gives a mean absolute residual of 0.3876531. Whilst this may not have a mean absolute residual lower than the first model, all teams maintain ratings above zero and keep realistic predicted probabilities.

## 5 Results

Results from this optimization show that , , and the calculation to utilize .

Below is the grid search for and . Multiple grid searches were conducted based on the explorations until the sequence was narrowed to a range of 10. The plot below shows a large range for both and to aid in mapping visual.

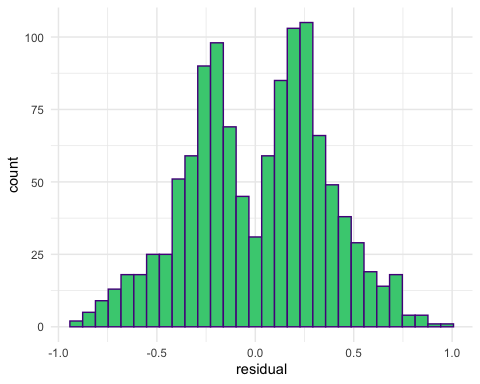
library(furrr)  
library(progressr)  
library(scico)  
plan(multisession)  
handlers("progress")  
options(progressr.enable = TRUE)  
  
grid\_100 = expand.grid(k = seq(60, 120, length.out = 20), home\_ice = seq(30, 50, length.out = 20), d = seq(0, 100, length.out = 1))  
  
mean\_residuals\_100 = with\_progress({future\_pmap\_dbl(grid\_100, \ (k, home\_ice, d) update\_rankings\_residuals(season = schedule\_reg, end\_date = "2025-03-25", ratings = rankings2324, k = k, home\_ice = home\_ice, d = d), .progress = TRUE)})  
  
residual\_100\_df <- grid\_100 |> mutate(mean\_residual = mean\_residuals\_100)  
  
ggplot(data = residual\_100\_df, aes(x = k,  
 y = home\_ice)) +  
 geom\_tile(aes(fill = mean\_residual)) +  
 geom\_point(aes(x = 88, y = 40), color = "black", fill = "black") +  
 scale\_fill\_viridis\_c(option = "D",   
 oob = scales::squish,   
 name = "Mean residual",  
 direction = -1) +  
 labs(x = "k",  
 y = "Home Ice Advantage",  
 title = "Optimization of k and Home Ice Advantage") +  
 theme\_minimal() +  
 theme(legend.position = "right",  
 plot.title = element\_text(hjust = 0.5))



The heat map showing the grid search to , , and . The factors are all optimized to lower the mean absolute value of the difference between expected outcome and actual outcome. In this case, the optimized values were able to lower the mean absolute value of the residuals to 0.3874257. was held at . , .

After finding the optimized values for and using , the entire season was run using the update\_rankings\_residuals() function. This function allowed us to look at residuals for every game in the 2024-2025 season. To see how the function was behaving with respects to home teams and away teams, a plot of the residuals was looked at instead of mean residuals, which take away the ability to investigate how the function deals with home and away teams. A plot of the residuals of the 2024-2025 season is shown below.

apr15 = apr\_15\_ranking |> bind\_rows()  
  
schedule\_elo = schedule |>  
 mutate(home\_elo = NA) |>  
 mutate(away\_elo = NA)  
  
schedule\_apr15 = left\_join(schedule\_elo, apr15,   
 by = join\_by(date == date, home\_team == Team)) |>  
 mutate(home\_elo = rating) |>  
 select(-rating)  
  
merged\_sched\_apr15 = left\_join(schedule\_apr15, apr15,  
 by = join\_by(date == date, away\_team == Team)) |>  
 mutate(away\_elo = rating) |>  
 select(-rating)  
  
schedule\_full\_apr15 = merged\_sched\_apr15 |>  
 mutate(outcome\_away = abs(outcome - 1)) |>   
 ## Calculating expected outcome variable for home and away team  
 mutate(exp\_home = 1/(1 + 10^((away\_elo - home\_elo)/400))) |>  
 mutate(exp\_away = 1/(1 + 10^((home\_elo - away\_elo)/400))) |>  
 ## Using expected outcome variable to generate new Elo ratings based on actual outcome and expected outcome  
 mutate(elo\_new\_home = home\_elo + 100 \* (outcome - exp\_home)) |>  
 mutate(elo\_new\_away = away\_elo + 100 \* (outcome\_away - exp\_away))  
  
## Making a residual column  
schedule\_full\_apr15 <- schedule\_full\_apr15 |>  
 mutate(residual = outcome - exp\_home) |>  
 mutate(abs\_residual = abs(residual))  
  
ggplot(data = schedule\_full\_apr15, aes(x = residual)) +  
 geom\_histogram(color = "purple4", fill = "seagreen3") +  
 theme\_minimal()



The plot of residuals shows all the differences between expected outcome and actual outcome. The positive residuals past 0.5 indicates the the model is predicting a home loss when actual result is a home win. Negative residuals beyond -0.5 indicates that the model predicted a home win with an observed home loss. We see that the model has large amounts of residuals around the which means that the expected outcome is still within the range of the actual outcome. For example, actual outcome is 0, and expected is 0.25, the model still predicted a loss. The only grey area is around ties, since ties are 0.5, a difference could be construed as tie or a win/loss.

After looking into the residuals, we wanted to test just how accurate the predicted win probabilities were. To do this, we took the entire season and binned each game by expected value in increments of 0.1. We then plotted the expected values against the proportion of games within the binned expectation that resulted in a home win. We expect a linear relation: . The expected relation is showing that of all teams with a certain predicted win proportion, the proportion of times those teams win should be equal to that predicted win proportion. Lastly, each binned expected value is weighted by the amount of games in each bin so the linear model isn’t skewed by bins with minimal games. The plot is shown below.

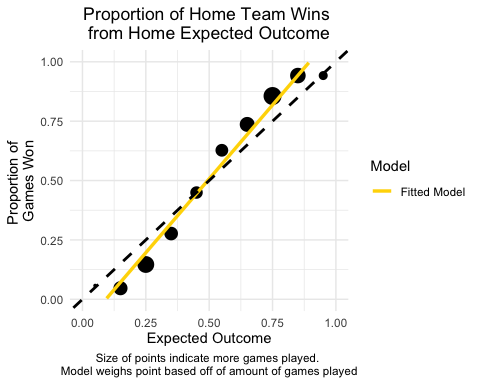
schedule\_full\_apr15 |> summarise(mean\_resid = mean(abs\_residual, na.rm = TRUE))

# A tibble: 1 × 1  
 mean\_resid  
 <dbl>  
1 0.294

prop\_wins15 <- schedule\_full\_apr15 |>  
 mutate(binned\_exp = floor(exp\_home / 0.1) \* 0.1 + 0.05) |>  
 group\_by(binned\_exp) |>  
 summarise(win\_prop = mean(outcome, na.rm = TRUE),   
 totalgames = n()) |>   
 filter(!is.na(binned\_exp))  
  
modgdha = lm(win\_prop ~ binned\_exp, data = prop\_wins15, weights = totalgames)  
summary(modgdha)

Call:  
lm(formula = win\_prop ~ binned\_exp, data = prop\_wins15, weights = totalgames)  
  
Weighted Residuals:  
 Min 1Q Median 3Q Max   
-1.03772 -0.41495 0.01982 0.49843 0.79635   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) -0.11495 0.04086 -2.813 0.0227 \*   
binned\_exp 1.24411 0.06996 17.782 1.02e-07 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 0.6344 on 8 degrees of freedom  
Multiple R-squared: 0.9753, Adjusted R-squared: 0.9722   
F-statistic: 316.2 on 1 and 8 DF, p-value: 1.024e-07

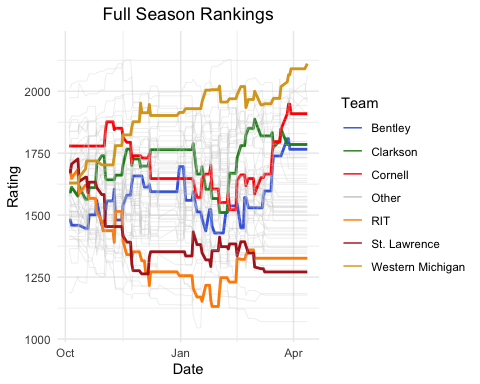
ggplot(data = prop\_wins15, aes(x = binned\_exp,   
 y = win\_prop,  
 size = totalgames)) +  
 geom\_point(color = "black", shape = 16) +  
 geom\_smooth(aes(color = "Fitted Model",  
 weight = totalgames), method = "lm", se = FALSE, size = 1.2) +  
 geom\_abline(data = data.frame(1),   
 aes(color = "Expected Linear Model",  
 linetype = "Expected Linear Model"),   
 slope = 1, intercept = 0, linetype = 2, size = 1) +  
 scale\_color\_manual(name = "Model",  
 values = c("Fitted Model" = "gold",   
 "Expected Linear Model" = "black")) +  
 labs(title = "Proportion of Home Team Wins \nfrom Home Expected Outcome",  
 x = "Expected Outcome",  
 y = "Proportion of \nGames Won",  
 caption = "Size of points indicate more games played. \nModel weighs point based off of amount of games played") +  
 guides(size = "none") +  
 theme\_minimal() +  
 theme(legend.position = "right",  
 plot.title = element\_text(hjust = 0.5),  
 plot.caption = element\_text(hjust = 0.5)) +  
 xlim(c(0, 1)) +  
 ylim(c(0, 1))



This plot bins every game played in the 2024-2025 season by every 0.1. Then uses the amount of games played in each bin to weigh each point. The y-axis is the proportion of wins for each binned proportion. We expect a slope of 1. This means that for example, at expected outcome of 0.9 we expect 90% of the games are won. In this case the fitted model is . This further shows the accuracy of the model.

Finally, we looked at the final rankings using the update\_rankings\_iter\_gd\_ha() function. This function runs the entire season and gathers ratings for each day and stores the rankings in a list. To see what the final rankings were, we pulled from the final day of the 2024-2025 post-season. To investigate the accuracy of the model, the rankings were compared to conference standings and the results of the NCAA Men’s Division 1 Hockey National Championship Tournament. Final rankings are shown below with plot showing the progress of each team throughout the entire season.

apr15 = apr\_15\_ranking |> bind\_rows()  
  
apr15\_lagged = apr15 |> group\_by(date) |>  
 summarise(last\_date = last(date)) |>  
 mutate(lag\_date = lag(last\_date)) |>  
 select(-last\_date)  
  
apr15\_full = left\_join(apr15, apr15\_lagged, join\_by(date == lag\_date)) |>  
 select(-date) |>  
 rename(date = date.y)  
  
highlight = c("St. Lawrence", "Western Michigan", "Clarkson", "Bentley", "RIT", "Cornell")  
  
highlighted\_color = c(  
 "St. Lawrence" = "firebrick",  
 "Western Michigan" = "goldenrod",  
 "Clarkson" = "forestgreen",  
 "Bentley" = "royalblue",  
 "RIT" = "darkorange",  
 "Cornell" = "red")  
  
apr15\_color = apr15\_full |> mutate(highlight = if\_else(Team %in% highlight, Team, "Other"))  
  
ggplot(data = apr15\_color, aes(x = date,  
 y = rating,  
 group = Team)) +  
 geom\_line(aes(color = highlight,   
 alpha = highlight,   
 linewidth = highlight)) +  
 scale\_color\_manual(values = c("Other" = "grey80",   
 highlighted\_color),  
 name = "Team") +  
 scale\_alpha\_manual(values = c("Other" = 0.5,   
 setNames(rep(1, length(highlighted\_color)),  
 names(highlighted\_color))),  
 guide = "none") +  
 scale\_linewidth\_manual(values = c("Other" = 0.2,   
 setNames(rep(1, length(highlighted\_color)),  
 names(highlighted\_color))),  
 guide = "none") +  
 theme\_minimal() +  
 labs(color = "Team",  
 title = "Full Season Rankings",  
 x = "Date",  
 y = "Rating") +  
 theme(legend.position = "right",  
 plot.title = element\_text(hjust = 0.5))



print(n = 64, apr\_15\_ranking[[110]] |> arrange(desc(rating))  
 |> select(-date))

# A tibble: 64 × 2  
 Team rating  
 <chr> <dbl>  
 1 Western Michigan 2186.  
 2 Boston University 1990.  
 3 Connecticut 1970.  
 4 Penn State 1964.  
 5 Denver 1954.  
 6 Michigan State 1943.  
 7 Boston College 1915.  
 8 Cornell 1909.  
 9 Minnesota State 1902.  
10 Maine 1890.  
11 Massachusetts 1888.  
12 North Dakota 1829.  
13 Minnesota 1798.  
14 Quinnipiac 1796.  
15 Arizona State 1786.  
16 Clarkson 1785.  
17 Omaha 1774.  
18 Bentley 1766.  
19 St. Thomas 1744.  
20 Providence 1734.  
21 Michigan 1733.  
22 Northeastern 1730.  
23 Dartmouth 1706.  
24 Notre Dame 1697.  
25 Ohio State 1687.  
26 Long Island 1662.  
27 New Hampshire 1648.  
28 Holy Cross 1638.  
29 Harvard 1632.  
30 Bowling Green 1593.  
31 Colorado College 1575.  
32 Bemidji State 1574.  
33 Colgate 1563.  
34 Minnesota-Duluth 1558.  
35 Wisconsin 1549.  
36 Alaska 1538.  
37 Vermont 1536.  
38 Augustana 1533.  
39 St. Cloud State 1533.  
40 Brown 1530.  
41 Sacred Heart 1518.  
42 Merrimack 1514.  
43 Mass.-Lowell 1511.  
44 Union 1500.  
45 Princeton 1479.  
46 Army 1477.  
47 Lake Superior 1449.  
48 Ferris State 1444.  
49 Air Force 1435.  
50 Michigan Tech 1424.  
51 Lindenwood 1420.  
52 American Int'l 1407.  
53 Stonehill 1372.  
54 Niagara 1369.  
55 Rensselaer 1359.  
56 Canisius 1352.  
57 Alaska-Anchorage 1345.  
58 RIT 1326.  
59 Northern Michigan 1298.  
60 Yale 1281.  
61 St. Lawrence 1271.  
62 Robert Morris 1183.  
63 Miami 1177.  
64 Mercyhurst 1070.

In these final season rankings, our Elo system ranked Western Michigan as the top team, with Boston University second. In fact, of the top 4 teams in our model, 3 of them made the Frozen Four, with Denver finishing in 5th. Western Michigan ultimately ended up winning the National Championship.

The progress chart shows some interesting patterns with some teams, especially Western Michigan. They started the season middle of the pack, but with continuous strong wins, they were able to reach 2nd place before the last quarter of the season. Other teams are highlighted based on relation to St. Lawrence, such as Cornell (ECAC champions) and Clarkson University (St. Lawrence’s biggest rival). Other schools were chosen for personal reasons, Bentley was chosen as they won their conference, Atlantic Hockey America for the first time in the school’s history and is also the school of my brother. RIT was also chosen due to it being my hometown. One thing to note is how almost all of these schools started in the middle of the rankings, but all finished in very different spots, showing just how much a team’s strength can change season to season. This further shows the usefulness of adjusting team ratings prior to the season starting.

Finally, with the model optimized and set let’s run an example, using the last game of the year, the NCAA Championship game between Western Michigan and Boston University.

schedule\_full\_apr15 |> slice(1153) |>  
 select(c(1, 3, 4, 5, 6, 9, 10, 11, 12))

# A tibble: 1 × 9  
 date away\_team away\_score home\_team home\_score score\_diff outcome  
 <date> <chr> <dbl> <chr> <dbl> <dbl> <dbl>  
1 2025-04-12 Western Michigan 6 Boston U… 2 -4 0  
# ℹ 2 more variables: home\_elo <dbl>, away\_elo <dbl>

Western Michigan:

From this calculation, Western Michigan was a strong favorite to win this game, which they won handily, because of this their rating increased by 44 points to 2230.11. Now, let’s see how Boston University was affected.

Boston University:

We see that again the change in rating is 44 with BU decreasing by 44 to 1945.89. This again, is a small decrease since BU were underdogs, and proceeded to lose by a large amount.

## 6 Conclusion

In order to make a better ratings system in NCAA Men’s Division 1 Ice Hockey, an Elo style system was created. Using goal differential, an update factor, and home ice advantage, an Elo model was optimized using a grid search method to get the mean absolute value of game residuals down to 0.388. The model successfully ranked the four teams to make the Frozen Four in the top five and successfully ranked the national champions, Western Michigan, in the top spot for the end of 2024-2025 season rankings.

This model allows teams to be ranked quantitatively and use factors such as strength of schedule and quality of win/loss to accurately scale the effect of each win or loss. In the future, I would like to try to add more parameters in to the model to see how big of an effect things like, OT and neutral site have on expected outcome. Mostly I would like to break down score differential into goals for and goals against to see if having good defense is more important than good offense and vice versa. My biggest regret is not being able to optimize the score differential parameter myself, and if given more time, I would do this.

## 7 Code Appendix

Libraries:

library(elo)  
library(dplyr)  
library(tidyverse)  
library(cowplot)  
library(ggrepel)  
library(lubridate)  
library(rvest)  
library(here)  
library(forcats)  
library(progressr)  
library(furrr)  
library(vctrs)  
library(purrr)

Scraping function:

##Function to load in schedule  
scrape\_men <- function(season = "20232024"){  
 ## URL for schedule data frame  
 url\_hockey <-paste("https://www.collegehockeynews.com/schedules/?season=", season, sep = "")  
 ## Selecting which schedule table to grab  
 tab\_hockey <- read\_html(url\_hockey) |>  
 html\_nodes("table")  
   
 ## The website likes to switch which table it uses. If function doesn't work try changing which table number you select  
 stats\_dirty <- tab\_hockey[[1]] |> html\_table()  
   
 ## Creating regex for date, and conference to make date and conference columns in dataframe  
 regex\_date <- "October|November|December|January|February|March|April"  
 regex\_conference <- "Atlantic Hockey|Big Ten|CCHA|ECAC|Hockey East|NCHC|Ind|Exhibition|Non-Conference"  
 ## Combining regexs with original table so that the original scraped dataframe has date and conferene as variables  
 stats\_regex <- stats\_dirty |> mutate(date = if\_else(str\_detect(X1, regex\_date),  
 true = X1, false = NA\_character\_),  
 conference = if\_else(str\_detect(X1, regex\_conference),   
 true = X1, false = NA\_character\_))  
   
 ## Filling in respective dates and conferences  
 stats\_filled <-stats\_regex |> fill(date, .direction = "down") |>  
 ## Selecting date and congference so they show up as X1 and X2 in the dataframe  
 fill(conference, .direction = "down") |> select(date, conference, everything())  
 ##filtering out anywhere that a conference value is undetected (Game category, not an actual game played)  
 stats\_filled\_cleaner <- stats\_filled |> filter(!str\_detect(X1, regex\_date) &  
 !str\_detect(X1, regex\_conference))  
 print(head(stats\_filled\_cleaner))  
   
 ## Dataframe is now in a format that is able to be worked on. Now creating specific variables that we want to look at  
 ## Selecting first 8 columns  
 schedule\_new <- stats\_filled\_cleaner |> select(date, conference, X1, X2, X3, X4, X5, X6) |>  
 ## Taking out first two rows (no data in them). Renaming columns to match what their variable is.  
 slice(-1 , -2) |> rename(game\_type = conference, away\_team = X1, away\_score = X2, location\_marker = X3, home\_team = X4, home\_score = X5, overtime = X6) |>   
 ## Take out the day of the week in our date columns as we don't need to know if a game was played on a Monday per-se.   
 separate(col = date, into = c("weekday", "dm", "y"),  
 sep = ", ") |>   
 unite("new\_date", c(dm, y),  
 sep = " ") |>  
 select(-weekday) |>  
 ##making date column into a <date> variable  
 mutate(date = mdy(new\_date)) |>  
 ## Taking out the <chr> date variable  
 select(-new\_date) |>   
 select(date, everything()) |>  
 ## Filtering out where there is no away team since that means no game was played  
 filter(away\_team != "") |>  
 ## Filtering out exhibition games since we aren't looking at exhibition games  
 filter(game\_type != "Exhibition") |>  
 ## Turning scores from <chr> to <dbl> variables  
 mutate(away\_score = as.double(away\_score)) |>  
 mutate(home\_score = as.double(home\_score)) |>  
 ## creating a variable to indicate if a game was played at a neutral site  
 mutate(neutral\_site = case\_when(location\_marker == "vs." ~ 1,  
 location\_marker == "at" ~ 0)) |>  
 ## Making the neutral\_site variable as <lgl>   
 mutate(neutral\_site = as.logical(neutral\_site)) |>  
 ## taking out location\_marker  
 select(-location\_marker) |>  
 ## Making a logical overtime variable. Note we are not differentiating between OT and 2OT  
 mutate(overtime = case\_when(overtime == "" ~ 0,  
 overtime == "ot" ~ 1,  
 overtime == "2ot" ~ 1)) |>  
 mutate(overtime = as.logical(overtime)) |>  
 ##Filtering out NA "overtime" values as this indicates no game played, since overtime will either be TRUE or FALSE  
 filter(!is.na(overtime))|>  
 ## Creating a score differential variable to indicate a win, loss, or tie for the home team. If we know the outcome for the home team, we know the outcome for the away team.  
 mutate(score\_diff = home\_score - away\_score) |>  
 ## making an outcome variable for home team so ties get input as 0.5, wins get input as 1, and loss get input as 0.  
 mutate(outcome =   
 case\_when(score\_diff == 0 ~ "0.5",  
 score\_diff > 0 ~ "1",  
 score\_diff < 0 ~ "0")) |>  
 ## turning score\_diff from <chr> to <dbl>  
 mutate(outcome = as.double(outcome)) |>  
 ## Filtering out games where D1 team played against D3 teams as these are exhibition as well  
 filter(game\_type != "Non-Conference v. D3")  
   
 ## Tidy schedule is returned  
 return(schedule\_new)  
}  
  
##Load in Schedule  
schedule <- scrape\_men("20242025")  
  
## Load in my arbitrary initial elo ranking  
X22Rankings <- read\_csv(here("datasets\_dataframes/22Rankings.csv"))

Function to do single day ratings:

##Function to update rankings  
##rating is the variable, ratings is the df.  
update\_rankings <- function(season, game\_date, ratings, k = 20){  
 ## Filters schedule to a specific date  
 elo\_ratings\_update <- season |> filter(date == game\_date) |>  
 ## Joins the Elo ratings from our rating file to the schedule file. Puts updated ratings in the schedule  
 left\_join(ratings, by = join\_by(away\_team == Team)) |>  
 rename(away\_elo = rating) |>   
 ## Updates ratings for home team in the schedule file  
 left\_join(ratings, by = join\_by(home\_team == Team)) |>  
 rename(home\_elo = rating) |>  
 ## Creating an away team outcome variable. Opposite of home team or same if tie.  
 mutate(outcome\_away = abs(outcome - 1)) |>   
 ## Calculating expected outcome variable for home and away team  
 mutate(exp\_home = 1/(1 + 10^((away\_elo - home\_elo)/400))) |>  
 mutate(exp\_away = 1/(1 + 10^((home\_elo - away\_elo)/400))) |>  
 ## Using expected outcome variable to generate new Elo ratings based on actual outcome and expected outcome  
 mutate(elo\_new\_home = home\_elo + k\*(outcome - exp\_home)) |>  
 mutate(elo\_new\_away = away\_elo + k\*(outcome\_away - exp\_away)) |>  
 rename(date\_update\_rankings = date)  
  
 ranked\_home <- left\_join(ratings, elo\_ratings\_update, by = join\_by(Team == home\_team)) |>  
 relocate(elo\_new\_home) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_home),  
 true = elo\_new\_home,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings)  
   
 ratings\_new <- left\_join(ranked\_home, elo\_ratings\_update, by = join\_by(Team == away\_team)) |>  
 relocate(elo\_new\_away) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_away),  
 true = elo\_new\_away,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings.x) |>  
 mutate(date\_update\_rankings.x = game\_date) |>  
 rename(date = date\_update\_rankings.x)  
   
 return(ratings\_new)  
}

Creating a vector of dates to be iterated to automate ratings:

dates\_vec <- unique(schedule$date)  
## defining what new\_rankings is going to be  
new\_rankings = rankings  
## Creating a for loop with the update\_rankings function to update rankings up to a specified date  
for (i in dates\_vec) {  
 new\_rankings <- update\_rankings(season = schedule, game\_date = i, ratings = new\_rankings, k = 100)  
   
}

Function to iterate through dates to automate ratings

update\_rankings\_iter <- function(season, end\_date, ratings, k){  
   
 new\_rankings <- list()  
   
 season\_cut <- season |>   
 ## Filter by a specified end date to deal with NA values (gmaes that have yet to be played)  
 filter(date <= ymd(end\_date))  
 ## Creating a vector for unique dates in a season  
 dates\_vector <- unique(season\_cut$date)  
 ## Defining our rankings within the function  
 new\_rankings[[1]] <- ratings  
 ## Creating a for loop for the function to generate new ratings with the updtae\_rankings function  
 for (i in 1:length(dates\_vector)) {  
 new\_rankings[[i + 1]] <- update\_rankings(season = season\_cut, game\_date = dates\_vector[i], ratings = new\_rankings[[i]], k = k)  
 }  
 return(new\_rankings)  
}

Function to update single day ratings with all three parameters, k, score differential, home ice advantage:

update\_rankings\_gd\_ha <- function(season, game\_date, ratings, k = 100, home\_ice = 50, d = 0.5){  
 ## Filters schedule to a specific date  
 elo\_ratings\_update <- season |> filter(date == game\_date) |>  
 ## Joins the Elo ratings from our rating file to the schedule file. Puts updated ratings in the schedule  
 left\_join(ratings, by = join\_by(away\_team == Team)) |>  
 rename(away\_elo = rating) |>   
 ## Updates ratings for home team in the schedule file  
 left\_join(ratings, by = join\_by(home\_team == Team)) |>  
 rename(home\_elo = rating) |>  
 ## Creating an away team outcome variable. Opposite of home team or same if tie.  
 mutate(outcome\_away = abs(outcome - 1)) |>   
 ## Calculating expected outcome variable for home and away team  
 mutate(exp\_home = 1/(1 + 10^((away\_elo - (home\_elo + home\_ice))/400))) |>  
 mutate(exp\_away = 1/(1 + 10^(((home\_elo + home\_ice) - away\_elo)/400))) |>  
 ## Using expected outcome variable to generate new Elo ratings based on actual outcome and expected outcome  
 ## fivethirtyeights parameters  
 ## 0.6686 \* log(abs(score\_diff)) + 0.8048  
 mutate(score\_mult = if\_else(score\_diff == 0,  
 true = 0.8048,  
 false = 0.6686 \* log(abs(score\_diff)) + 0.8048)) |>  
 mutate(elo\_new\_home = home\_elo + k \* score\_mult \* (outcome - exp\_home)) |>  
 mutate(elo\_new\_away = away\_elo + k \* score\_mult \* (outcome\_away - exp\_away)) |>  
 rename(date\_update\_rankings = date)  
  
 ranked\_home\_gd\_ha <- left\_join(ratings, elo\_ratings\_update, by = join\_by(Team == home\_team)) |>  
 relocate(elo\_new\_home) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_home),  
 true = elo\_new\_home,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings)  
   
 ratings\_new\_gd\_ha <- left\_join(ranked\_home\_gd\_ha, elo\_ratings\_update, by = join\_by(Team == away\_team)) |>  
 relocate(elo\_new\_away) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_away),  
 true = elo\_new\_away,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings.x) |>  
 mutate(date\_update\_rankings.x = game\_date) |>  
 rename(date = date\_update\_rankings.x)  
   
 return(ratings\_new\_gd\_ha)  
}

Function to iterate through entire season automatically:

rankings = X22Rankings  
  
##Function to update rankings  
##rating is the variable, ratings is the df.  
update\_rankings\_gd\_ha <- function(season, game\_date, ratings, k = 100, home\_ice = 50, d = 0.5){  
 ## Filters schedule to a specific date  
 elo\_ratings\_update <- season |> filter(date == game\_date) |>  
 ## Joins the Elo ratings from our rating file to the schedule file. Puts updated ratings in the schedule  
 left\_join(ratings, by = join\_by(away\_team == Team)) |>  
 rename(away\_elo = rating) |>   
 ## Updates ratings for home team in the schedule file  
 left\_join(ratings, by = join\_by(home\_team == Team)) |>  
 rename(home\_elo = rating) |>  
 ## Creating an away team outcome variable. Opposite of home team or same if tie.  
 mutate(outcome\_away = abs(outcome - 1)) |>   
 ## Calculating expected outcome variable for home and away team  
 mutate(exp\_home = 1/(1 + 10^((away\_elo - (home\_elo + home\_ice))/400))) |>  
 mutate(exp\_away = 1/(1 + 10^(((home\_elo + home\_ice) - away\_elo)/400))) |>  
 ## Using expected outcome variable to generate new Elo ratings based on actual outcome and expected outcome  
 ## 0.6686 \* log(abs(score\_diff)) + 0.8048  
 mutate(score\_mult = if\_else(score\_diff == 0,  
 true = 0.8048,  
 false = 0.6686 \* log(abs(score\_diff)) + 0.8048)) |>  
 mutate(elo\_new\_home = home\_elo + k \* score\_mult \* (outcome - exp\_home)) |>  
 mutate(elo\_new\_away = away\_elo + k \* score\_mult \* (outcome\_away - exp\_away)) |>  
 ##mutate(elo\_new\_home = home\_elo + k\*(outcome - exp\_home) + d \* score\_diff) |>  
 mutate(elo\_new\_home = if\_else(elo\_new\_home < 100,  
 true = 100,  
 false = elo\_new\_home)) |>  
 ##mutate(elo\_new\_away = away\_elo + k\*(outcome\_away - exp\_away) + d \* -1 \* (score\_diff)) |>  
 mutate(elo\_new\_away = if\_else(elo\_new\_away < 100,  
 true = 100,  
 false = elo\_new\_away)) |>  
 rename(date\_update\_rankings = date)  
   
 ## Find out which is the right date, select() and relocate(), \*\*DO THIS FIRST\*\*: try renaming date in one of the df to make less confusing (at start).  
 ranked\_home\_gd\_ha <- left\_join(ratings, elo\_ratings\_update, by = join\_by(Team == home\_team)) |>  
 relocate(elo\_new\_home) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_home),  
 true = elo\_new\_home,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings)  
   
 ratings\_new\_gd\_ha <- left\_join(ranked\_home\_gd\_ha, elo\_ratings\_update, by = join\_by(Team == away\_team)) |>  
 relocate(elo\_new\_away) |>  
 mutate(rating = if\_else(!is.na(elo\_new\_away),  
 true = elo\_new\_away,  
 false = rating)) |>  
 select(Team, rating, date\_update\_rankings.x) |>  
 mutate(date\_update\_rankings.x = game\_date) |>  
 rename(date = date\_update\_rankings.x)  
   
 return(ratings\_new\_gd\_ha)  
}

Creating a vector of dates to use again to automate:

dates\_vec <- unique(schedule$date)  
## defining what new\_rankings is going to be  
new\_rankings = rankings  
## Creating a for loop with the update\_rankings function to update rankings up to a specified date  
for (i in dates\_vec) {  
 new\_rankings <- update\_rankings\_gd\_ha(season = schedule, game\_date = i, ratings = new\_rankings, k = 100, home\_ice = 50, d = 0.5)  
   
}

Function to return a completed set of rankings:

Function to run ratings and return the mean residual

update\_rankings\_residuals = function(season, end\_date, ratings, k, home\_ice, d){   
   
 new\_rankings = update\_rankings\_iter\_gd\_ha(season = season, end\_date = end\_date, ratings = ratings, k = k, home\_ice = home\_ice, d = d)  
   
 full\_rankings = new\_rankings |> bind\_rows()  
   
 lagged\_dates = full\_rankings |> group\_by(date) |>  
 summarise(last\_date = last(date)) |>  
 mutate(lag\_date = lag(last\_date)) |>  
 select(-last\_date)  
   
 lagged\_rankings <- left\_join(full\_rankings, lagged\_dates, join\_by(date == lag\_date)) |>  
 select(-date) |>  
 rename(date = date.y)  
   
 season = season |>  
 mutate(home\_elo = NA) |>  
 mutate(away\_elo = NA)  
   
 merged\_season\_home = left\_join(season, lagged\_rankings,   
 by = join\_by(date == date, home\_team == Team)) |>  
 mutate(home\_elo = rating) |>  
 select(-rating)  
   
 merged\_season = left\_join(merged\_season\_home, lagged\_rankings,  
 by = join\_by(date == date, away\_team == Team)) |>  
 mutate(away\_elo = rating) |>  
 select(-rating)  
   
 full\_season = merged\_season |>  
 mutate(outcome\_away = abs(outcome - 1)) |>   
 ## Calculating expected outcome variable for home and away team  
 mutate(exp\_home = 1/(1 + 10^((away\_elo - (home\_elo + home\_ice))/400))) |>  
 mutate(exp\_away = 1/(1 + 10^(((home\_elo + home\_ice) - away\_elo)/400))) |>  
 ## Using expected outcome variable to generate new Elo ratings based on actual outcome and expected outcome  
 ##   
 mutate(score\_mult = if\_else(score\_diff == 0,  
 true = 0.8048,  
 false = 0.6686 \* log(abs(score\_diff)) + 0.8048)) |>  
 mutate(elo\_new\_home = home\_elo + k \* score\_mult \* (outcome - exp\_home)) |>  
 mutate(elo\_new\_away = away\_elo + k \* score\_mult \* (outcome\_away - exp\_away)) |>  
 mutate(elo\_new\_home = if\_else(elo\_new\_home < 100,  
 true = 100,  
 false = elo\_new\_home)) |>  
 mutate(elo\_new\_away = if\_else(elo\_new\_away < 100,  
 true = 100,  
 false = elo\_new\_away)) |>  
 mutate(residual = outcome - exp\_home) |>  
 mutate(abs\_residual = abs(residual))  
   
 mean\_residual = full\_season |> summarise(avg = mean(abs\_residual, na.rm = TRUE))  
   
 return(pull(mean\_residual))  
}

Additional data frames:

Optimization process:

Optimized ratings:

apr\_15\_ranking = update\_rankings\_iter\_gd\_ha(schedule, "2025-04-15",  
 rankings2324, optimal$k, optimal$home\_ice,  
 0)

Data frame for mapping plot:

plan(multisession)  
handlers("progress")  
options(progressr.enable = TRUE)  
  
grid\_100 = expand.grid(k = seq(60, 120, length.out = 20), home\_ice = seq(30, 50, length.out = 20), d = seq(0, 100, length.out = 1))  
  
mean\_residuals\_100 = with\_progress({future\_pmap\_dbl(grid\_100, \ (k, home\_ice, d) update\_rankings\_residuals(season = schedule\_reg, end\_date = "2025-03-25", ratings = rankings2324, k = k, home\_ice = home\_ice, d = d), .progress = TRUE)})  
  
residual\_100\_df <- grid\_100 |> mutate(mean\_residual = mean\_residuals\_100)

## 8 Works Cited

College Hockey News. (2024). *Men’s Division I Hockey Schedule* – 2023–2024 Season. https://www.collegehockeynews.com/schedules/

College Hockey News. (2025). *Men’s Division I Hockey Schedule* – 2024–2025 Season. https://www.collegehockeynews.com/schedules/

**Silver, N.** (n.d.). *How our NHL predictions work*. FiveThirtyEight. <https://fivethirtyeight.com/methodology/how-our-nhl-predictions-work/>