ADVANCED COMPUTATIONAL RATINGS FOR COLLEGE FOOTBALL TEAMS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Computer Science

By

Joel Michael Hensley

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2010

Fargo, North Dakota

// ABSTRACT

Hensley, Joel Michael, M.S., Computer Science, College of Science and Mathematics, North Dakota State University, April 2010. Advanced Computational Ratings for College Football Teams. Major Professor: Dr. Kendall Nygard.

This paper explores the subject of rating systems applied to the world of college football. Current rating system methodologies are examined, and four rating systems are developed and evaluated in a program. The Hensley Rating system is introduced as a new method. The details of each of these systems are discussed, and the results are analyzed and evaluated using data from the college football seasons of 2000 – 2009.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF TABLES vi

LIST OF FIGURES viii

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. LITERATURE REVIEW 3

Least Squares 3

Maximum Likelihood Estimator 4

Game Adjustment Ratings 5

CHAPTER 3. RATING METHODS 8

Database Design 8

Entity Groups 10

Standard Rating 12

Home Field Advantage (HFA) 13

Max Point Differential (MPD) 16

Hensley Rating 17

CHAPTER 4. ANALYSIS 22

CHAPTER 5. CONCLUSION 28

REFERENCES 30

APPENDIX A. TEAM RATINGS FROM 2000 – 2009 32

APPENDIX B. CONFERENCE RATINGS FROM 2000 - 2009 37

APPENDIX C. DIVISION RATINGS FROM 2000 – 2009 43

APPENDIX D. CODE 47

Database Layer Project 47

Database Import Project 68

Rating System Project 79

Rating Evaluator Project 112

LIST OF TABLES

Table Page

Table 1. Sample Schedule 10

Table 2. Home Field Advantage by Year 15

Table 3. Max Point Differential Comparison 17

Table 4. Rating Comparison for 2009 23

Table 5. Correlation against Final AP Poll for 2000 – 2009 24

Table 6. Correlation against Average Computer Rankings for 2000 – 2009 25

Table 7. Number of Upsets for 2000 – 2009 26

Table 8. Average Mistake Value for 2000 – 2009 27

Table 9. 2009 Team Results 32

Table 10. 2008 Team Results 32

Table 11. 2007 Team Results 33

Table 12. 2006 Team Results 33

Table 13. 2005 Team Results 34

Table 14. 2004 Team Results 34

Table 15. 2003 Team Results 35

Table 16. 2002 Team Results 35

Table 17. 2001 Team Results 36

Table 18. 2000 Team Results 36

Table 19. 2009 Conference Results 37

Table 20. 2008 Conference Results 38

Table 21. 2007 Conference Results 38

Table 22. 2006 Conference Results 39

Table 23. 2005 Conference Results 39

Table 24. 2004 Conference Results 40

Table 25. 2003 Conference Results 40

Table 26. 2002 Conference Results 41

Table 27. 2001 Conference Results 41

Table 28. 2000 Conference Results 42

Table 29. 2009 Division Results 43

Table 30. 2008 Division Results 43

Table 31. 2007 Division Results 44

Table 32. 2006 Division Results 44

Table 33. 2005 Division Results 44

Table 34. 2004 Division Results 45

Table 35. 2003 Division Results 45

Table 36. 2002 Division Results 45

Table 37. 2001 Division Results 46

Table 38. 2000 Division Results 46

LIST OF FIGURES

Figure | Page

1. Database UML Diagram 9
2. Sample Team Graph 11
3. Teams Connected By Week 12
4. Level of Victory vs. Winning Score 19

CHAPTER 1. INTRODUCTION

The world of college football undergoes intense controversy year after year in determining who should play for the national title. In 1997, the Associated Press (AP) Poll selected Michigan as the national champion, while the USA Today Coaches Poll selected Nebraska. Therefore, beginning in the 1998 season, the Bowl Championship Series (BCS) was created to ensure that a national championship game was played each season between the #1 and #2 rated teams with the winner crowned as the national champion. As part of the BCS ratings, computer polls were brought in to help bring an unbiased opinion and attempt to objectively measure each team. As of the 2009 season, the six components of the computer rating in the BCS formula were Anderson & Hester, Richard Billingsley, the Colley Matrix, Kenneth Massey, Jeff Sagarin, and Peter Wolfe (BCS Computer Rankings, 2010).

Every computer rating system is different. Some take into account only the winner and loser of each game. Others factor in the final scores. A few even go so far as to include detailed statistics for each game into their rating systems. In the course of this paper, the only information that will be used to compute the ratings will be the final score and the location of the game, such as home, away, or at a neutral site.

It's important to remember that computer ratings are simply a way to classify teams, not name a champion. I strongly believe the only fair method of selecting a national champion is holding a tournament among the top teams. Currently, the BCS uses a two team playoff. However, it is my hope that in the future this playoff can be

expanded to more teams, and if that happens there will be an even greater need for accurate and trusted computer ratings to help determine which teams should be invited. Therefore, the intent of this paper is to provide a new rating system that will help solve this problem.

One of the main drawbacks to rating systems that only use the margin of victory in its calculations is that it doesn't take into account the final score. For instance, a win by 14 points is the same whether the final score is 14 – 0 or 56 – 42. However in my opinion, the 56 – 42 win is less impressive and should be adjusted accordingly during the rating calculations. In games with a high amount of scoring, there is a good chance that both teams were each able to score at will throughout the game. Therefore, in a given rematch, there is a higher probability that the loser may in fact win. The goal of the new rating system will be to accurately reward teams based on the final score and not just the margin of victory.

In Chapter 2, a brief comparison of existing computer rating systems is presented. These existing computer rating systems are organized into three groups based on their algorithms. In Chapter 3, the four rating methods that were created in the program are explored. The Standard, Home Field Advantage, and Max Point Differential rating methods are all based on existing ideas, while the Hensley Rating uses a new algorithm. Chapter 4 then analyzes the four rating methods created, using past data and other existing computer rating systems as benchmarks. Finally, Chapter 5 summarizes the information presented in this paper.

CHAPTER 2. LITERATURE REVIEW

There are many different types of computer rating systems in use today. Each rating system uses general statistics and mathematics principles to produce the final ratings. Today, there are over 100 different rating systems used in college football. Most of the internal details for each rating system are kept secret. However, the general method employed by most rating systems is explained. A few of the more common types of methods are discussed below.

Least Squares

In the least squares approach, the rating is built upon the following equation for a particular game i:

$$y_i = r_a - r_b + e_i$$

where y_i is some observable measure for the game, such as the scoring margin, r_a and r_b are the ratings for team A and team B respectively, and e_i is the error term. The goal of this method is to then minimize $\sum e_i^2$. Once all the observations are recorded, a system of linear equations is created. A common approach is to replace any one row in the matrix with all 1's and a 0 in the rightmost cell. This adds the stipulation that all the ratings sum to 0 (Massey, Statistical Models Applied to the Rating of Sports Teams, 1997).

Other factors such as a home field advantage rating can also be applied to this type of model, making it fairly flexible. Weights can even be given to different games to reflect the different levels of importance, such as valuing conference games higher

than non-conference games (Gill & Keating, 2009). Yet another customization to this method is to account for offensive and defensive ratings (Govan, Langville, & Meyer, 2009). However, one of the most significant components of this method is what to use for the value y_i, also known as the game outcome measure. If the margin of victory is to be ignored, then this value is simply a 1 for a win and a -1 for a loss. Other methods simply truncate the margin of victory to some predefined value, such as 14 or 28 (West & Lamsal, 2008). In this way, teams are not rewarded for running up the score against an inferior opponent.

One thing to note about these methods is that they require all the teams to be connected in order to solve the system of linear equations. Therefore, early on in the season, these methods will produce a number of independent ratings. Teams from different sets of ratings cannot be compared, so this method requires a rather complete graph to produce useful results.

Since a team's rating in the least squares model is derived specifically from the ratings of all the other teams in the league, the strength of schedule is entirely incorporated into the rating. The term infinite strength of schedule is often used when referring to these types of systems.

This type of rating system is used in the Colley Matrix (Colley, 2002).

Maximum Likelihood Estimator

The maximum likelihood estimator is a technique that uses the probability of a team to win a game against another opponent to compute its ratings. For instance, suppose the probability of team A to beat team B is

$$\frac{r_A}{(r_A + r_B)}$$

where r_A and r_B are the ratings for team A and B respectively. This method then uses a system of nonlinear equations and attempts to obtain ratings for the teams that will generate the highest probability, or maximum likelihood. This is generally done iteratively until it converges on a final rating.

This type of rating system is used by Kenneth Massey (Massey, Massey Ratings Description) and Peter Wolfe (Wolfe, About These Ratings).

Game Adjustment Ratings

Instead of trying to solve a system of equations to determine ratings, a number of rating systems rather perform adjustments after each game is played. In this method, all teams are initially assigned some starting rating value. If previous seasons are ignored, then all teams will start at the same rating each season. Otherwise, the teams may start with the rating at the end of the previous season. Then as each game is played, the teams involved have their ratings adjusted based on the result of the game. The goal of these types of methods is to reward teams for wins and penalize teams for losses (Trono, 2007). The difference between the methods comes down to how much or how little the changes should be.

One famous example of this methodology is the Elo rating method created by Arpad Elo (Elo, 1978). It is most commonly used to rate chess players, but Jeff Sagarin has applied this method to college football (Sagarin, 2010). In this method, an equation is used to determine the expected outcome for each game between two

teams. For instance, suppose team A is expected to beat team B by 10 points. However once the game is played, team A only wins by 3 points. Therefore, the rating of team A would be lowered since it did not meet expectations. However, the amount of adjustment is directly based on the expectations about the outcome before the game is played.

For instance, if team A has a rating of 1000 and team B has a rating of 1200, then a win by team A will result in a higher increase in rating than if team B had won. This is because team B is expected to win the game, so if team A wins the ratings need to adjust more. In this system, the adjustment factor is always added to the winner and subtracted from the loser. Therefore, the sum of all the team's ratings in this system never changes.

Another example of a game adjustment method is used by Richard Billingsley. His rating system is comprised of the following six components:

1. Starting position
2. Accumulating points
3. Strength of opponent
4. Instituting deductions for losses
5. Site of the game
6. Instituting head to head rules

These components are then used to adjust each team's rating after each game. For example, under his system losing to the top ranked team as of the first week of the year won't hurt a team's rating very much. However, if that top ranked team begins to

lose more and more games throughout the year and another team loses to them in the last week of the season, then the losing team's rating will be dropped by a higher margin. His model makes the assumption that when you play a team is vital and whatever happens to an opponent's rating after a game is ignored (Billingsley, October).

CHAPTER 3. RATING METHODS

After performing the research surrounding the different rating methods, three different existing methods were implemented, and one new method was created. All of the rating methods implemented used a least squares approach, as well as shared the same database design. The following sections detail the rationale for each rating method, the logic behind these rating methods, and the calculations performed in each of them.

Database Design

Before any rating method can be applied, one must first determine what sorts of inputs will be accepted. In all of the rating methods discussed in this paper, the only pieces of information inputted are the list of teams, conferences, and divisions, as well as a list of every game played during that year. The final score for each game is input, as is which team was the home team. Finally, a Boolean variable is input with each game that indicated whether or not the game was played in a neutral site. This information was input into the database via a comma separated value file. The project used to import this data can be found in Appendix D, under the heading Database Import Project.

Figure 1 below provides a UML diagram of the database used for this project. The four main entities are Team, Conference, Division, and Game. Every team belongs to a conference; everyone conference belongs to a division; and every game consists of two teams. Additionally, this model supports tie games. In an effort to improve the

performance of the database, the TeamResult, ConferenceResult, and DivisionResult tables were created. These tables store information obtained during program execution, such as wins, losses, and rating, and can then be accessed instantly without having to re-compute the values. The Group field in the Team, Conference, and Division tables is used determine which teams, conferences, and divisions are connected through their schedule of games. The algorithm to determine this value is discussed next.

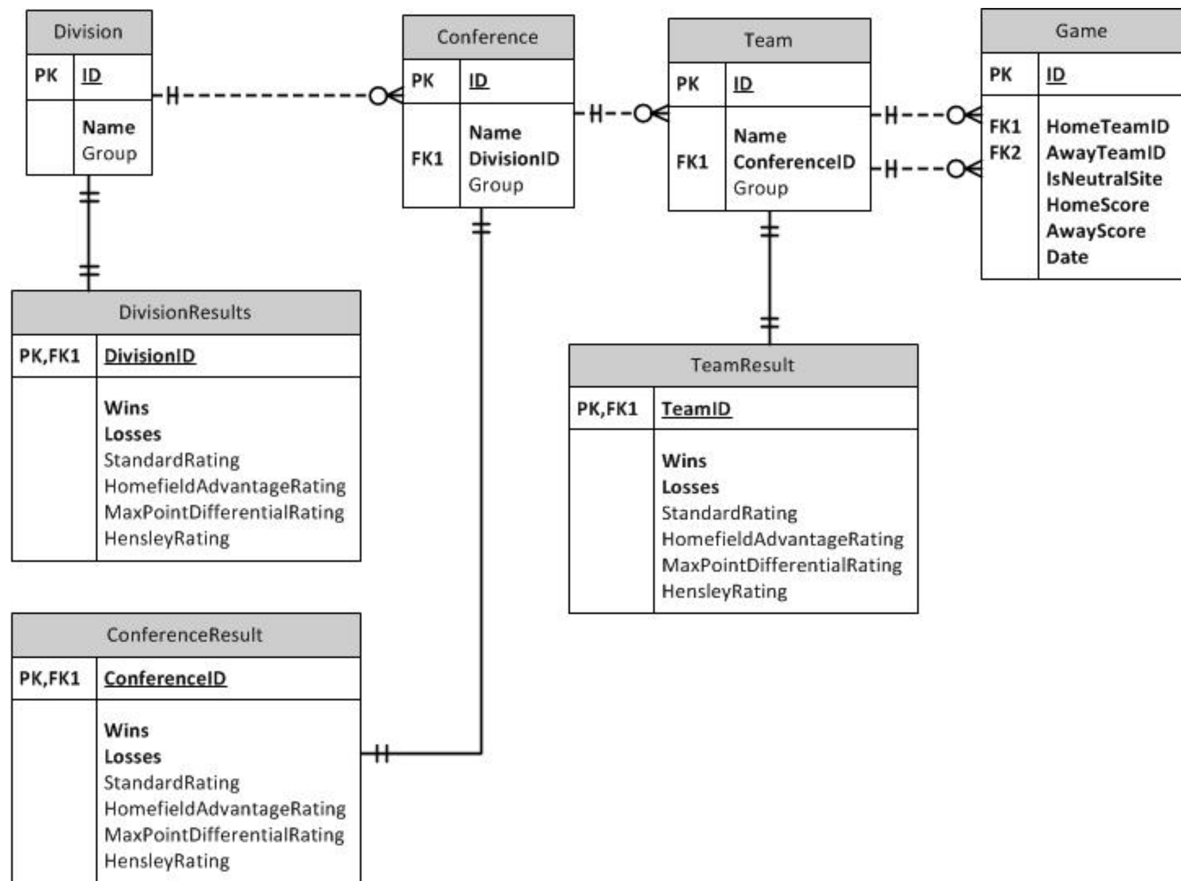

Figure 1. Database UML Diagram

Entity Groups

When considering a collection of games between different teams, it is helpful to picture it as a graph. The teams represent the vertices on the graph, while the games represent the edges connecting the two teams. Consider the sample schedule listed in Table 1 below.

Table 1. Sample Schedule

Week 1	Team A vs. Team B Team C vs. Team D Team E vs. Team F Team G vs. Team H
Week 2	Team A vs. Team C Team B vs. Team D
Week 3	Team E vs. Team B

After the first three weeks, the graph would look like Figure 2. The first group would be Team A, Team B, Team C, Team D, Team E, and Team F. The second group would be Team G and Team H. A group is defined as the collection of teams that are connected via the path on the graph. This is important later on for determining ratings since ratings are unique to the group for which they are calculated. Suppose that in week 4 Team G played against Team D. This would then connect all the teams, and there would be only one group. At that point, the ratings would encompass all teams and, therefore, be a good measure of comparing teams against one another.

For conferences and divisions, the group is defined as the collection of either conferences or divisions that have teams that are connected to teams from other conferences or divisions. For instance, if Team A plays Team B and the two teams are in different conferences, the conferences are connected. When determining the

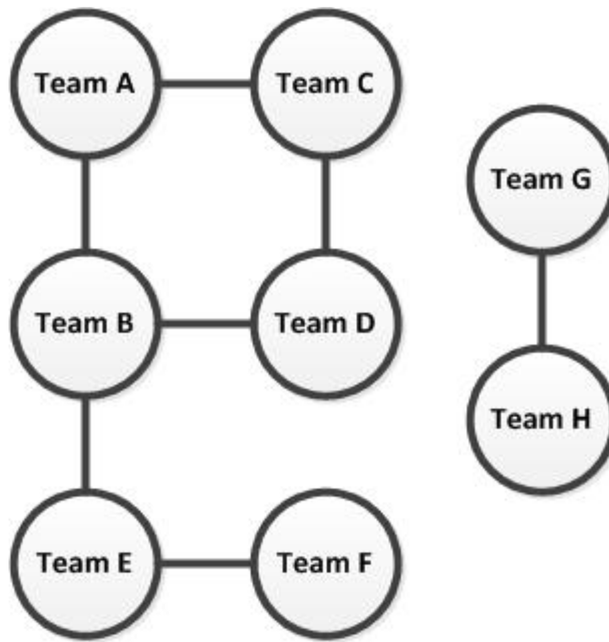

Figure 2. Sample Team Graph

groups for conferences, games that are played between two teams in the same conference are ignored. Likewise when determining the groups for divisions, games that are played between two teams in the same division are ignored.

With so many games being played each week, the teams become connected in a very small amount of time. In order to get an idea of how quickly the teams were connected, I imported each week of game separately and calculated the largest number of connected teams. Figure 3 below shows the results of these calculations after each week of games. In both 2008 and 2009, 716 teams were connected after the 5th week of games. There were 726 total teams in both 2008 and 2009; however, the New England Small College Athletic Conference is unique in that it does not play any non-conference games. Therefore, all 10 of the teams in that conference are never

connected to the rest of the teams each year. Thus, all of those teams were omitted from the primary group of rating and had their own independent set of ratings.

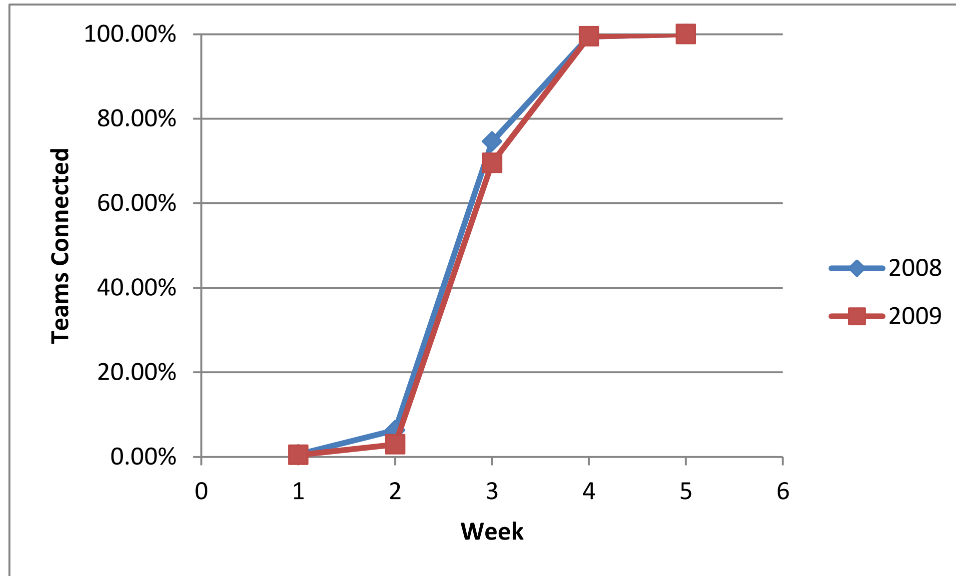

Figure 3. Teams Connected By Week

Standard Rating

The standard rating developed in the program uses the least squares method where y_i, or the observable game outcome, is simply the point differential. This method was the starting block for the program and was designed first so that the other rating systems could build onto it. The rating system of equations is of the following form:

$$\begin{bmatrix} G_1 & -g_{12} & -g_{13} & \dots & -g_{1n} \\ -g_{12} & G_2 & -g_{23} & \dots & -g_{2n} \\ -g_{13} & -g_{23} & G_3 & \dots & -g_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ -g_{1(n-1)} & -g_{2(n-1)} & -g_{3(n-1)} & \dots & G_{n-1} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \dots \\ r_n \end{bmatrix} = \begin{bmatrix} pd_1 \\ pd_2 \\ pd_3 \\ \dots \\ pd_{n-1} \\ 0 \end{bmatrix}$$

where G_n is number of games played by team n, g_{xy} is number of games played between team x and team y, r_n is the rating for team n, and pd_n is the total point differential for team n.

This system of equations is then stored as a matrix with N+1 columns, where the final column is the right hand side of the system of equations. This matrix is constructed using the following procedure:

```
Initialize all values in Matrix to 0
For every team t in all teams
   Set Matrix[t][t] = Total number of games played for team t
   Set Matrix[t][N+1] = Total Point Differential for team t
For every game g in all games
   Subtract one from Matrix[homeTeam][awayTeam]
   Subtract one from Matrix[awayTeam][homeTeam]
Set Matrix[N] = 0, 0, 0, …, 1
```

The matrix was then reduced using a class I wrote that applied the Gauss-Jordan elimination technique.

This same methodology was applied to conferences and divisions with the only difference being that only intra-conference and intra-division games were used as the observations, respectively. The entire code for this rating system can be found in Appendix D under the heading Rating System Project.

Home Field Advantage (HFA)

To determine the impact home games have on a team's rating, a general home field rating needs to be introduced. This home field rating is an adjustment that is

added to home team's rating. For example, if Team A's rating is 10, Team B's rating is 8, and the home field rating is 3, then the expected outcome for a game played at Team B would be a win for Team B. However, a home game played at Team A would be a loss for Team B. The rating system of equations is of the following form:

$$\begin{bmatrix} G_1 & -g_{12} & -g_{13} & \dots & -g_{1n} & H_1 \\ -g_{12} & G_2 & -g_{23} & \dots & -g_{2n} & H_2 \\ -g_{13} & -g_{23} & G_3 & \dots & -g_{3n} & H_3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -g_{1(n-1)} & -g_{2(n-1)} & -g_{3(n-1)} & \dots & G_{n-1} & H_{n-1} \\ H_1 & H_2 & H_3 & \dots & H_n & G_h \\ 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \dots \\ r_{n-1} \\ r_h \end{bmatrix} = \begin{bmatrix} pd_1 \\ pd_2 \\ pd_3 \\ \dots \\ pd_{n-1} \\ pd_h \\ 0 \end{bmatrix}$$

where H_n is the $Number\ of\ Home\ Games - Number\ of\ Away\ Games$ for team n, G_h is the total number of home games for all teams, r_h is the home field advantage rating, pd_h is the total point differential for all home games.

This system of equations is then stored as a matrix with N+2 columns, where the final column is the right hand side of the system of equations. This matrix is constructed using the following procedure:

```
Initialize all values in Matrix to 0
For every team t in all teams
    Set Matrix[t][t] = Total number of games played for team t
    Set Matrix[t][N+1] = Home Game Differential
    Set Matrix[N+1][t] = Home Game Differential
    Set Matrix[t][N+2] = Total Point Differential
For every game g in all games
    Subtract one from Matrix[homeTeam][awayTeam]
    Subtract one from Matrix[awayTeam][homeTeam]
```

```
Set Matrix[N] = 0, 0, 0, …, 1
Set Matrix[N+1][N+1] = Total Number of Home Games
Set Matrix[N+1][N+2] = Total Point Differential of Home Games
```

Using this method, the home field advantage rating was calculated from 2000 – 2009 and Table 2 shows the results. It is interesting to see that the average home field advantage is significantly higher for conferences and division than teams. One possible reason for this is because many of the weaker conferences and weaker divisions play a large majority of their intra-conference and intra-division games away and against better opponents. It is also interesting to note the decline in the home field advantage across teams, conferences, and divisions from 2000 – 2009. This could possibly be due to a higher degree of parity in college football in recent years.

Table 2. Home Field Advantage by Year

Year	Team	Conference	Division
2009	2.322	3.386	3.902
2008	2.665	3.869	5.252
2007	2.580	3.514	3.325
2006	2.627	4.547	3.171
2005	2.897	3.784	2.660
2004	3.284	4.526	4.265
2003	3.024	4.404	5.808
2002	2.785	4.809	6.180
2001	2.822	4.780	6.665
2000	3.372	4.560	6.215
Average	2.838	4.218	4.744

In the real world, not all game locations are equal. For instance, playing in Lincoln, NE, where every game has been sold out since 1962 and seating is now up to 81,067 is very intimidating to opponents (Memorial Stadium, 2009). On the other hand, the University of Idaho's Kibbie Dome has a maximum stadium capacity of

15,820 and wouldn't create much noise to interfere with the opposing team (Kibbie Dome, 2010). Therefore, future work could be done to determine a unique home field advantage rating for each team. The entire code for this rating system can be found in Appendix D under the heading Rating System Project.

Max Point Differential (MPD)

The debate between whether or not to include point differential in ratings is controversial. As of the 2009 season, computer ratings used by the BCS were not allowed to factor in the point differential for their ratings. I personally believe that ignoring the point differential entirely is ignoring valuable data. However, the extent to which it is taken into account is much trickier. For instance, consider the following two examples. Suppose Team A beats Team B by 56 points, primarily because it keeps its starters in the entire game. However, Team C beats Team B by 35 points but was up by 56 points before it put in the second and third string players. Should Team A's victory be counted as more impressive than Team C's victory? This is where adding a cap to the point differential is appropriate. The matrix for N teams was constructed the same way as in the Standard Rating with just one change: the total point differential for each team was computed with a user specified cap for each game's point differential. For example, if team A had the following point differentials: +13, +24, and -16, then the total point differential with a cap of 14 would be 13 + 14 – 14 = 13.

In order to experiment with what the cap should be, I computed the ratings for the 2009 season for caps of 1, 7, 14, and 28. The case of 1 represents a complete

abandonment of the point differential. In that case, any win is 1 point and any loss is -1 point. Table 3 shows the comparison of the final AP poll with the different caps.

Table 3. Max Point Differential Comparison

Team	**Final AP Poll**	**MPD=1 Rating**	**MPD=7 Rating**	**MPD=14 Rating**	**MPD=28 Rating**
Alabama (14-0)	#1	2.654 (1)	17.593 (1)	33.856 (1)	58.250 (1)
Texas (13-1)	#2	2.393 (3)	15.573 (3)	29.768 (4)	55.103 (3)
Florida (13-1)	#3	2.477 (2)	16.447 (2)	31.172 (2)	55.270 (2)
Boise State (14-0)	#4	2.275 (7)	14.866 (6)	27.846 (7)	48.288 (8)
Ohio State (11-2)	#5	2.240 (8)	14.751 (7)	28.253 (6)	49.080 (6)
TCU (12-1)	#6	2.308 (4)	14.899 (5)	29.100 (5)	52.526 (5)
Iowa (11-2)	#7	2.234 (9)	14.064 (12)	26.317 (16)	44.004 (23)
Cincinnati (12-1)	#8	2.283 (6)	14.249 (9)	26.781 (12)	45.623 (19)
Penn State (11-2)	#9	2.104 (13)	13.945 (13)	26.870 (11)	47.964 (11)
Virginia Tech (10-3)	#10	2.186 (11)	14.966 (4)	29.823 (3)	54.377 (4)

Clearly, the cap on the point differential plays a large part in the ratings. Both Iowa and Cincinnati get progressively lower ratings as the cap is increased. However, Virginia Tech which suffered three very close losses suffers when the cap is set to 1 point. In an effort to provide a median cap of these values, only the MPD=14 Rating is considered for the remainder of this paper. Future work could be done to provide further investigation on the ideal choice for the cap. The entire code for this rating system can be found in Appendix D under the heading Rating System Project.

Hensley Rating

After completing the three previous rating systems, I wanted to create a new method that combined the advantages of both the HFA Rating and the MPD Rating. Thus, the Hensley Rating is a hybrid of the two. I started with the HFA Rating system

and then started to experiment with different values of factoring in the point differential. I wanted to create a function that would take in as inputs the winning score and the losing score and would return a value that indicated the measure of the victory.

In order to create this function, I started to consider what defined the level of victory in a game's final score. Obviously, the point differential was a large factor. However, I didn't believe that alone was entirely accurate. For example, a victory 48 – 41 is not as impressive as a victory 10 – 3. In high scoring games, another possession for the losing team may have been all that was needed for the go ahead score. However, low scoring games indicate a defensive battle and a much lower chance that the losing team was only a possession away from winning. Therefore, I believe level of victory should follow diminishing returns in term of the total score.

With these principles in mind, I then began experimenting with different components of a game's final score. For instance, I looked at the winner's score, the loser's score, the margin of victory, and a variety of ratios involving all three components. I then created a spreadsheet in Excel to help experiment different possible equations to compute an accurate level of victory. In the spreadsheet, I created two columns: one for the winner's score and one for the loser's score. I then decided it made the most sense to analyze the data by keeping the margin of victory the same and simply incrementing both scores by the same amount each time. In this way, I was able to see how a given level of victory equation behaved for a given

margin of victory as the winner's score increased. Finally, I created a number of series to track various margin of victories.

Upon completing this experimentation, I developed the following equations to compute the level of victory for a particular game:

$$MarginOfVictory = WinnerScore - LoserScore$$

$$Volatility = \frac{LoserScore}{WinnerScore}$$

$$LevelOfVictory = \sqrt{(1 - Volatility) \cdot MarginOfVictory}$$

The $Volatility$ component is what adds the diminishing return to the value. It attempts to measure the degree of unpredictability given a rematch. The value will always be greater than or equal to 0 and less than 1 since ties are not allowed in college football. For example, a victory 48 – 41 yields a $Volatility\ = 0.85$, while a victory 10 – 3 yields a $Volatility\ = 0.30$. In other words, the 48 – 41 game would have a much higher chance that a rematch could end with the losing team winning. Given a set point differential, the $Volatility$ is directly related to the total points scored. As more points are scored, the $Volatility$ will increase. In the extreme case of a shutout, the $Volatility = 0$. The $LevelOfVictory$ then adjusts the point differential based on this unpredictability factor. The square root was taken in an effort to bring these values closer together and smooth the values. Since the point differential is always positive, this operation is always valid. Finally, the $LevelOfVictory$ was capped at a minimum value of 1.0 and a maximum value of 4.0. To get a clear understanding of the $LevelOfVictory$ function, Figure 4 shows how the value is related to the winning score for a few common point differentials.

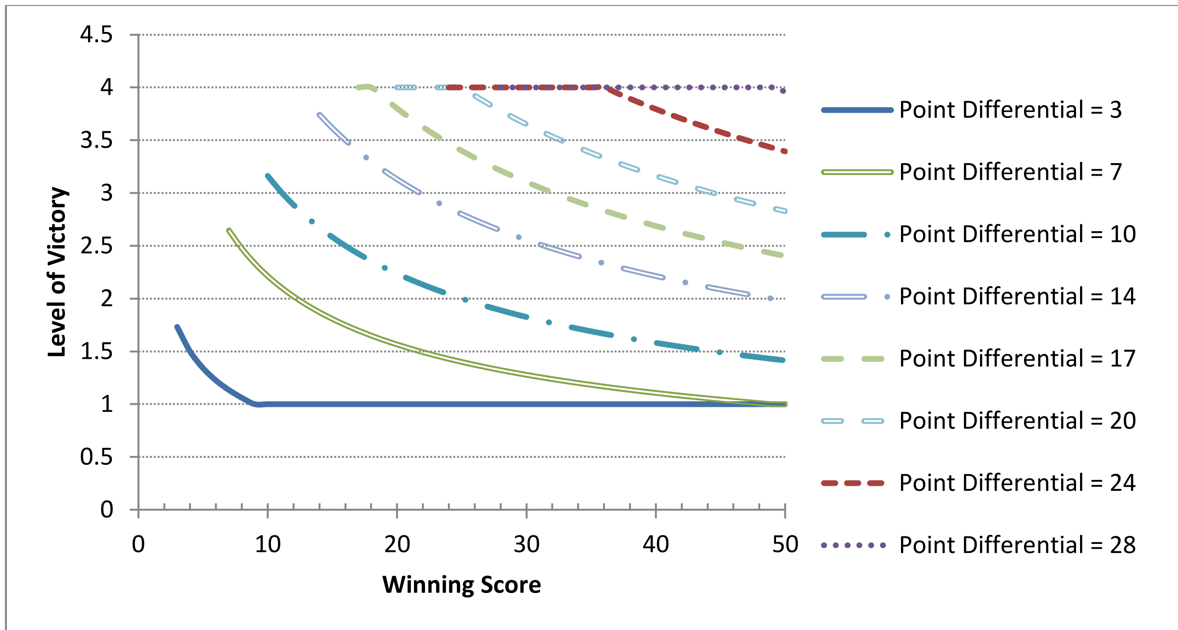

Figure 4. Level of Victory vs. Winning Score

A few important observations can be made from looking at the above figure. All games in which the point differential is 3 and the winning score is greater than 8 will result in the lowest $LevelOfVictory$. On the flip side, all games in which the point differential is 28 will result in the highest $LevelOfVictory$ when the winning score is less than 50 points. Another helpful way to compare the $LevelOfVictory$ is to look at final scores that result in the same value. For instance, a final score of 7 – 0, 16 – 6, 31 – 17, or 46 – 29 will all result in a $LevelOfVictory = 2.5$, which is the median value possible. The various $LevelOfVictory$ values are essentially equivalent to the following insights:

- 3.0 – 4.0 = Dominating victory, very small chance of losing a rematch
- 2.0 – 3.0 = Comfortable victory, small chance of losing a rematch
- 1.0 – 2.0 = Close victory, 50-50 chance of losing a rematch

With this function defined, the Hensley Rating can now be computed. The matrix is created in exactly the same way as the HFA Rating, except any point differential calculation is replaced with the $LevelOfVictory$ value. The loser of every game simply receives the negative value of the $LevelOfVictory$. This ensures the sum across all teams for all games is equal to 0. The entire code for this rating system can be found in Appendix D under the heading Rating System Project.

The intent of the Hensley Rating is to reward teams that play a difficult schedule and win games with a high $LevelOfVictory$. Therefore, defensively dominant teams that keep the scores low but have lower point differential than some of the high scoring offenses won't be penalized. I believe this greatly bridges the importance of both defense and offense in a team. In rating systems with only point differential, a team with a mediocre defense but a terrific offense is rewarded much more than a team with a terrific defense but a mediocre offense.

CHAPTER 4. ANALYSIS

Once the rating systems were developed, they were each run against previous years of college football data. Ten years of data were considered, from 2000 – 2009 (Howell, 2008). All teams from all conferences and divisions were included. In 2009, this includes 726 teams, 81 conferences, 6 divisions and 3,980 games (Wolfe, NCAA and NAIA Scores, 2010).

Determining the validity of a rating system can be tricky. Obviously, there should be some correlation between the top teams in a given rating system and the top teams listed in the human polls. However, it is important to note that human polls are biased and are based on a person's opinion. These polls are highly impacted by wins and losses and less so by strength of schedule. In human polls, people can base their votes on whatever criteria they choose. Additionally, human polls are highly influenced by pre-season polls. For instance, if team A is ranked #1 and team B is ranked #5 before the start of the season and both teams win all their games, it would be virtually impossible for team B to overpass team A, regardless of how the strength of schedules compare. Therefore, it is expected that differences will arise between any computer rating system and the human polls. This section will still perform a correlation analysis to the Final AP poll for various seasons, but these correlation results are less significant than those against existing computer rating systems.

Table 4 shows the results of the four different rating systems for the 2009 season. The MPD and Hensley Rating had Alabama as the best team, while the Standard and HFA Rating selected Florida (despite Alabama beating Florida in the SEC

Championship game). Two other irregularities jump out when comparing the four rating systems to the final AP poll. Iowa was ranked #7 in the final AP poll, but significantly lower in all the computer polls. This is an excellent example of how playing a weak schedule can negatively impact a team's rating. On the other hand, Virginia Tech is a good example of how playing a difficult schedule can positively impact a team's rating. Despite losing three games, Virginia Tech was ranked in the top 5 in all four computer ratings, while the final AP poll had them ranked #10. Please see Appendix A for a complete comparison of the four rating systems to the final AP poll for 2000 – 2009. Appendices B and C provide the same comparison but for conferences and divisions instead.

Table 4. Rating Comparison for 2009

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Alabama (14-0)	#1	76.832 (2)	72.576 (2)	33.856 (1)	8.569 (1)
Texas (13-1)	#2	76.118 (3)	72.149 (3)	29.768 (4)	7.798 (4)
Florida (13-1)	#3	77.138 (1)	73.023 (1)	31.172 (2)	8.002 (2)
Boise State (14-0)	#4	66.584 (10)	63.389 (7)	27.846 (7)	7.024 (8)
Ohio State (11-2)	#5	66.788 (9)	62.278 (10)	28.253 (6)	7.132 (6)
TCU (12-1)	#6	71.931 (5)	68.669 (4)	29.100 (5)	7.577 (5)
Iowa (11-2)	#7	60.197 (25)	56.011 (24)	26.317 (16)	6.602 (16)
Cincinnati (12-1)	#8	64.158 (15)	60.749 (13)	26.781 (12)	6.810 (11)
Penn State (11-2)	#9	64.456 (14)	59.923 (16)	26.870 (11)	6.911 (10)
Virginia Tech (10-3)	#10	72.292 (4)	68.331 (5)	29.823 (3)	7.809 (3)

To provide a fast way to analyze each rating system for a given year, I created a separate project in my solution that computed a variety of statistics. The entire project's code is located in Appendix D, under the heading Rating Evaluator Project. The statistics that follow are all computed with this project.

In order to judge how closely the ratings correlated with the final AP poll, the Pearson correlation coefficient was calculated and the results from 2000 – 2009 are displayed in Table 5 (Final AP Polls). The Hensley Rating had the highest average value at 0.79 and indicates a strong positive correlation between the final AP poll.

Table 5. Correlation against Final AP Poll for 2000 – 2009

Year	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
2009	0.70	0.71	0.78	0.77
2008	0.73	0.74	0.85	0.86
2007	0.58	0.57	0.64	0.66
2006	0.69	0.69	0.73	0.77
2005	0.70	0.72	0.76	0.88
2004	0.73	0.72	0.79	0.77
2003	0.70	0.68	0.80	0.78
2002	0.78	0.78	0.83	0.84
2001	0.73	0.73	0.82	0.80
2000	0.75	0.77	0.76	0.79
Average	**0.71**	**0.71**	**0.78**	**0.79**

On Kenneth Massey's website, he has collected the rankings from 114 computer rating systems for college football and used these ratings to compute the average rank for each team in the Football Bowl Subdivision (FBS) (Massey, College Football Ranking Comparison, 2010). To determine how my ratings correlated to these rankings, I first computed the rank for every team in my ratings. I then removed any teams that were not part of the FBS and recalculated their ranks accordingly. Finally, I computed the correlation coefficient between each of my four rating systems to this average computer ranking. Table 6 shows the results for 2000 – 2009. It's important to remember the difference between ratings and rankings. In a ranking, the

teams are simply ordered from best to worst. In a rating, each team is assigned a numerical value indicating their strength. Therefore, rankings are not as precise.

Table 6. Correlation against Average Computer Rankings for 2000 – 2009

Year	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
2009	0.96	0.97	0.98	0.99
2008	0.96	0.96	0.98	0.99
2007	0.96	0.97	0.97	0.98
2006	0.96	0.96	0.98	0.98
2005	0.96	0.96	0.98	0.98
2004	0.94	0.95	0.97	0.97
2003	0.96	0.96	0.98	0.98
2002	0.97	0.97	0.98	0.99
2001	0.96	0.96	0.98	0.98
2000	0.98	0.98	0.98	0.99
Average	**0.96**	**0.96**	**0.98**	**0.98**

After reviewing the data, all four rating systems displayed an extremely high positive correlation, with average values all greater than 0.95. Once again the MPD and Hensley Rating yielded the highest correlations. Overall, these very strong signs that correlations provide indicate that these four rating systems are quite accurate and agree with the consensus of existing computer rating systems.

Another method of measuring the validity of rating systems is examining the past games and determining the number of times a higher ranked team lost to a lower ranked team. In a perfect world, this number would be 0. However, college football is far from the ideal world. Upsets are commonplace and even good teams will lose to bad teams throughout the season. Table 7 shows the percentage of all games that were upsets for each of the rating systems for 2000 – 2009. The Average Computer

Ratings column was obtained from Kenneth Massey's comparison web page. The MPD Rating had the lowest average percentage of upsets at 15.7%, with the Hensley Rating nearly as good at 15.9%. All four of the rating systems performed better than the average of existing rating systems.

Table 7. Number of Upsets for 2000 – 2009

Year	Avg Computer Rating	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
2009	18.3%	16.9%	16.8%	15.6%	15.3%
2008	18.7%	16.6%	16.6%	14.5%	14.7%
2007	18.7%	16.8%	16.7%	15.0%	15.7%
2006	16.8%	18.9%	18.6%	16.5%	16.6%
2005	19.1%	18.6%	18.7%	16.7%	16.7%
2004	17.2%	17.9%	17.8%	15.3%	15.9%
2003	18.0%	17.0%	17.3%	15.9%	16.3%
2002	16.8%	17.8%	18.0%	16.0%	16.3%
2001	16.9%	16.8%	17.0%	15.1%	15.4%
2000	16.8%	17.9%	17.7%	16.3%	16.0%
Average	**17.7%**	**17.5%**	**17.5%**	**15.7%**	**15.9%**

One final algorithm used to determine the amount of error in a rating system was developed by Eugene Potemkin (Potemkin). It consists of the following formulas:

$$Estimate\ of\ Mistake = Size\ of\ Mistake * Importance\ of\ Mistake$$

$$Size\ of\ Mistake = N * \frac{Rank_{Winner} - Rank_{Loser}}{Rank_{Winner} * Rank_{Loser}}$$

$$Importance\ of\ Mistake = N * \frac{Rank_{Winner} + Rank_{Loser}}{Rank_{Winner} * Rank_{Loser}}$$

where N is the number of teams in the rating (Potemkin). By summing together all the mistakes, one can then compare rating systems to one another. For every upset, the

estimate of mistake was calculated and totaled. Table 8 shows a comparison between the four rating systems for the average mistake value for 2000 – 2009. The Hensley Rating had the lowest average mistake size at 1,204, while the Standard Rating had the highest average mistake size at 1,343.

Table 8. Average Mistake Value for 2000 – 2009

Year	**Standard Rating**	**HFA Rating**	**MPD=14 Rating**	**Hensley Rating**
2009	914	871	361	375
2008	1,212	1,210	1,358	1,325
2007	2,983	2,414	2,740	2,588
2006	1,346	1,885	2,005	1,994
2005	278	280	902	289
2004	367	369	176	217
2003	1,951	1,932	1,415	1,396
2002	1,949	1,935	2,197	2,106
2001	801	796	783	790
2000	1,634	1,550	923	960
Average	**1,343**	**1,324**	**1,286**	**1,204**

After performing the analysis on the different ratings, the Hensley Rating appear to be the best choice. It yielded the highest correlation to the final AP poll and average computer rankings, had just 0.2% more upsets than the MPD Rating, and had the lowest average mistake size. Using the same metrics, the MPD Rating seem the next best, followed by the HFA Rating, and finally the Standard Rating.

CHAPTER 5. CONCLUSION

Each and every year the controversy over who should play for the national championship spreads across the United States. Some years there are only two undefeated teams, and the choice is seemingly obvious. Other times there are three or even four teams all with 1 loss. Then, the question of who deserves to play is often difficult to judge. The two human polls included in the BCS are not enough to settle this debate. Thus, the computer ratings are brought in to help add some unbiased, quantitative analysis.

The goal of computer ratings is to use whatever information was collected in the past, such as schedules, wins, and points scored, and compute the most accurate value that represents each team's strength. Many methods exist for these rating systems, such as the least squares, maximum likelihood estimator, and the Elo rating. In this paper, the least squares approach was applied to four different rating systems: the Standard Rating, the HFA Rating, the MPD Rating, and the Hensley Rating.

The Standard Rating made no adjustments to the point differential, while the MPD Rating set a cap on this value for each game. The HFA Rating added a home field advantage component to better reflect the reality of home and away games. Finally, the Hensley Rating was designed to incorporate the benefits of both the MPD Rating and the HFA Rating. Instead of using a strict cap on the point differential, a function was created to compute the value of the victory for each game. The goal of this was to extract more meaningful information from the winning and losing score instead of simply the point differential.

The ratings systems were then used on the previous years of college football from 2000 – 2009. The analysis showed a strong correlation to the final AP top 25 for all ten years, with the Hensley Rating providing the highest average correlation. Even more remarkable was the extremely high correlation for all four rating systems to the average computer ranking across 114 existing systems. Finally, a measure for estimating the mistake size of each upset present in the rankings was applied. Once again, the Hensley Rating yielded the lowest average error. Overall, the Hensley Rating performed the best of the four rating systems. Its unique capability to determine the level of each victory makes it quite effective at identifying the best teams.

The importance of the computer ratings in the BCS is often overlooked. Accounting for a full third of the BCS Rating, the computer ratings can indeed be the difference between who plays in the national championship game. For instance after the last week of the 2001 season, Nebraska was ranked #4 in both human polls but averaged 2.17 in the computer ratings and was selected to play in the national championship game (2001 BCS Standings). As such, these computer ratings must be as accurate as possible. The Hensley Rating focuses on the importance of the strength of schedule and the strength of each victory. Its level of victory function benefits both defensively dominate and offensively dominate teams equally. Because of this, I believe the Hensley Rating makes an excellent addition to the catalogue of computer rating systems for college football.

REFERENCES

2001 BCS Standings. (n.d.). Retrieved March 25, 2010, from College Football Poll: http://www.collegefootballpoll.com/2001_archive_bcs.html

Memorial Stadium. (2009, December 5). Retrieved March 25, 2010, from Nebraska: http://www.huskers.com/ViewArticle.dbml?DB_OEM_ID=100&ATCLID=734

BCS Computer Rankings. (2010, January 21). Retrieved March 27, 2010, from Bowl Championship Series: http://www.bcsfootball.org/news/story?id=4765872

Kibbie Dome. (2010). Retrieved March 28, 2010, from College Gridirons: http://www.collegegridirons.com/wac/KibbieDome.htm

Billingsley, R. (October, 2008). *Dynamics of the System*. Retrieved February 11, 2010, from College Football Research Center: http://www.cfrc.com/Archives/Dynamics_08.htm

Colley, W. N. (2002). Colley's Bias Free College Football Ranking Method: The Colley Matrix Explained.

Elo, A. (1978). *The Rating of Chessplayers.*

Final AP Polls. (n.d.). Retrieved February 23, 2010, from http://preseason.stassen.com/final-ap-poll/

Gill, R., & Keating, J. (2009). Assessing Methods for College Football Rankings. *Journal of Quantitative Analysis in Sports*.

Govan, A. Y., Langville, A. N., & Meyer, C. D. (2009). Offense-Defense Approach to Ranking Team Sports. *Journal of Quantitative Analysis in Sports*.

Howell, J. (2008). *James Howell's College Football Scores*. Retrieved January 5, 2010, from http://homepages.cae.wisc.edu/~dwilson/rsfc/history/howell/

Massey, K. (1997). Statistical Models Applied to the Rating of Sports Teams.

Massey, K. (2010). *College Football Ranking Comparison*. Retrieved February 17, 2010, from Massey Ratings: http://www.masseyratings.com/cf/compare.htm

Massey, K. (n.d.). *Massey Ratings Description*. Retrieved January 15, 2010, from Massey Ratings: http://www.masseyratings.com/theory/massey.htm

Potemkin, E. (n.d.). *RankRat*. Retrieved March 25, 2010, from http://rsport.netorn.ru/cf/rankrank0304.htm

Sagarin, J. (2010). *Jeff Sagarin in NCAA football ratings*. Retrieved March 3, 2010, from USAToday.com: http://www.usatoday.com/sports/sagarin/fbt09.htm

Trono, J. A. (2007). An Effective Nonlinear Rewards-Based Ranking System. *Journal of Quantitative Analysis in Sports*.

West, B. T., & Lamsal, M. (2008). A New Application of Linear Modeling in the Prediction of College Football Bowl Outcomes and the Development of Team Ratings. *Journal of Quantitative Analysis in Sports*.

Wolfe, P. (2010). *NCAA and NAIA Scores*. Retrieved January 13, 2010, from 2009 College Football: http://prwolfe.bol.ucla.edu/cfootball/scores.htm

Wolfe, P. (n.d.). *About These Ratings*. Retrieved February 7, 2010, from 2009 College Football: http://prwolfe.bol.ucla.edu/cfootball/descrip.htm

APPENDIX A. TEAM RATINGS FROM 2000 – 2009

In order to effectively analyze the four rating systems developed, I computed the results of all college football games for the years 2000 – 2009. Tables 9 – 18 summarize the results. The final rating with each of the four rating systems is displayed and compared to the final AP poll for that year. The teams are ordered by their position in the final AP poll.

Table 9. 2009 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Alabama (14-0)	#1	76.832 (2)	72.576 (2)	33.856 (1)	8.569 (1)
Texas (13-1)	#2	76.118 (3)	72.149 (3)	29.768 (4)	7.798 (4)
Florida (13-1)	#3	77.138 (1)	73.023 (1)	31.172 (2)	8.002 (2)
Boise State (14-0)	#4	66.584 (10)	63.389 (7)	27.846 (7)	7.024 (8)
Ohio State (11-2)	#5	66.788 (9)	62.278 (10)	28.253 (6)	7.132 (6)
TCU (12-1)	#6	71.931 (5)	68.669 (4)	29.100 (5)	7.577 (5)
Iowa (11-2)	#7	60.197 (25)	56.011 (24)	26.317 (16)	6.602 (16)
Cincinnati (12-1)	#8	64.158 (15)	60.749 (13)	26.781 (12)	6.810 (11)
Penn State (11-2)	#9	64.456 (14)	59.923 (16)	26.870 (11)	6.911 (10)
Virginia Tech (10-3)	#10	72.292 (4)	68.331 (5)	29.823 (3)	7.809 (3)

Table 10. 2008 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Florida (13-1)	#1	90.515 (1)	85.820 (1)	37.984 (1)	9.457 (1)
Utah (13-0)	#2	71.968 (12)	68.314 (9)	30.990 (8)	7.744 (8)
Southern Cal (12-1)	#3	88.273 (2)	83.956 (2)	36.435 (2)	9.092 (2)
Texas (12-1)	#4	84.551 (4)	79.863 (4)	35.407 (4)	8.902 (4)
Oklahoma (12-2)	#5	88.130 (3)	83.750 (3)	35.814 (3)	8.978 (3)
Alabama (12-2)	#6	72.410 (10)	67.916 (12)	31.393 (7)	7.864 (7)
TCU (11-2)	#7	74.315 (7)	70.753 (6)	29.531 (14)	7.609 (9)
Penn State (11-2)	#8	80.532 (5)	76.280 (5)	32.505 (5)	8.280 (5)
Ohio State (10-3)	#9	72.353 (11)	68.114 (11)	30.833 (10)	7.527 (10)
Oregon (10-3)	#10	72.433 (9)	68.214 (10)	29.875 (12)	7.181 (16)

Table 11. 2007 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
LSU (12-2)	#1	72.676 (3)	68.022 (3)	31.853 (1)	7.651 (3)
Georgia (11-2)	#2	66.011 (10)	61.282 (10)	30.278 (4)	7.292 (7)
Southern Cal (11-2)	#3	70.282 (8)	65.842 (8)	31.291 (2)	7.690 (2)
Missouri (12-2)	#4	71.194 (6)	66.815 (6)	30.153 (8)	7.324 (6)
Ohio State (11-2)	#5	67.972 (9)	63.425 (9)	30.210 (6)	7.535 (4)
West Virginia (11-2)	#6	73.557 (2)	69.624 (1)	30.855 (3)	7.707 (1)
Kansas (12-1)	#7	72.240 (4)	67.937 (4)	27.421 (13)	6.965 (11)
Oklahoma (11-3)	#8	73.888 (1)	69.457 (2)	28.647 (10)	7.277 (8)
Virginia Tech (11-3)	#9	63.692 (14)	59.527 (14)	30.168 (7)	7.234 (10)
Boston College (11-3)	#10t	57.800 (30)	53.607 (30)	27.839 (12)	6.626 (17)
Texas (10-3)	#10t	63.089 (16)	58.906 (16)	25.040 (26)	6.347 (20)

Table 12. 2006 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Florida (13-1)	#1	70.664 (5)	65.675 (5)	31.460 (5)	7.883 (5)
Ohio State (12-1)	#2	72.908 (1)	68.596 (2)	31.511 (4)	8.175 (2)
LSU (11-2)	#3	72.835 (3)	67.326 (3)	30.204 (7)	7.559 (7)
Southern Cal (11-2)	#4	72.873 (2)	68.617 (1)	34.256 (1)	8.448 (1)
Boise State (13-0)	#5	64.244 (14)	60.785 (12)	28.370 (10)	7.182 (10)
Louisville (12-1)	#6	71.241 (4)	67.142 (4)	32.365 (2)	8.135 (3)
Wisconsin (12-1)	#7	61.589 (21)	57.458 (20)	27.356 (15)	7.146 (11)
Michigan (11-2)	#8	66.919 (9)	62.572 (8)	32.363 (3)	8.041 (4)
Auburn (11-2)	#9	62.525 (16)	57.350 (21)	27.414 (13)	6.885 (15)
West Virginia (11-2)	#10	66.924 (8)	62.490 (9)	29.556 (8)	7.480 (8)

Table 13. 2005 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Texas (13-0)	#1	95.427 (1)	90.756 (1)	38.058 (2)	9.550 (1)
Southern Cal (12-1)	#2	91.907 (2)	87.047 (2)	38.359 (1)	9.513 (2)
Penn State (11-1)	#3	80.597 (4)	75.032 (4)	36.486 (4)	8.760 (4)
Ohio State (10-2)	#4	84.405 (3)	78.727 (3)	38.013 (3)	9.163 (3)
West Virginia (11-1)	#5	69.839 (17)	65.382 (16)	30.832 (18)	7.566 (13)
LSU (11-2)	#6	73.239 (11)	68.332 (11)	31.295 (12)	7.917 (8)
Virginia Tech (11-2)	#7	79.438 (5)	74.559 (5)	34.822 (5)	8.526 (5)
Alabama (10-2)	#8	68.492 (22)	63.156 (23)	30.680 (19)	7.624 (12)
Notre Dame (9-3)	#9	76.812 (6)	71.497 (6)	34.208 (6)	8.207 (6)
Georgia (10-3)	#10	71.451 (13)	66.233 (14)	32.455 (9)	7.809 (9)

Table 14. 2004 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Southern Cal (13-0)	#1	88.947 (1)	83.22 (1)	39.217 (1)	9.733 (1)
Auburn (13-0)	#2	75.161 (7)	69.201 (8)	34.534 (5)	8.563 (5)
Oklahoma (12-1)	#3	78.298 (5)	72.702 (5)	36.971 (3)	8.922 (4)
Utah (12-0)	#4	78.395 (4)	72.941 (4)	37.137 (2)	9.001 (2)
Texas (11-1)	#5	75.065 (8)	69.477 (7)	34.475 (6)	8.319 (6)
Louisville (11-1)	#6	78.744 (3)	74.308 (3)	32.421 (9)	8.119 (8)
Georgia (10-2)	#7	70.168 (12)	64.419 (13)	31.267 (13)	7.657 (12)
Iowa (10-2)	#8	65.710 (22)	60.182 (21)	30.598 (18)	7.655 (13)
California (10-2)	#9	81.178 (2)	76.239 (2)	35.275 (4)	8.983 (3)
Virginia Tech (10-3)	#10	73.956 (9)	68.434 (9)	31.218 (14)	7.600 (14)

Table 15. 2003 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Southern Cal (12-1)	#1	81.246 (3)	76.212 (3)	36.485 (1)	8.946 (1)
LSU (13-1)	#2	82.696 (2)	77.383 (2)	34.042 (2)	8.673 (2)
Oklahoma (12-2)	#3	86.926 (1)	81.247 (1)	33.215 (3)	8.406 (3)
Ohio St (11-2)	#4	69.908 (18)	63.654 (20)	30.326 (10)	7.360 (12)
Miami FL (11-2)	#5	73.108 (11)	67.279 (12)	30.172 (11)	7.407 (11)
Michigan (10-3)	#6	77.294 (5)	71.397 (7)	31.537 (5)	7.691 (7)
Georgia (11-3)	#7	75.427 (8)	69.906 (9)	32.897 (4)	8.051 (4)
Iowa (10-3)	#8	73.128 (10)	67.455 (11)	30.394 (9)	7.469 (9)
Washington St (10-3)	#9	68.095 (23)	63.425 (23)	30.419 (8)	7.353 (13)
Miami OH (13-1)	#10	73.304 (9)	70.058 (8)	29.809 (12)	7.436 (10)

Table 16. 2002 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Ohio State (14-0)	#1	72.188 (9)	66.597 (9)	33.125 (4)	8.200 (6)
Miami FL (12-1)	#2	76.820 (4)	72.000 (4)	33.154 (3)	8.466 (2)
Georgia (13-1)	#3	73.914 (6)	68.503 (6)	32.431 (6)	8.251 (5)
Southern Cal (11-2)	#4	83.445 (2)	78.294 (2)	36.868 (1)	9.297 (1)
Oklahoma (12-2)	#5	79.360 (3)	74.448 (3)	33.578 (2)	8.384 (3)
Texas (11-2)	#6	75.143 (5)	70.128 (5)	31.436 (7)	8.084 (7)
Kansas State (11-2)	#7	86.181 (1)	80.942 (1)	32.599 (5)	8.259 (4)
Iowa (11-2)	#8	72.428 (8)	66.960 (8)	31.145 (8)	7.872 (9)
Michigan (10-3)	#9	67.738 (16)	61.953 (18)	29.986 (12)	7.298 (16)
Washington St (10-3)	#10	68.704 (13)	63.858 (13)	30.698 (10)	7.487 (12)

Table 17. 2001 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Miami FL (12-0)	#1	93.318 (1)	88.344 (1)	39.084 (1)	9.837 (1)
Oregon (11-1)	#2	72.814 (9)	67.836 (8)	34.495 (6)	8.451 (5)
Florida (10-2)	#3	89.763 (2)	84.257 (2)	38.1 (2)	9.401 (2)
Tennessee (11-2)	#4	76.256 (5)	70.825 (5)	36.252 (3)	8.736 (3)
Texas (11-2)	#5	80.844 (3)	76.016 (3)	34.876 (4)	8.648 (4)
Oklahoma (11-2)	#6	75.508 (6)	70.02 (6)	33.265 (8)	8.173 (7)
LSU (10-3)	#7	70.016 (13)	64.532 (15)	33.418 (7)	7.795 (11)
Nebraska (11-2)	#8	78.555 (4)	72.765 (4)	34.688 (5)	8.369 (6)
Colorado (10-3)	#9	70.518 (12)	65.532 (12)	33.082 (9)	7.806 (10)
Washington St (10-2)	#10	67.475 (21)	62.814 (20)	31.719 (16)	7.659 (15)

Table 18. 2000 Team Results

Team	Final AP Poll	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Oklahoma (13-0)	#1	85.567 (4)	78.892 (3)	35.930 (2)	8.709 (2)
Miami FL (11-1)	#2	87.803 (2)	81.444 (2)	36.624 (1)	8.857 (1)
Washington (11-1)	#3	74.245 (9)	67.201 (9)	32.58 (7)	7.642 (8)
Oregon State (11-1)	#4	75.571 (8)	68.503 (8)	33.566 (5)	7.982 (6)
Florida State (11-2)	#5	90.944 (1)	83.522 (1)	35.349 (3)	8.685 (3)
Virginia Tech (11-1)	#6	78.227 (7)	71.634 (6)	34.738 (4)	8.129 (4)
Oregon (10-2)	#7	69.329 (18)	62.285 (19)	30.366 (12)	7.247 (11)
Nebraska (10-2)	#8	85.722 (3)	78.601 (4)	33.091 (6)	7.991 (5)
Kansas State (11-3)	#9	81.058 (5)	73.731 (5)	30.643 (10)	7.442 (9)
Florida (10-3)	#10	78.647 (6)	71.456 (7)	32.336 (8)	7.826 (7)

APPENDIX B. CONFERENCE RATINGS FROM 2000 - 2009

In college football, teams are organized into conferences ranging from around 8 – 12 teams. In order to rate these conferences, the intra-conference games were used as the observations and the four rating systems were applied. In order to effectively analyze the four rating systems developed, I computed the results of all college football games for the years 2000 – 2009. Tables 19 – 28 summarize the results. The conferences are ordered by the Hensley Rating.

Table 19. 2009 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
SEC (48-10)	55.797 (1)	49.860 (1)	21.652 (1)	5.165 (1)
Big East(36-10)	49.416 (3)	44.287 (3)	20.332 (2)	4.727 (2)
Big 12 (39-17)	52.855 (2)	47.088 (2)	19.579 (3)	4.626 (3)
Big 10 (36-15)	48.634 (4)	42.653 (4)	18.818 (4)	4.409 (4)
Pac-10 (23-14)	47.737 (6)	42.138 (6)	18.504 (5)	4.343 (5)
ACC (33-22)	47.744 (5)	42.346 (5)	18.352 (6)	4.330 (6)
Mountain West (25-16)	42.900 (8)	38.079 (8)	16.835 (7)	3.916 (7)
FBS Independents (19-15)	43.955 (7)	38.690 (7)	16.481 (8)	3.880 (8)
WAC (21-22)	35.972 (9)	31.706 (9)	13.142 (9)	3.151 (9)
Conference USA (21-33)	32.983 (10)	28.341 (11)	11.399 (10)	2.616 (10)

Table 20. 2008 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Big 12 (42-13)	58.072 (1)	51.752 (1)	23.208 (1)	5.427 (1)
SEC (43-13)	53.438 (2)	47.348 (2)	22.781 (2)	5.194 (2)
ACC (41-17)	49.328 (4)	43.223 (4)	21.517 (3)	4.807 (3)
Big 10 (33-18)	50.681 (3)	44.603 (3)	19.996 (4)	4.674 (4)
Big East (33-14)	47.456 (6)	42.289 (6)	19.593 (5)	4.551 (5)
Pac-10 (19-17)	48.951 (5)	42.94 (5)	19.361 (6)	4.378 (6)
Mountain West (28-13)	45.293 (7)	40.113 (7)	18.159 (7)	4.219 (7)
MAC (22-35)	39.742 (8)	35.937 (8)	14.77 (9)	3.479 (8)
WAC (19-23)	35.928 (11)	31.266 (12)	15.053 (8)	3.383 (9)
FBS Independents (18-28)	39.561 (9)	34.919 (9)	14.544 (11)	3.376 (10)

Table 21. 2007 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Pac-10 (25-12)	52.791 (1)	47.048 (1)	22.454 (1)	5.188 (1)
SEC (47-10)	52.105 (2)	45.912 (4)	22.357 (2)	5.188 (2)
Big East (32-13)	51.37 (4)	46.147 (3)	21.519 (3)	5.06 (3)
Big 10 (38-14)	48.759 (5)	42.989 (5)	21.291 (4)	4.841 (4)
Big 12 (41-15)	52.1 (3)	46.736 (2)	20.101 (5)	4.836 (5)
ACC (35-21)	45.434 (6)	40.305 (6)	19.863 (6)	4.555 (6)
Mountain West (24-17)	43.504 (7)	38.879 (7)	17.901 (7)	4.25 (7)
WAC (18-23)	35.902 (8)	31.792 (8)	14.507 (8)	3.356 (8)
FBS Independents (19-26)	32.033 (9)	27.961 (9)	12.285 (9)	2.808 (9)
Conference USA (18-36)	31.635 (10)	26.98 (10)	11.954 (10)	2.709 (10)

Table 22. 2006 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Big East (37-8)	56.416 (1)	49.583 (1)	23.306 (1)	5.561 (1)
SEC (47-10)	54.418 (2)	46.254 (2)	22.547 (2)	5.157 (2)
Big 10 (35-17)	49.726 (4)	42.756 (4)	20.427 (3)	4.763 (3)
Pac-10 (25-12)	47.957 (5)	41.154 (5)	19.682 (4)	4.626 (4)
Big 12 (36-20)	51.12 (3)	43.801 (3)	19.007 (5)	4.39 (5)
ACC (33-23)	46.802 (6)	39.247 (6)	18.464 (6)	4.201 (6)
Mountain West (19-21)	44.813 (7)	38.837 (7)	17.205 (7)	3.968 (7)
WAC (21-21)	39.445 (8)	33.455 (8)	16.574 (8)	3.782 (8)
FBS Independents (19-23)	37.372 (10)	31.135 (11)	15.33 (9)	3.484 (9)
Conference USA (21-32)	37.514 (9)	31.408 (10)	14.745 (10)	3.26 (10)

Table 23. 2005 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Big 10 (31-10)	59.456 (2)	52.779 (2)	24.908 (1)	5.864 (1)
ACC (32-12)	56.372 (4)	50.204 (4)	24.121 (3)	5.706 (2)
Pac-10 (26-10)	59.765 (1)	53.456 (1)	24.203 (2)	5.687 (3)
Big 12 (36-8)	58.414 (3)	52.196 (3)	23.714 (4)	5.616 (4)
FBS Independents (19-12)	55.741 (5)	49.991 (5)	23.588 (5)	5.612 (5)
SEC (30-12)	54.261 (6)	47.872 (6)	22.272 (6)	5.301 (6)
Big East (20-16)	53.412 (7)	47.859 (7)	20.438 (7)	4.899 (7)
Mountain West (15-17)	50.424 (8)	45.382 (8)	19.341 (8)	4.716 (8)
Conference USA (18-24)	42.823 (9)	38.227 (9)	16.985 (9)	3.954 (9)
WAC (10-24)	39.865 (10)	35.108 (10)	14.652 (10)	3.451 (10)

Table 24. 2004 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Pac-10 (23-13)	60.84 (1)	53.685 (1)	25.979 (1)	6.136 (1)
Big 10 (27-14)	57.126 (3)	50.272 (3)	24.274 (4)	5.776 (2)
FBS Independents (15-7)	55.278 (5)	48.193 (5)	24.998 (2)	5.699 (3)
ACC (27-13)	57.256 (2)	50.547 (2)	23.941 (5)	5.653 (4)
Big 12 (32-11)	55.468 (4)	48.525 (4)	24.371 (3)	5.568 (5)
SEC (28-14)	55.162 (6)	47.397 (6)	23.143 (6)	5.328 (6)
Mountain West (18-17)	50.882 (7)	44.37 (7)	21.873 (7)	5.041 (7)
Big East (24-16)	47.394 (8)	41.539 (8)	20.743 (8)	4.909 (8)
WAC (19-19)	44.652 (9)	38.737 (10)	19.023 (9)	4.317 (9)
Conference USA (17-21)	44.424 (10)	38.781 (9)	18.78 (10)	4.307 (10)

Table 25. 2003 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
ACC (27-15)	56.633 (1)	49.769 (1)	22.056 (1)	5.208 (1)
Big 10 (33-19)	55.183 (3)	47.549 (3)	21.876 (2)	5.002 (2)
SEC (37-19)	55.296 (2)	47.937 (2)	20.983 (4)	4.961 (3)
Big East (27-18)	53.423 (5)	46.721 (4)	21.215 (3)	4.936 (4)
Big 12 (39-18)	53.583 (4)	46.3 (5)	20.64 (6)	4.904 (5)
Pac-10 (29-17)	49.15 (6)	42.447 (6)	20.79 (5)	4.778 (6)
Mountain West (24-18)	44.687 (7)	38.627 (8)	17.558 (8)	4.177 (7)
FBS Independents (27-20)	44.259 (8)	38.908 (7)	17.666 (7)	4.123 (8)
Conference USA (24-25)	40.138 (9)	34.176 (9)	15.386 (9)	3.465 (9)
MAC (26-32)	38.358 (10)	33.295 (10)	13.548 (11)	3.196 (10)

Table 26. 2002 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Pac-10 (32-15)	51.605 (1)	43.603 (1)	21.094 (1)	4.891 (1)
Big 12 (41-18)	51.525 (2)	43.571 (2)	20.789 (2)	4.768 (2)
ACC (30-17)	47.437 (5)	39.907 (5)	20.046 (4)	4.675 (3)
SEC (40-16)	49.5 (3)	41.118 (3)	19.892 (5)	4.663 (4)
Big 10 (38-15)	48.626 (4)	39.909 (4)	20.54 (3)	4.646 (5)
Big East (30-16)	46.603 (6)	39.133 (6)	18.974 (6)	4.451 (6)
FBS Independents (32-33)	36.989 (8)	30.437 (8)	14.551 (7)	3.362 (7)
Mountain West (18-27)	37.695 (7)	31.278 (7)	14.232 (8)	3.335 (8)
Conference USA (17-28)	34.628 (9)	27.876 (9)	13.462 (9)	3.059 (9)
WAC (16-29)	31.931 (10)	25.435 (10)	12.803 (10)	2.885 (10)

Table 27. 2001 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Big 12 (36-10)	51.344 (1)	43.044 (1)	21.959 (2)	4.794 (1)
Big East (26-12)	49.877 (2)	42.079 (2)	21.503 (4)	4.775 (2)
SEC (34-10)	49.636 (3)	41.056 (3)	22.1 (1)	4.745 (3)
Pac-10 (26-9)	47.66 (4)	39.664 (5)	21.674 (3)	4.671 (4)
Big 10 (24-16)	47.573 (5)	39.863 (4)	20.666 (5)	4.434 (5)
ACC (23-13)	44.624 (6)	36.589 (6)	18.92 (6)	4.179 (6)
Mountain West (20-18)	42.271 (7)	35.25 (7)	16.716 (7)	3.763 (7)
MAC (20-25)	34.001 (10)	28.885 (9)	13.87 (10)	3.099 (8)
WAC (16-22)	34.745 (9)	28.182 (10)	14.138 (8)	3.048 (9)
Conference USA (18-28)	35.588 (8)	29.53 (8)	13.931 (9)	3.042 (10)

Table 28. 2000 Conference Results

Conference	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Big East (28-9)	40.656 (2)	33.586 (1)	17.491 (1)	3.677 (1)
Big 12 (32-13)	41.615 (1)	33.479 (2)	16.748 (2)	3.554 (2)
SEC (31-14)	39.444 (3)	31.815 (3)	16.044 (3)	3.418 (3)
Pac-10 (26-10)	35.854 (6)	27.863 (6)	15.315 (4)	3.2 (4)
ACC (20-13)	36.176 (5)	28.2 (5)	14.497 (5)	2.986 (5)
Big 10 (25-17)	36.503 (4)	28.8 (4)	14.006 (6)	2.928 (6)
Conference USA (22-19)	30.243 (7)	23.898 (7)	12.063 (7)	2.517 (7)
Mountain West (18-20)	29.343 (8)	23.124 (8)	11.182 (8)	2.343 (8)
Ohio AC (11-3)	13.925 (14)	13.743 (13)	5.875 (14)	1.86 (9)
Patriot League (23-14)	10.035 (19)	11.315 (17)	6.1 (13)	1.735 (10)

APPENDIX C. DIVISION RATINGS FROM 2000 – 2009

In college football, conferences are aligned to divisions based on a number of factors, most notably the number of scholarships awarded. In order to rate these divisions, the intra-division games were used as the observations, and the four rating systems were applied. In order to effectively analyze the four rating systems developed, I computed the results of all college football games for the years 2000 – 2009. Tables 29 – 38 summarize the results. The divisions are ordered by the Hensley Rating.

Table 29. 2009 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
FBS (89-5)	42.017 (1)	36.719 (1)	16.191 (1)	3.796 (1)
FCS (65-103)	12.921 (2)	11.484 (2)	5.011 (2)	1.18 (2)
NCAA Division II (61-59)	-4.801 (3)	-3.455 (3)	-1.87 (3)	-0.302 (3)
NCAA Division III (39-31)	-7.712 (4)	-6.287 (4)	-2.668 (4)	-0.638 (4)
NAIA (39-86)	-16.308 (5)	-14.331 (5)	-6.041 (5)	-1.448 (5)
Other (3-12)	-26.117 (6)	-24.13 (6)	-10.623 (6)	-2.588 (6)

Table 30. 2008 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (85-2)	41.959 (1)	35.324 (1)	17.223 (1)	3.902 (1)
Division I-AA (113-102)	13.085 (2)	11.582 (2)	4.97 (2)	1.156 (2)
Division I-AA Non-scholarship (38-59)	-4.54 (3)	-4.114 (3)	-2.094 (3)	-0.506 (3)
Division III (42-25)	-14.577 (5)	-12.573 (5)	-4.94 (4)	-1.095 (4)
Division II (51-71)	-11.712 (4)	-8.816 (4)	-5.274 (5)	-1.098 (5)
NAIA (31-101)	-24.215 (6)	-21.404 (6)	-9.884 (6)	-2.359 (6)

Table 31. 2007 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (71-9)	37.414 (1)	33.207 (1)	15.202 (1)	3.499 (1)
Division I-AA (86-84)	11.798 (2)	10.596 (2)	5.063 (2)	1.161 (2)
Division I-AA Non-scholarship (38-49)	-3.264 (3)	-2.877 (3)	-1.466 (3)	-0.345 (3)
Division II (62-74)	-11.623 (4)	-10.25 (4)	-4.104 (4)	-0.98 (4)
Division III (28-30)	-12.965 (5)	-11.288 (5)	-6.139 (5)	-1.308 (5)
NAIA (36-75)	-21.359 (6)	-19.388 (6)	-8.556 (6)	-2.027 (6)

Table 32. 2006 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (71-7)	39.414 (1)	35.306 (1)	16.181 (1)	4.082 (1)
Division I-AA (90-85)	12.696 (2)	11.759 (2)	5.387 (2)	1.331 (2)
Division II (75-68)	-8.346 (4)	-6.884 (3)	-2.985 (3)	-0.803 (3)
Division I-AA Non-scholarship (26-55)	-7.586 (3)	-7.13 (4)	-3.848 (4)	-0.972 (4)
Division III (37-26)	-13.248 (5)	-11.998 (5)	-4.962 (5)	-1.127 (5)
NAIA (24-82)	-22.93 (6)	-21.053 (6)	-9.773 (6)	-2.512 (6)

Table 33. 2005 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (52-2)	41.905 (1)	38.598 (1)	17.251 (1)	4.237 (1)
Division I-AA (80-63)	16.46 (2)	15.715 (2)	5.51 (2)	1.417 (2)
Division II (63-61)	-9.4 (3)	-8.227 (3)	-3.166 (3)	-0.724 (3)
Division I-AA Non-scholarship (23-51)	-9.776 (4)	-9.478 (4)	-5.108 (4)	-1.218 (4)
Division III (48-46)	-16.773 (5)	-15.668 (5)	-6.34 (5)	-1.628 (5)
NAIA (44-87)	-22.416 (6)	-20.94 (6)	-8.148 (6)	-2.084 (6)

Table 34. 2004 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (50-6)	38.173 (1)	33.27 (1)	16.198 (1)	3.687 (1)
Division I-AA (89-61)	16.637 (2)	15.543 (2)	5.751 (2)	1.463 (2)
Division I-AA Non-scholarship (23-41)	-5.575 (3)	-5.476 (3)	-2.86 (3)	-0.714 (3)
Division II (59-72)	-10.329 (4)	-8.688 (4)	-3.72 (4)	-0.775 (4)
Division III (41-36)	-15.65 (5)	-13.575 (5)	-6.257 (5)	-1.474 (5)
NAIA (40-86)	-23.256 (6)	-21.074 (6)	-9.112 (6)	-2.186 (6)

Table 35. 2003 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (62-10)	34.158 (1)	27.402 (1)	13.011 (1)	2.947 (1)
Division I-AA (91-95)	10.879 (2)	9.686 (2)	3.6 (2)	0.869 (2)
Division II (82-76)	-5.103 (4)	-2.591 (3)	-1.43 (3)	-0.19 (3)
Division I-AA Non-scholarship (35-45)	-5.082 (3)	-5.113 (4)	-1.606 (4)	-0.48 (4)
NAIA (51-85)	-17.578 (6)	-14.641 (5)	-6.259 (5)	-1.459 (5)
Division III (37-47)	-17.274 (5)	-14.743 (6)	-7.315 (6)	-1.687 (6)

Table 36. 2002 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (57-8)	33.786 (1)	26.284 (1)	13.754 (1)	3.136 (1)
Division I-AA (97-75)	9.754 (2)	8.244 (2)	4.011 (2)	0.972 (2)
Division II (65-96)	-8.436 (4)	-5.982 (3)	-3.457 (4)	-0.766 (3)
Division I-AA Non-scholarship (33-53)	-7.555 (3)	-7.052 (4)	-3.23 (3)	-0.864 (4)
NAIA (61-65)	-13.856 (6)	-10.715 (5)	-5.393 (5)	-1.168 (5)
Division III (40-56)	-13.692 (5)	-10.78 (6)	-5.686 (6)	-1.311 (6)

Table 37. 2001 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (49-8)	35.566 (1)	27.405 (1)	15.036 (1)	3.085 (1)
Division I-AA (83-75)	14.511 (2)	12.51 (2)	5.268 (2)	1.215 (2)
Division II (84-72)	-0.707 (3)	0.807 (3)	-0.436 (3)	0.026 (3)
Division I-AA Non-scholarship (35-53)	-1.382 (4)	-1.274 (4)	-1.932 (4)	-0.358 (4)
NAIA (64-95)	-11.388 (6)	-8.944 (6)	-4.484 (5)	-0.863 (5)
Division III (59-66)	-10.784 (5)	-8.455 (5)	-4.678 (6)	-0.917 (6)
Other (7-12)	-25.816 (7)	-22.049 (7)	-8.775 (7)	-2.188 (7)

Table 38. 2000 Division Results

Division	Standard Rating	HFA Rating	MPD=14 Rating	Hensley Rating
Division I-A (39-16)	22.76 (1)	15.566 (1)	8.408 (1)	1.415 (1)
Division I-AA Non-scholarship (56-34)	3.391 (3)	4.798 (2)	2.157 (2)	0.713 (2)
Division I-AA (97-85)	3.519 (2)	2.334 (3)	1.717 (3)	0.263 (3)
Division III (63-70)	-5.834 (4)	-3.843 (4)	-2.904 (5)	-0.445 (4)
NAIA (63-68)	-6.988 (5)	-4.925 (5)	-3.326 (6)	-0.577 (5)
Division II (66-109)	-8.546 (7)	-5.726 (6)	-3.546 (7)	-0.631 (6)
Other (9-11)	-8.303 (6)	-8.205 (7)	-2.506 (4)	-0.737 (7)

APPENDIX D. CODE

The entire solution was written in C# inside Visual Studio, with a SQL Server database. The solution consisted of four projects – a database interaction layer, a data importer, the rating system calculator, and a ratings evaluator. The ADO.NET Entity Framework was used to connect to the database. The following pages display the code for the entire solution.

Database Layer Project

```
/* CollegeFootballEntities.cs
 *
 * Joel Hensley
 * January 13, 2010
 *
 * This class is used to add addtional methods and properties
 * to the CollegeFootballEntities object.
 */

using System;
using System.Collections.Generic;
using System.Linq;

namespace DatabaseLayer
{
    partial class CollegeFootballEntities
    {
        /// <summary>
        /// Gets all the games played between different conferences
        /// </summary>
        /// <param name="conferenceGroup">
        /// The group of conferences being examined
        /// </param>
        /// <returns>The interconference games</returns>
        public IEnumerable<Game> GetInterConferenceGames(int conferenceGroup)
        {
            IEnumerable<Game> interConferenceGames = null;

            interConferenceGames = from g in Games
                                   join ht in Teams on g.HomeTeamID equals ht.ID
                                   join at in Teams on g.AwayTeamID equals at.ID
                                   join hc in Conferences on ht.ConferenceID equals hc.ID
                                   where ht.ConferenceID != at.ConferenceID
                                   && hc.Group == conferenceGroup
                                   select g;
```

```
    return interConferenceGames;
}

/// <summary>
/// Gets all the games played between different divisions
/// </summary>
/// <param name="divisionGroup">
/// The group of divisions being examined
/// </param>
/// <returns>The interdivision games</returns>
public IEnumerable<Game> GetInterDivisionGames(int divisionGroup)
{
    IEnumerable<Game> interDivisionGames = null;

    interDivisionGames = from g in Games
                join ht in Teams on g.HomeTeamID equals ht.ID
                join at in Teams on g.AwayTeamID equals at.ID
                join hc in Conferences on ht.ConferenceID equals hc.ID
                join ac in Conferences on at.ConferenceID equals ac.ID
                join hd in Divisions on hc.DivisionID equals hd.ID
                where hc.DivisionID != ac.DivisionID
                && hd.Group == divisionGroup
                select g;

    return interDivisionGames;
}

/// <summary>
/// Gets the number of wins for a given conference in a given set of games
/// </summary>
/// <param name="interConferenceGames">
/// The set of interconference games
/// </param>
/// <param name="conference">The conference being examined</param>
/// <returns>The total number of wins</returns>
public int GetWins(IEnumerable<Game> interConferenceGames,
                Conference conference)
{
    int wins;

    wins = (from g in interConferenceGames
            join ht in Teams on g.HomeTeamID equals ht.ID
            join at in Teams on g.AwayTeamID equals at.ID
            where (ht.ConferenceID == conference.ID && g.HomeScore >
                g.AwayScore)
                || (at.ConferenceID == conference.ID && g.AwayScore >
                 g.HomeScore)
            select g.ID).Count();

    return wins;
}

/// <summary>
/// Gets the number of losses for a given conference in a given set of games
/// </summary>
/// <param name="interConferenceGames">
```

```
/// The set of interconference games
/// </param>
/// <param name="conference">The conference being examined</param>
/// <returns>The total number of losses</returns>
public int GetLosses(IEnumerable<Game> interConferenceGames,
                     Conference conference)
{
    int losses;

    losses = (from g in interConferenceGames
              join ht in Teams on g.HomeTeamID equals ht.ID
              join at in Teams on g.AwayTeamID equals at.ID
              where (ht.ConferenceID == conference.ID && g.HomeScore <
                     g.AwayScore)
                  || (at.ConferenceID == conference.ID && g.AwayScore <
                      g.HomeScore)
              select g.ID).Count();

    return losses;
}

/// <summary>
/// Gets the number of wins for a given division in a given set of games
/// </summary>
/// <param name="interConferenceGames">
/// The set of interdivision games
/// </param>
/// <param name="conference">The division being examined</param>
/// <returns>The total number of wins</returns>
public int GetWins(IEnumerable<Game> interDivisionGames, Division division)
{
    int wins;

    wins = (from g in interDivisionGames
            join ht in Teams on g.HomeTeamID equals ht.ID
            join at in Teams on g.AwayTeamID equals at.ID
            join hc in Conferences on ht.ConferenceID equals hc.ID
            join ac in Conferences on at.ConferenceID equals ac.ID
            where (hc.DivisionID == division.ID && g.HomeScore >
                   g.AwayScore)
                || (ac.DivisionID == division.ID && g.AwayScore >
                    g.HomeScore)
            select g.ID).Count();

    return wins;
}

/// <summary>
/// Gets the number of losses for a given division in a given set of games
/// </summary>
/// <param name="interDivisionGames">The set of interdivision games</param>
/// <param name="division">The division being examined</param>
/// <returns>The total number of losses</returns>
public int GetLosses(IEnumerable<Game> interDivisionGames,
                     Division division)
{
```

```
    int losses;

    losses = (from g in interDivisionGames
            join ht in Teams on g.HomeTeamID equals ht.ID
            join at in Teams on g.AwayTeamID equals at.ID
            join hc in Conferences on ht.ConferenceID equals hc.ID
            join ac in Conferences on at.ConferenceID equals ac.ID
            where (hc.DivisionID == division.ID && g.HomeScore <
                   g.AwayScore)
                || (ac.DivisionID == division.ID && g.AwayScore <
                    g.HomeScore)
            select g.ID).Count();

    return losses;
}

/// <summary>
/// Gets the total point differential for a conference in a given set of
/// games
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="conference">The conference</param>
/// <returns>The total point differential</returns>
public int GetPointDifferential(IEnumerable<Game> games,
                                Conference conference)
{
    return GetPointDifferential(games, conference, 0);
}

/// <summary>
/// Gets the total point differential adjusted for the maximum point
/// differential
/// per game for a conference in a given set of games
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="conference">The conference</param>
/// <param name="maxPointDifferential">
/// The maximum point differential
/// </param>
/// <returns>The total point differential</returns>
/// <remarks>
/// If the maximum point differential is less than or equal to zero, then
/// no adjustment is made to the total point differential.
/// </remarks>
public int GetPointDifferential(IEnumerable<Game> games,
                                Conference conference,
                                int maxPointDifferential)
{
    int gameDifferential;
    int calculatedPointDifferential = 0;

    var homeGameScores = from g in games
                         join ht in Teams on g.HomeTeamID equals ht.ID
                         where ht.ConferenceID == conference.ID
                         select new { g.HomeScore, g.AwayScore };
    var awayGameScores = from g in games
```

```
                        join at in Teams on g.AwayTeamID equals at.ID
                        where at.ConferenceID == conference.ID
                        select new { g.HomeScore, g.AwayScore };

    if (maxPointDifferential > 0)
    {
        foreach (var homeGameScore in homeGameScores)
        {
            gameDifferential = homeGameScore.HomeScore -
                               homeGameScore.AwayScore;
            gameDifferential = (Math.Abs(gameDifferential) >
                                maxPointDifferential)
                    ? maxPointDifferential * Math.Sign(gameDifferential)
                    : gameDifferential;
            calculatedPointDifferential += gameDifferential;
        }

        foreach (var awayGameScore in awayGameScores)
        {
            gameDifferential = awayGameScore.AwayScore -
                               awayGameScore.HomeScore;
            gameDifferential = (Math.Abs(gameDifferential) >
                                maxPointDifferential)
                    ? maxPointDifferential * Math.Sign(gameDifferential)
                    : gameDifferential;
            calculatedPointDifferential += gameDifferential;
        }
    }
    else
    {
        calculatedPointDifferential = (homeGameScores.Sum(h => h.HomeScore)
                                    + awayGameScores.Sum(h => h.AwayScore))
                                    - (homeGameScores.Sum(h => h.AwayScore)
                                    + awayGameScores.Sum(h => h.HomeScore));
    }

    return calculatedPointDifferential;
}

/// <summary>
/// Gets the total computed win-loss value for a conference in a given
/// set of games using a specified upper and lower bounds.
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="conference">The conference</param>
/// <param name="lowerBounds">The lower bounds to the win-loss value</param>
/// <param name="upperBounds">The upper bounds to the win-loss value</param>
/// <returns>The total computed win-loss value</returns>
public double GetHensleyPointDifferential(IEnumerable<Game> games,
            Conference conference, double lowerBounds, double upperBounds)
{
    bool isWinner;
    double winnerScore;
    double loserScore;
    double computedDifferentialScore;
    double cumulativeTotal = 0;
```

```
var homeGameScores = from g in games
                     join ht in Teams on g.HomeTeamID equals ht.ID
                     where ht.ConferenceID == conference.ID
                     select new { g.HomeScore, g.AwayScore };
var awayGameScores = from g in games
                     join at in Teams on g.AwayTeamID equals at.ID
                     where at.ConferenceID == conference.ID
                     select new { g.HomeScore, g.AwayScore };

foreach (var gameScore in homeGameScores)
{
    isWinner = (gameScore.HomeScore > gameScore.AwayScore);
    if (isWinner)
    {
        winnerScore = (double)gameScore.HomeScore;
        loserScore = (double)gameScore.AwayScore;
    }
    else
    {
        winnerScore = (double)gameScore.AwayScore;
        loserScore = (double)gameScore.HomeScore;
    }

    computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                     winnerScore, loserScore, lowerBounds, upperBounds);

    if (!isWinner)
    {
        // The loser receives the negative value of this computation
        computedDifferentialScore *= -1;
    }

    cumulativeTotal += computedDifferentialScore;
}

foreach (var gameScore in awayGameScores)
{
    isWinner = (gameScore.AwayScore > gameScore.HomeScore);
    if (isWinner)
    {
        winnerScore = (double)gameScore.AwayScore;
        loserScore = (double)gameScore.HomeScore;
    }
    else
    {
        winnerScore = (double)gameScore.HomeScore;
        loserScore = (double)gameScore.AwayScore;
    }

    computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                     winnerScore, loserScore, lowerBounds, upperBounds);

    if (!isWinner)
    {
        // The loser receives the negative value of this computation
```

```
                computedDifferentialScore *= -1;
            }

            cumulativeTotal += computedDifferentialScore;
        }

        return cumulativeTotal;
    }

    /// <summary>
    /// Gets the total point differential for a division in a given set of games
    /// </summary>
    /// <param name="games">The set of games</param>
    /// <param name="division">The division</param>
    /// <returns>The total point differential</returns>
    public int GetPointDifferential(IEnumerable<Game> games, Division division)
    {
        return GetPointDifferential(games, division, 0);
    }

    /// <summary>
    /// Gets the total point differential adjusted for the maximum point
    /// differential
    /// per game for a division in a given set of games
    /// </summary>
    /// <param name="games">The set of games</param>
    /// <param name="division">The division</param>
    /// <param name="maxPointDifferential">
    /// The maximum point differential
    /// </param>
    /// <returns>The total point differential</returns>
    /// <remarks>
    /// If the maximum point differential is less than or equal to zero, then
    /// no adjustment is made to the total point differential.
    /// </remarks>
    public int GetPointDifferential(IEnumerable<Game> games, Division division,
                                    int maxPointDifferential)
    {
        int gameDifferential;
        int calculatedPointDifferential = 0;

        var homeGameScores = from g in games
                             join ht in Teams on g.HomeTeamID equals ht.ID
                             join hc in Conferences on ht.ConferenceID equals hc.ID
                             where hc.DivisionID == division.ID
                             select new { g.HomeScore, g.AwayScore };
        var awayGameScores = from g in games
                             join at in Teams on g.AwayTeamID equals at.ID
                             join ac in Conferences on at.ConferenceID equals ac.ID
                             where ac.DivisionID == division.ID
                             select new { g.HomeScore, g.AwayScore };

        if (maxPointDifferential > 0)
        {
            foreach (var homeGameScore in homeGameScores)
            {
```

```
                gameDifferential = homeGameScore.HomeScore -
                                   homeGameScore.AwayScore;
                gameDifferential = (Math.Abs(gameDifferential) >
                                    maxPointDifferential)
                           ? maxPointDifferential * Math.Sign(gameDifferential)
                           : gameDifferential;
                calculatedPointDifferential += gameDifferential;
            }

            foreach (var awayGameScore in awayGameScores)
            {
                gameDifferential = awayGameScore.AwayScore -
                                   awayGameScore.HomeScore;
                gameDifferential = (Math.Abs(gameDifferential) >
                                     maxPointDifferential)
                           ? maxPointDifferential * Math.Sign(gameDifferential)
                           : gameDifferential;
                calculatedPointDifferential += gameDifferential;
            }
        }
        else
        {
            calculatedPointDifferential = (homeGameScores.Sum(h => h.HomeScore)
                                        + awayGameScores.Sum(h => h.AwayScore))
                                        - (homeGameScores.Sum(h => h.AwayScore)
                                        + awayGameScores.Sum(h => h.HomeScore));
        }

        return calculatedPointDifferential;
    }

    /// <summary>
    /// Gets the total computed win-loss value for a division in a given
    /// set of games using a specified upper and lower bounds.
    /// </summary>
    /// <param name="games">The set of games</param>
    /// <param name="division">The division</param>
    /// <param name="lowerBounds">The lower bounds to the win-loss value</param>
    /// <param name="upperBounds">The upper bounds to the win-loss value</param>
    /// <returns>The total computed win-loss value</returns>
    public double GetHensleyPointDifferential(IEnumerable<Game> games,
                      Division division, double lowerBounds, double upperBounds)
    {
        bool isWinner;
        double winnerScore;
        double loserScore;
        double computedDifferentialScore;
        double cumulativeTotal = 0;

        var homeGameScores = from g in games
                       join ht in Teams on g.HomeTeamID equals ht.ID
                       join hc in Conferences on ht.ConferenceID equals hc.ID
                       where hc.DivisionID == division.ID
                       select new { g.HomeScore, g.AwayScore };
        var awayGameScores = from g in games
                       join at in Teams on g.AwayTeamID equals at.ID
```

```
                join ac in Conferences on at.ConferenceID equals ac.ID
                where ac.DivisionID == division.ID
                select new { g.HomeScore, g.AwayScore };

foreach (var gameScore in homeGameScores)
{
    isWinner = (gameScore.HomeScore > gameScore.AwayScore);
    if (isWinner)
    {
        winnerScore = (double)gameScore.HomeScore;
        loserScore = (double)gameScore.AwayScore;
    }
    else
    {
        winnerScore = (double)gameScore.AwayScore;
        loserScore = (double)gameScore.HomeScore;
    }

    computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                    winnerScore, loserScore, lowerBounds, upperBounds);

    if (!isWinner)
    {
        // The loser receives the negative value of this computation
        computedDifferentialScore *= -1;
    }

    cumulativeTotal += computedDifferentialScore;
}

foreach (var gameScore in awayGameScores)
{
    isWinner = (gameScore.AwayScore > gameScore.HomeScore);
    if (isWinner)
    {
        winnerScore = (double)gameScore.AwayScore;
        loserScore = (double)gameScore.HomeScore;
    }
    else
    {
        winnerScore = (double)gameScore.HomeScore;
        loserScore = (double)gameScore.AwayScore;
    }

    computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                     winnerScore, loserScore, lowerBounds, upperBounds);

    if (!isWinner)
    {
        // The loser receives the negative value of this computation
        computedDifferentialScore *= -1;
    }

    cumulativeTotal += computedDifferentialScore;
}
```

```
    return cumulativeTotal;
}

/// <summary>
/// Gets the total number of home games for a conference minues the
/// total number of away games for a conference.
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="conference">The conference</param>
/// <returns>The home game differential</returns>
public int GetHomeGameDifferential(IEnumerable<Game> games,
                                   Conference conference)
{
    int homeGameCount = (from g in games
                         join ht in Teams on g.HomeTeamID equals ht.ID
                         where ht.ConferenceID == conference.ID
                         && g.IsNeutralSite == false
                         select g.ID).Count();
    int awayGameCount = (from g in games
                         join at in Teams on g.AwayTeamID equals at.ID
                         where at.ConferenceID == conference.ID
                         && g.IsNeutralSite == false
                         select g.ID).Count();

    return homeGameCount - awayGameCount;
}

/// <summary>
/// Gets the total number of home games for a division minues the
/// total number of away games for a division.
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="division">The division</param>
/// <returns>The home game differential</returns>
public int GetHomeGameDifferential(IEnumerable<Game> games,
                                   Division division)
{
    int homeGameCount = (from g in games
                   join ht in Teams on g.HomeTeamID equals ht.ID
                   join hc in Conferences on ht.ConferenceID equals hc.ID
                   where hc.DivisionID == division.ID
                   && g.IsNeutralSite == false
                   select g.ID).Count();
    int awayGameCount = (from g in games
                   join at in Teams on g.AwayTeamID equals at.ID
                   join ac in Conferences on at.ConferenceID equals ac.ID
                   where ac.DivisionID == division.ID
                   && g.IsNeutralSite == false
                   select g.ID).Count();

    return homeGameCount - awayGameCount;
}

/// <summary>
/// Gets all the games for all the teams in a conference
/// </summary>
```

```
/// <param name="conference">The conference</param>
/// <returns>All games played by that conference</returns>
public IEnumerable<Game> GetGames(Conference conference)
{
    IEnumerable<Game> conferenceGames = null;

    conferenceGames = from g in Games
                      join ht in Teams on g.HomeTeamID equals ht.ID
                      join at in Teams on g.AwayTeamID equals at.ID
                      where (ht.ConferenceID == conference.ID
                             && at.ConferenceID != conference.ID)
                      || (ht.ConferenceID != conference.ID
                          && at.ConferenceID == conference.ID)
                      select g;

    return conferenceGames;
}

/// <summary>
/// Gets all the games for all the teams in a division
/// </summary>
/// <param name="division">The division</param>
/// <returns>All games played by that division</returns>
public IEnumerable<Game> GetGames(Division division)
{
    IEnumerable<Game> divisionGames = null;

    divisionGames = from g in Games
                    join ht in Teams on g.HomeTeamID equals ht.ID
                    join at in Teams on g.AwayTeamID equals at.ID
                    join hc in Conferences on ht.ConferenceID equals hc.ID
                    join ac in Conferences on at.ConferenceID equals ac.ID
                    where (hc.DivisionID == division.ID
                           && ac.DivisionID != division.ID)
                    || (hc.DivisionID != division.ID
                        && ac.DivisionID == division.ID)
                    select g;

    return divisionGames;
}

/// <summary>
/// Gets all the games played by teams in a given group
/// </summary>
/// <param name="groupNum">The group number</param>
/// <returns>All games played by that team group</returns>
public IEnumerable<Game> GetGames(int groupNum)
{
    IEnumerable<Game> groupGames = null;

    groupGames = from g in Games
                 join ht in Teams on g.HomeTeamID equals ht.ID
                 where ht.Group == groupNum
                 select g;

    return groupGames;
```

```
}

/// <summary>
/// Gets the total number of home games in a set of games
/// </summary>
/// <param name="games">The set of games</param>
/// <returns>The total number of home games</returns>
public int GetHomeGameCount(IEnumerable<Game> games)
{
    int conferenceHomeGameCount = (from g in games
                                   where g.IsNeutralSite == false
                                   select g.ID).Count();

    return conferenceHomeGameCount;
}

/// <summary>
/// Gets the total point differential of all home games in a given
/// set of games
/// </summary>
/// <param name="games">The set of games</param>
/// <returns>The total point differential</returns>
public int GetHomeGamePointDifferential(IEnumerable<Game> games)
{
    var homeGames = from g in games
                    where g.IsNeutralSite == false
                    select new { g.HomeScore, g.AwayScore };

    return homeGames.Sum(g => g.HomeScore)
           - homeGames.Sum(g => g.AwayScore);
}

/// <summary>
/// Gets the total computed win-loss value of all home games in a given
/// set of games using a specified upper and lower bounds.
/// </summary>
/// <param name="games">The set of games</param>
/// <param name="lowerBounds">The lower bounds to the win-loss value</param>
/// <param name="upperBounds">The upper bounds to the win-loss value</param>
/// <returns>The total computed win-loss value</returns>
public double GetHomeGameHensleyPointDifferential(IEnumerable<Game> games,
                                     double lowerBounds, double upperBounds)
{
    bool isWinner;
    double winnerScore;
    double loserScore;
    double computedDifferentialScore;
    double cumulativeTotal = 0;

    var homeGames = from g in games
                    where g.IsNeutralSite == false
                    select new { g.HomeScore, g.AwayScore };

    foreach (var gameScore in homeGames)
    {
        isWinner = (gameScore.HomeScore > gameScore.AwayScore);
```

```
                if (isWinner)
                {
                    winnerScore = (double)gameScore.HomeScore;
                    loserScore = (double)gameScore.AwayScore;
                }
                else
                {
                    winnerScore = (double)gameScore.AwayScore;
                    loserScore = (double)gameScore.HomeScore;
                }

                computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                                 winnerScore, loserScore, lowerBounds, upperBounds);

                if (!isWinner)
                {
                    // The loser receives the negative value of this computation
                    computedDifferentialScore *= -1;
                }

                cumulativeTotal += computedDifferentialScore;
            }

            return cumulativeTotal;
        }
    }
}

/* Game.cs
 *
 * Joel Hensley
 * January 18, 2010
 *
 * This class is used to add addtional methods and properties
 * to the Game object.
 */

using System;

namespace DatabaseLayer
{
    partial class Game
    {
        private int homeConferenceID = -1;
        /// <summary>
        /// Gets the conference ID of the home team
        /// </summary>
        public int HomeConferenceID
        {
            get
            {
                if (homeConferenceID != -1)
                {
                    return homeConferenceID;
                }
                else
```

```
        {
            if (!HomeTeamReference.IsLoaded)
            {
                HomeTeamReference.Load();
            }
            homeConferenceID = HomeTeam.ConferenceID;
            return homeConferenceID;
        }
    }
}

private int awayConferenceID = -1;
/// <summary>
/// Gets the conference ID of the away team
/// </summary>
public int AwayConferenceID
{
    get
    {
        if (awayConferenceID != -1)
        {
            return awayConferenceID;
        }
        else
        {
            if (!AwayTeamReference.IsLoaded)
            {
                AwayTeamReference.Load();
            }
            awayConferenceID = AwayTeam.ConferenceID;
            return awayConferenceID;
        }
    }
}

private int homeDivisionID = -1;
/// <summary>
/// Gets the division ID of the home team
/// </summary>
public int HomeDivisionID
{
    get
    {
        if (homeDivisionID != -1)
        {
            return homeDivisionID;
        }
        else
        {
            if (!HomeTeamReference.IsLoaded)
            {
                HomeTeamReference.Load();
            }
            if (!HomeTeam.ConferenceReference.IsLoaded)
            {
                HomeTeam.ConferenceReference.Load();
```

```
            }
            homeDivisionID = HomeTeam.Conference.DivisionID;
            return homeDivisionID;
        }
    }
}

private int awayDivisionID = -1;
/// <summary>
/// Gets the division ID of the away team
/// </summary>
public int AwayDivisionID
{
    get
    {
        if (awayDivisionID != -1)
        {
            return awayDivisionID;
        }
        else
        {
            if (!AwayTeamReference.IsLoaded)
            {
                AwayTeamReference.Load();
            }
            if (!AwayTeam.ConferenceReference.IsLoaded)
            {
                AwayTeam.ConferenceReference.Load();
            }
            awayDivisionID = AwayTeam.Conference.DivisionID;
            return awayDivisionID;
        }
    }
}

/// <summary>
/// Gets the computed win-loss value for a game
/// </summary>
/// <param name="winnerScore">The winner score</param>
/// <param name="loserScore">The loser score</param>
/// <param name="lowerBounds">
/// The lower bounds for the win-loss value
/// </param>
/// <param name="upperBounds">
/// The upper bounds for the win-loss value
/// </param>
/// <returns>The computed win-loss value</returns>
public static double GetHensleyPointDifferentialScore(double winnerScore,
                double loserScore, double lowerBounds, double upperBounds)
{
    double gameDifferential;
    double winnerLoserRatio;
    double computedDifferentialScore;

    gameDifferential = winnerScore - loserScore;
    winnerLoserRatio = loserScore / winnerScore;
```

```
            computedDifferentialScore = Math.Sqrt((1 - winnerLoserRatio)
                                                  * gameDifferential);

            if (computedDifferentialScore < lowerBounds)
            {
                // Adjust if below lower bounds
                computedDifferentialScore = lowerBounds;
            }
            else if (computedDifferentialScore > upperBounds)
            {
                // Adjust if above upper bounds
                computedDifferentialScore = upperBounds;
            }

            return computedDifferentialScore;
        }
    }
}

/* Team.cs
 *
 * Joel Hensley
 * January 16, 2010
 *
 * This class is used to add addtional methods and properties
 * to the Team object.
 */

using System;
using System.Collections.Generic;
using System.Linq;

namespace DatabaseLayer
{
    partial class Team
    {
        private bool isHomeGameDifferentialSet = false;
        private int homeGameDifferential = 0;

        /// <summary>
        /// Gets the total number of home games minues
        /// the total number of away games
        /// </summary>
        public int HomeGameDifferential
        {
            get
            {
                if (isHomeGameDifferentialSet)
                {
                    return homeGameDifferential;
                }
                else
                {
                    isHomeGameDifferentialSet = true;

                    if (!HomeGames.IsLoaded)
```

```
                {
                    HomeGames.Load();
                }
                if (!AwayGames.IsLoaded)
                {
                    AwayGames.Load();
                }

                homeGameDifferential = HomeGames.Where(g => g.IsNeutralSite
                                                  == false).Count()
                    - AwayGames.Where(g => g.IsNeutralSite == false).Count();

                return homeGameDifferential;
            }
        }
    }

    private int pointDifferential = 0;
    private bool isPointDifferentialSet = false;

    /// <summary>
    /// Gets the total point differential
    /// </summary>
    public int PointDifferential
    {
        get
        {
            if (isPointDifferentialSet)
            {
                return pointDifferential;
            }
            else
            {
                isPointDifferentialSet = true;

                if (!HomeGames.IsLoaded)
                {
                    HomeGames.Load();
                }
                if (!AwayGames.IsLoaded)
                {
                    AwayGames.Load();
                }

                foreach (Game game in HomeGames)
                {
                    pointDifferential += (game.HomeScore - game.AwayScore);
                }

                foreach (Game game in AwayGames)
                {
                    pointDifferential += (game.AwayScore - game.HomeScore);
                }

                return pointDifferential;
            }
```

```
    }
}

/// <summary>
/// Gets the total point differential using a specified
/// max point differential
/// </summary>
/// <param name="maxPointDifferential">
/// The maximum point differential for each game
/// </param>
/// <returns>The total point differential</returns>
public int GetPointDifferential(int maxPointDifferential)
{
    int gameDifferential;
    int calculatedPointDifferential = 0;

    if (!HomeGames.IsLoaded)
    {
        HomeGames.Load();
    }
    if (!AwayGames.IsLoaded)
    {
        AwayGames.Load();
    }

    foreach (Game game in HomeGames)
    {
        gameDifferential = game.HomeScore - game.AwayScore;
        gameDifferential = (Math.Abs(gameDifferential) >
                            maxPointDifferential)
                         ? maxPointDifferential * Math.Sign(gameDifferential)
                         : gameDifferential;
        calculatedPointDifferential += gameDifferential;
    }

    foreach (Game game in AwayGames)
    {
        gameDifferential = game.AwayScore - game.HomeScore;
        gameDifferential = (Math.Abs(gameDifferential) >
                            maxPointDifferential)
                         ? maxPointDifferential * Math.Sign(gameDifferential)
                         : gameDifferential;
        calculatedPointDifferential += gameDifferential;
    }

    return calculatedPointDifferential;
}

/// <summary>
/// Gets the total computed win-loss value using a specified
/// upper and lower bounds.
/// </summary>
/// <param name="lowerBounds">The lower bounds to the win-loss value</param>
/// <param name="upperBounds">The upper bounds to the win-loss value</param>
/// <returns>The total computed win-loss value</returns>
public double GetHensleyPointDifferential(double lowerBounds,
```

```
                                                double upperBounds)
{
    bool isWinner;
    double winnerScore;
    double loserScore;
    double computedDifferentialScore;
    double cumulativeTotal = 0;

    if (!HomeGames.IsLoaded)
    {
        HomeGames.Load();
    }
    if (!AwayGames.IsLoaded)
    {
        AwayGames.Load();
    }

    foreach (Game game in HomeGames)
    {
        isWinner = (game.HomeScore > game.AwayScore);
        if (isWinner)
        {
            winnerScore = (double)game.HomeScore;
            loserScore = (double)game.AwayScore;
        }
        else
        {
            winnerScore = (double)game.AwayScore;
            loserScore = (double)game.HomeScore;
        }

        computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                         winnerScore, loserScore, lowerBounds, upperBounds);

        if (!isWinner)
        {
            // The loser receives the negative value of this computation
            computedDifferentialScore *= -1;
        }

        cumulativeTotal += computedDifferentialScore;
    }

    foreach (Game game in AwayGames)
    {
        isWinner = (game.AwayScore > game.HomeScore);
        if (isWinner)
        {
            winnerScore = (double)game.AwayScore;
            loserScore = (double)game.HomeScore;
        }
        else
        {
            winnerScore = (double)game.HomeScore;
            loserScore = (double)game.AwayScore;
        }
```

```
            computedDifferentialScore = Game.GetHensleyPointDifferentialScore(
                                winnerScore, loserScore, lowerBounds, upperBounds);

            if (!isWinner)
            {
                // The loser receives the negative value of this computation
                computedDifferentialScore *= -1;
            }

            cumulativeTotal += computedDifferentialScore;
        }

        return cumulativeTotal;
    }

    private IEnumerable<Game> games = null;

    /// <summary>
    /// Gets the union of all away games and home games
    /// </summary>
    public IEnumerable<Game> Games
    {
        get
        {
            if (games != null)
            {
                return games;
            }
            else
            {
                if (!HomeGames.IsLoaded)
                {
                    HomeGames.Load();
                }
                if (!AwayGames.IsLoaded)
                {
                    AwayGames.Load();
                }

                games = HomeGames.Union(AwayGames).OrderBy(g => g.Date);
                return games;
            }
        }
    }

    private int wins = -1;
    /// <summary>
    /// Gets the number of games won
    /// </summary>
    public int Wins
    {
        get
        {
            if (wins != -1)
            {
```

```
                return wins;
            }
            else
            {
                if (!HomeGames.IsLoaded)
                {
                    HomeGames.Load();
                }
                if (!AwayGames.IsLoaded)
                {
                    AwayGames.Load();
                }

                wins = HomeGames.Where(g => g.HomeScore > g.AwayScore).Count()
                     + AwayGames.Where(g => g.AwayScore > g.HomeScore).Count();
                return wins;
            }
        }
    }

    private int losses = -1;
    /// <summary>
    /// Gets the number of games lost
    /// </summary>
    public int Losses
    {
        get
        {
            if (losses != -1)
            {
                return losses;
            }
            else
            {
                if (!HomeGames.IsLoaded)
                {
                    HomeGames.Load();
                }
                if (!AwayGames.IsLoaded)
                {
                    AwayGames.Load();
                }

                losses = HomeGames.Where(g => g.HomeScore < g.AwayScore).Count()
                      + AwayGames.Where(g => g.AwayScore < g.HomeScore).Count();
                return losses;
            }
        }
    }

    private IEnumerable<Team> opponents = null;
    /// <summary>
    /// Gets all opponents
    /// </summary>
    public IEnumerable<Team> Opponents
    {
```

```
            get
            {
                if (opponents != null)
                {
                    return opponents;
                }
                else
                {
                    if (!HomeGames.IsLoaded)
                    {
                        HomeGames.Load();
                    }
                    if (!AwayGames.IsLoaded)
                    {
                        AwayGames.Load();
                    }

                    opponents = HomeGames.Select(g => g.AwayTeam
                                ).Union(AwayGames.Select(g => g.HomeTeam));

                    return opponents;
                }
            }
        }
    }
}
```

Database Import Project

```
/* Program.cs
 *
 * Joel Hensley
 * January 13, 2010
 *
 * This class is used to remove any previous data from the database,
 * import the teams, conferences, divisions, and games into the
 * database, and finally create the different groups for the teams,
 * conferences, and divisions.
 */

using System;
using System.Linq;
using System.IO;
using DatabaseLayer;

namespace DataImport
{
    class Program
    {
        private CollegeFootballEntities entities = null;
        private static ImportSettings settings = new ImportSettings();

        /// <summary>
        /// Default constructor
```

```
/// </summary>
public Program()
{
    entities = new CollegeFootballEntities();
}

/// <summary>
/// Calls run
/// </summary>
/// <param name="args">No args are required</param>
static void Main(string[] args)
{
    Program p = new Program();
    p.Run();
}

/// <summary>
/// Removes any previous data in the database, import the teams, conferences
/// and divisions, imports the games, and the creates the groups for the
/// teams, conferences, and divisions.
/// </summary>
public void Run()
{
    TeamGraph tg = new TeamGraph(entities);
    ConferenceGraph cg = new ConferenceGraph(entities);
    DivisionGraph dg = new DivisionGraph(entities);

    try
    {
        if (settings.DeleteExistingData)
        {
            Console.Write("Removing previous data...");
            DeleteExistingData();
            Console.WriteLine("DONE");
        }

        if (settings.ImportTeams)
        {
            Console.Write("Creating divisions, conferences, and teams...");
            ImportTeams();
            Console.WriteLine("DONE");
        }

        if (settings.ImportGames)
        {
            Console.Write("Importing games...");
            ImportGames();
            Console.WriteLine("DONE");
        }

        if (settings.CreateGroups)
        {
            Console.Write("Creating team groups...");
            tg.CreateTeamGroups();
            Console.WriteLine("DONE");
```

```
                Console.Write("Creating conference groups...");
                cg.CreateConferenceGroups();
                Console.WriteLine("DONE");

                Console.Write("Creating division groups...");
                dg.CreateDivisionGroups();
                Console.WriteLine("DONE");
            }

            Console.WriteLine("\nData import complete");
        }
        catch (Exception)
        {
            Console.WriteLine("Error during data import");
        }

        Console.WriteLine("\nPress any key to continue");
        Console.ReadKey();
    }

    /// <summary>
    /// Removes all data in the database.
    /// </summary>
    private void DeleteExistingData()
    {
        entities.DeleteAllRows();
    }

    /// <summary>
    /// Imports the teams, conferences, and divisions.
    /// </summary>
    /// <remarks>
    /// The comma-delimited file must have the following
    /// format: divisionName,conferenceName,teamName
    /// </remarks>
    private void ImportTeams()
    {
        StreamReader reader = new StreamReader(settings.TeamsFileName);
        Division division;
        Conference conference;
        Team team;
        string[] row;
        string divisionName;
        string conferenceName;
        string teamName;

        while (reader.EndOfStream == false)
        {
            String line = reader.ReadLine();

            // CSV Format: <Division name>,<Conference name>,<Team name>
            row = line.Split(',');

            if (row.Length != 3)
            {
                Console.WriteLine(String.Format("Error in line: {0}", line));
```

```
                continue;
            }

            divisionName = row[0];
            conferenceName = row[1];
            teamName = row[2];

            division = entities.Divisions.FirstOrDefault(d => d.Name ==
                                                              divisionName);

            if (division == null)
            {
                // Division does not exist, so create one
                division = new Division();
                division.Name = divisionName;
                entities.Divisions.AddObject(division);
                entities.SaveChanges();
                entities.Refresh(System.Data.Objects.RefreshMode.StoreWins,
                                 division);
            }

            conference = entities.Conferences.FirstOrDefault(c => c.Name ==
                               conferenceName && c.DivisionID == division.ID);

            if (conference == null)
            {
                // Conference does not exist, so create one
                conference = new Conference();
                conference.Name = conferenceName;
                conference.Division = division;
                entities.Conferences.AddObject(conference);
                entities.SaveChanges();
                entities.Refresh(System.Data.Objects.RefreshMode.StoreWins,
                                 conference);
            }

            team = entities.Teams.FirstOrDefault(t => t.Name == teamName
                                        && t.ConferenceID == conference.ID);

            if (team == null)
            {
                // Team does not exist, so create one
                team = new Team();
                team.Name = teamName;
                team.Conference = conference;
                entities.Teams.AddObject(team);
            }
        }

        entities.SaveChanges();
    }

    /// <summary>
    /// Imports the games.
    /// </summary>
    /// <remarks>
```

```
/// The comma-delimited file must have the following format:
/// GameDate,AwayTeamName,AwayTeamScore,
/// HomeTeamName,HomeTeamScore,IsNeutralSite
/// The value IsNeutralSite must be either "true" or "false"
/// </remarks>
private void ImportGames()
{
    StreamReader reader = new StreamReader(settings.GamesFileName);
    Game game;
    Team homeTeam;
    Team awayTeam;
    string[] row;
    string homeTeamName;
    string awayTeamName;
    string homeScoreString;
    string awayScoreString;
    string gameDateString;
    string isNeutralSiteString;
    DateTime gameDate;
    int homeScore;
    int awayScore;
    bool isNeutralSite;

    while (reader.EndOfStream == false)
    {
        String line = reader.ReadLine();

        // CSV Format: <Game date>,<Away team name>,<Away team score>,
        // <Home team name>,<Home team score>,<Is Neutral Site>
        row = line.Split(',');

        if (row.Length != 6)
        {
            Console.WriteLine(String.Format("Error in line: {0}", line));
            continue;
        }

        gameDateString = row[0];
        awayTeamName = row[1];
        awayScoreString = row[2];
        homeTeamName = row[3];
        homeScoreString = row[4];
        isNeutralSiteString = row[5];

        homeTeam = entities.Teams.FirstOrDefault(t => t.Name ==
                                                     homeTeamName);
        awayTeam = entities.Teams.FirstOrDefault(t => t.Name ==
                                                     awayTeamName);

        if (homeTeam == null)
        {
            Console.WriteLine(String.Format("Invalid home team: {0}",
                                             homeTeamName));
            continue;
        }
```

```
    if (awayTeam == null)
    {
        Console.WriteLine(String.Format("Invalid away team: {0}",
                                        awayTeamName));
        continue;
    }

    try
    {
        homeScore = Convert.ToInt32(homeScoreString);
        awayScore = Convert.ToInt32(awayScoreString);
    }
    catch (FormatException)
    {
        Console.WriteLine(String.Format("Invalid score in line: {0}",
                                        line));
        continue;
    }

    if (isNeutralSiteString.Equals("true"))
    {
        isNeutralSite = true;
    }
    else if (isNeutralSiteString.Equals("false"))
    {
        isNeutralSite = false;
    }
    else
    {
        Console.WriteLine(String.Format("Invalid isNeutralSite in line:
                                        {0}", line));
        continue;
    }

    try
    {
        gameDate = Convert.ToDateTime(gameDateString);
    }
    catch (Exception)
    {
        Console.WriteLine(String.Format("Invalid date in line: {0}",
                                        line));
        continue;
    }

    game = new Game();
    game.HomeTeam = homeTeam;
    game.HomeScore = homeScore;
    game.AwayTeam = awayTeam;
    game.AwayScore = awayScore;
    game.IsNeutralSite = isNeutralSite;
    game.Date = gameDate;

    entities.Games.AddObject(game);
}
```

```
            entities.SaveChanges();
        }
    }
}

/* TeamGraph.cs
 *
 * Joel Hensley
 * January 27, 2010
 *
 * This class is used to identify and store the different team
 * groups in the database. A team is connected to another team
 * if there is a series of games that connects to the teams together.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DatabaseLayer;

namespace DataImport
{
    class TeamGraph
    {
        private CollegeFootballEntities entities = null;

        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        public TeamGraph(CollegeFootballEntities _entities)
        {
            entities = _entities;
        }

        /// <summary>
        /// Loops through all the teams in the database until all
        /// teams are included in some group.
        /// </summary>
        public void CreateTeamGroups()
        {
            Team team;
            int group = 1;

            team = entities.Teams.FirstOrDefault(t => t.Group == null);
            while (team != null)
            {
                ConnectTeams(team, group);
                team = entities.Teams.FirstOrDefault(t => t.Group == null);
                group++;
            }
        }

        /// <summary>
        /// Recursively loops through a team's opponents until
```

```
        /// all teams in the group are found.
        /// </summary>
        /// <param name="team">The team being examined</param>
        /// <param name="group">The group the team belongs to</param>
        private void ConnectTeams(Team team, int group)
        {
            team.Group = group;
            entities.SaveChanges();

            foreach (Team opponent in team.Opponents.Where(t => t.Group == null))
            {
                ConnectTeams(opponent, group);
            }
        }
    }
}

/* ConferenceGraph.cs
 *
 * Joel Hensley
 * February 9, 2010
 *
 * This class is used to identify and store the different conference
 * groups in the database. A conference is connected to another conference
 * if there is a series of games for any of the teams in that conference
 * that connects to any of the teams in another conference.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace DataImport
{
    class ConferenceGraph
    {
        private CollegeFootballEntities entities = null;
        private List<int> visitedGroups = null;

        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        public ConferenceGraph(CollegeFootballEntities _entities)
        {
            entities = _entities;
            visitedGroups = new List<int>();
        }

        /// <summary>
        /// Loops through all the conferences in the database until all
        /// conferences are included in some group.
        /// </summary>
        public void CreateConferenceGroups()
        {
            Conference conference;
```

```
        int group = 1;

        conference = entities.Conferences.FirstOrDefault(c => c.Group == null);
        while (conference != null)
        {
            visitedGroups.Clear();
            ConnectConferences(conference, group);
            conference = entities.Conferences.FirstOrDefault(t => t.Group ==
                                                                 null);
            group++;
        }
    }

    /// <summary>
    /// Recursively loops through a given conference's distinct set of team
    /// groups to find other conferences with at least one team in that group.
    /// </summary>
    /// <param name="team">The conference being examined</param>
    /// <param name="group">The group the conference belongs to</param>
    private void ConnectConferences(Conference conference, int group)
    {
        List<Conference> groupConferences;
        List<int?> groupList;
        conference.Group = group;
        entities.SaveChanges();

        if (!conference.Teams.IsLoaded)
        {
            conference.Teams.Load();
        }

        groupList = conference.Teams.Where(t => t.Group != null).Select(t =>
                                                    t.Group).Distinct().ToList();

        foreach (int teamGroup in groupList)
        {
            if (visitedGroups.Contains(teamGroup))
            {
                // We've already visited that group
                continue;
            }

            visitedGroups.Add(teamGroup);

            groupConferences = (from t in entities.Teams
                   join c in entities.Conferences on t.ConferenceID equals c.ID
                   where t.Group == teamGroup
                   && c.Group == null
                   select c).ToList();

            foreach (Conference otherConference in groupConferences)
            {
                // It's possible the conference has already been examined
                // so we don't want to call the recursive method if so.
                entities.Refresh(System.Data.Objects.RefreshMode.StoreWins,
                                 otherConference);
```

```
                    if (otherConference.Group == null)
                    {
                        ConnectConferences(otherConference, group);
                    }
                }
            }
        }
    }
}

/* DivisionGraph.cs
 *
 * Joel Hensley
 * February 9, 2010
 *
 * This class is used to identify and store the different division
 * groups in the database. A division is connected to another division
 * if there is a series of games for any of the teams in that division
 * that connects to any of the teams in another division.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace DataImport
{
    class DivisionGraph
    {
        private CollegeFootballEntities entities = null;
        private List<int> visitedGroups = null;

        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        public DivisionGraph(CollegeFootballEntities _entities)
        {
            entities = _entities;
            visitedGroups = new List<int>();
        }

        /// <summary>
        /// Loops through all the divisions in the database until all
        /// divisions are included in some group.
        /// </summary>
        public void CreateDivisionGroups()
        {
            Division division;
            int group = 1;

            division = entities.Divisions.FirstOrDefault(d => d.Group == null);
            while (division != null)
            {
                visitedGroups.Clear();
```

```
            ConnectDivisions(division, group);
            division = entities.Divisions.FirstOrDefault(t => t.Group == null);
            group++;
        }
    }

    /// <summary>
    /// Recursively loops through a given division's distinct set of team groups
    /// to find other divisions with at least one team in that group.
    /// </summary>
    /// <param name="team">The division being examined</param>
    /// <param name="group">The group the division belongs to</param>
    private void ConnectDivisions(Division division, int group)
    {
        List<Division> groupDivisions;
        List<int?> groupList;
        division.Group = group;
        entities.SaveChanges();

        groupList = (from t in entities.Teams
                 join c in entities.Conferences on t.ConferenceID equals c.ID
                 where c.DivisionID == division.ID
                 && t.Group != null
                 select t.Group).Distinct().ToList();

        foreach (int teamGroup in groupList)
        {
            if (visitedGroups.Contains(teamGroup))
            {
                // We've already visited that group
                continue;
            }

            visitedGroups.Add(teamGroup);

            groupDivisions = (from t in entities.Teams
                    join c in entities.Conferences on t.ConferenceID equals c.ID
                    join d in entities.Divisions on c.DivisionID equals d.ID
                    where t.Group == teamGroup
                    && d.Group == null
                    select d).ToList();

            foreach (Division otherDivision in groupDivisions)
            {
                // Its possible the division has already been examined
                // so we don't want to call the recursive method if so.
                entities.Refresh(System.Data.Objects.RefreshMode.StoreWins,
                                 otherDivision);

                if (otherDivision.Group == null)
                {
                    ConnectDivisions(otherDivision, group);
                }
            }
        }
    }
```

```
    }
}
```

Rating System Project

```
/* Program.cs
 *
 * Joel Hensley
 * January 16, 2010
 *
 * This class is used to call the different rating systems, write the results
 * to CSV files, write the results to the database, and then output
 * the progress to the console.
 */

using System;
using System.IO;
using System.Linq;
using System.Collections.Generic;
using DatabaseLayer;

namespace RatingSystem
{
    class Program
    {
        private RatingSettings settings;
        private CollegeFootballEntities entities;
        private Dictionary<int, double> stdRatingVector = null;
        private Dictionary<int, double> hfaRatingVector = null;
        private Dictionary<int, double> mpdRatingVector = null;
        private Dictionary<int, double> hensleyRatingVector = null;
        private double homeFieldAdvantage;
        private const int RoundDecimals = 3;

        /// <summary>
        /// Default constructor
        /// </summary>
        public Program()
        {
            settings = new RatingSettings();
            entities = new CollegeFootballEntities();
            homeFieldAdvantage = 0;
        }

        /// <summary>
        /// Calls run
        /// </summary>
        /// <param name="args">No args are requried</param>
        static void Main(string[] args)
        {
            Program p = new Program();
```

```
        p.Run();

        Console.WriteLine("\nPress any key to continue");
        Console.ReadKey();
    }

    /// <summary>
    /// Loops through each group in the teams, conferences, and divisions
    /// and computes the ratings specified in the settings file.
    /// </summary>
    public void Run()
    {
        int count;
        List<int?> groups = null;

        if (settings.ComputeTeamRatings)
        {
            groups = entities.Teams.Where(t => t.Group != null).Select(t =>
                                          t.Group).Distinct().ToList();

            foreach (int group in groups)
            {
                count = entities.Teams.Where(t => t.Group == group).Count();

                if (count > 1)
                {
                    // There must be at least 2 teams in the group to calculate
                    // ratings
                    ComputeTeamRatings(group);
                }
            }
        }

        if (settings.ComputeConferenceRatings)
        {
            groups = entities.Conferences.Where(c => c.Group != null).Select(
                                              c => c.Group).Distinct().ToList();

            foreach (int group in groups)
            {
                count = entities.Conferences.Where(c => c.Group ==
                                                   group).Count();

                if (count > 1)
                {
                    // There must be at least 2 conferences in the group to
                    // calculate ratings
                    ComputeConferenceRatings(group);
                }
            }
        }

        if (settings.ComputeDivisionRatings)
        {
            groups = entities.Divisions.Where(d => d.Group != null).Select(d =>
```

```
                                                  d.Group).Distinct().ToList();

            foreach (int group in groups)
            {
                count = entities.Divisions.Where(d => d.Group == group).Count();

                if (count > 1)
                {
                    // There must be at least 2 divisions in the group to
                    // calculate ratings
                    ComputeDivisionRatings(group);
                }
            }
        }
    }

    /// <summary>
    /// Computes the rating types for teams specified in the settings file.
    /// </summary>
    /// <param name="group">The group of teams to process</param>
    public void ComputeTeamRatings(int group)
    {
        double[][] matrix;

        if (settings.ComputeStandardRatings
            || settings.ComputeHomefieldAdvantageRatings
            || settings.ComputeMaxPointDifferentialRatings
            || settings.ComputeHensleyRatings)
        {
            Console.WriteLine("Computing Team Ratings");
            Console.WriteLine("----------------------");
        }

        if (settings.ComputeStandardRatings)
        {
            StandardRating stdRating = new StandardRating(entities, group);
            Console.WriteLine("\n{0} for group {1}", StandardRating.RatingName,
                              group);
            Console.Write("Creating team matrix...");
            matrix = stdRating.CreateTeamMatrix();
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            stdRatingVector = stdRating.GetRatings(matrix, RatingObject.Team);
            Console.WriteLine("DONE");
        }

        if (settings.ComputeHomefieldAdvantageRatings)
        {
            HomeFieldAdvantageRating hfaRating = new
                                 HomeFieldAdvantageRating(entities, group);
            Console.WriteLine("\n{0} for group {1}",
                                HomeFieldAdvantageRating.RatingName, group);
            Console.Write("Creating team matrix...");
            matrix = hfaRating.CreateTeamMatrix();
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
```

```
            hfaRatingVector = hfaRating.GetRatings(matrix, RatingObject.Team);
            homeFieldAdvantage = hfaRating.HomeFieldAdvantage;
            Console.WriteLine("DONE");
        }

        if (settings.ComputeMaxPointDifferentialRatings)
        {
            MaxPointDifferentialRating mpdRating = new
                              MaxPointDifferentialRating(entities, group,
                                              settings.MaxPointDifferential);
            Console.WriteLine("\n{0} for group {1}",
                               MaxPointDifferentialRating.RatingName, group);
            Console.Write("Creating team matrix...");
            matrix = mpdRating.CreateTeamMatrix();
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            mpdRatingVector = mpdRating.GetRatings(matrix, RatingObject.Team);
            Console.WriteLine("DONE");
        }

        if (settings.ComputeHensleyRatings)
        {
            HensleyRating hensleyRating = new HensleyRating(entities, group,
                                  settings.LowerBounds, settings.UpperBounds);
            Console.WriteLine("\n{0} for group {1}", HensleyRating.RatingName,
                              group);
            Console.Write("Creating team matrix...");
            matrix = hensleyRating.CreateTeamMatrix();
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            hensleyRatingVector = hensleyRating.GetRatings(matrix,
                                                               RatingObject.Team);
            Console.WriteLine("DONE");
        }

        OutputResults(group, RatingObject.Team, null);
    }

    /// <summary>
    /// Computes the rating types for conferences specified in the settings
    /// file.
    /// </summary>
    /// <param name="group">The group of conferences to process</param>
    public void ComputeConferenceRatings(int group)
    {
        double[][] matrix;
        IEnumerable<Game> interConferenceGames;

        interConferenceGames = entities.GetInterConferenceGames(group);

        if (settings.ComputeStandardRatings
            || settings.ComputeHomefieldAdvantageRatings
            || settings.ComputeMaxPointDifferentialRatings
            || settings.ComputeHensleyRatings)
        {
            Console.WriteLine("\nComputing Conference Ratings");
```

```
    Console.WriteLine("---------------------------");
}

if (settings.ComputeStandardRatings)
{
    StandardRating stdRating = new StandardRating(entities, group);
    Console.WriteLine("\n{0} for group {1}", StandardRating.RatingName,
                      group);
    Console.Write("Creating conference matrix...");
    matrix = stdRating.CreateConferenceMatrix(interConferenceGames);
    Console.WriteLine("DONE");
    Console.Write("Calculating ratings...");
    stdRatingVector = stdRating.GetRatings(matrix,
                                           RatingObject.Conference);
    Console.WriteLine("DONE");
}

if (settings.ComputeHomefieldAdvantageRatings)
{
    HomeFieldAdvantageRating hfaRating = new
                          HomeFieldAdvantageRating(entities, group);
    Console.WriteLine("\n{0} for group {1}",
                        HomeFieldAdvantageRating.RatingName, group);
    Console.Write("Creating conference matrix...");
    matrix = hfaRating.CreateConferenceMatrix(interConferenceGames);
    Console.WriteLine("DONE");
    Console.Write("Calculating ratings...");
    hfaRatingVector = hfaRating.GetRatings(matrix,
                                           RatingObject.Conference);
    homeFieldAdvantage = hfaRating.HomeFieldAdvantage;
    Console.WriteLine("DONE");
}

if (settings.ComputeMaxPointDifferentialRatings)
{
    MaxPointDifferentialRating mpdRating = new
                       MaxPointDifferentialRating(entities, group,
                                    settings.MaxPointDifferential);
    Console.WriteLine("\n{0} for group {1}",
                     MaxPointDifferentialRating.RatingName, group);
    Console.Write("Creating conference matrix...");
    matrix = mpdRating.CreateConferenceMatrix(interConferenceGames);
    Console.WriteLine("DONE");
    Console.Write("Calculating ratings...");
    mpdRatingVector = mpdRating.GetRatings(matrix,
                                           RatingObject.Conference);
    Console.WriteLine("DONE");
}

if (settings.ComputeHensleyRatings)
{
    HensleyRating hensleyRating = new HensleyRating(entities, group,
                        settings.LowerBounds, settings.UpperBounds);
    Console.WriteLine("\n{0} for group {1}", HensleyRating.RatingName,
                      group);
    Console.Write("Creating conference matrix...");
```

```
            matrix = hensleyRating.CreateConferenceMatrix(interConferenceGames);
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            hensleyRatingVector = hensleyRating.GetRatings(matrix,
                                                  RatingObject.Conference);
            Console.WriteLine("DONE");
        }

        OutputResults(group, RatingObject.Conference, interConferenceGames);
    }

    /// <summary>
    /// Computes the rating types for divisions specified in the settings file.
    /// </summary>
    /// <param name="group">The group of divisions to process</param>
    public void ComputeDivisionRatings(int group)
    {
        double[][] matrix;
        IEnumerable<Game> interDivisionGames;

        interDivisionGames = entities.GetInterDivisionGames(group);

        if (settings.ComputeStandardRatings
            || settings.ComputeHomefieldAdvantageRatings
            || settings.ComputeMaxPointDifferentialRatings
            || settings.ComputeHensleyRatings)
        {
            Console.WriteLine("\nComputing Division Ratings");
            Console.WriteLine("--------------------------");
        }

        if (settings.ComputeStandardRatings)
        {
            StandardRating stdRating = new StandardRating(entities, group);
            Console.WriteLine("\n{0} for group {1}", StandardRating.RatingName,
                              group);
            Console.Write("Creating division matrix...");
            matrix = stdRating.CreateDivisionMatrix(interDivisionGames);
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            stdRatingVector = stdRating.GetRatings(matrix,
                                              RatingObject.Division);
            Console.WriteLine("DONE");
        }

        if (settings.ComputeHomefieldAdvantageRatings)
        {
            HomeFieldAdvantageRating hfaRating = new
                                 HomeFieldAdvantageRating(entities, group);
            Console.WriteLine("\n{0} for group {1}",
                               HomeFieldAdvantageRating.RatingName, group);
            Console.Write("Creating division matrix...");
            matrix = hfaRating.CreateDivisionMatrix(interDivisionGames);
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            hfaRatingVector = hfaRating.GetRatings(matrix,
```

```
                                                RatingObject.Division);
            homeFieldAdvantage = hfaRating.HomeFieldAdvantage;
            Console.WriteLine("DONE");
        }

        if (settings.ComputeMaxPointDifferentialRatings)
        {
            MaxPointDifferentialRating mpdRating = new
                                      MaxPointDifferentialRating(entities,
                                          group, settings.MaxPointDifferential);
            Console.WriteLine("\n{0} for group {1}",
                                   MaxPointDifferentialRating.RatingName, group);
            Console.Write("Creating division matrix...");
            matrix = mpdRating.CreateDivisionMatrix(interDivisionGames);
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            mpdRatingVector = mpdRating.GetRatings(matrix,
                                                RatingObject.Division);
            Console.WriteLine("DONE");
        }

        if (settings.ComputeHensleyRatings)
        {
            HensleyRating hensleyRating = new HensleyRating(entities, group,
                                   settings.LowerBounds, settings.UpperBounds);
            Console.WriteLine("\n{0} for group {1}", HensleyRating.RatingName,
                                  group);
            Console.Write("Creating division matrix...");
            matrix = hensleyRating.CreateDivisionMatrix(interDivisionGames);
            Console.WriteLine("DONE");
            Console.Write("Calculating ratings...");
            hensleyRatingVector = hensleyRating.GetRatings(matrix,
                                                      RatingObject.Division);
            Console.WriteLine("DONE");
        }

        OutputResults(group, RatingObject.Division, interDivisionGames);
    }

    /// <summary>
    /// Writes the results to the specified CSV files and the results table in
    /// the database.
    /// </summary>
    /// <param name="group">The group being processed</param>
    /// <param name="ratingObject">
    /// Specifies the type of object the rating applies to
    /// </param>
    /// <param name="games">The games being processed</param>
    public void OutputResults(int group, RatingObject ratingObject,
                              IEnumerable<Game> games)
    {
        StreamWriter sw;
        string fileName = String.Empty;
        string headerLine = "Team";
        string detailLine = String.Empty;
        int i = 1;
```

```
Console.Write("\nWriting to output file...");

string displayName = String.Empty;
IOrderedEnumerable<int> sortedKeys = null;

switch (ratingObject)
{
    case RatingObject.Team:
        fileName = String.Format("{0}_{1}.csv",
                                 settings.TeamResultsOutputFile, group);
        break;
    case RatingObject.Conference:
        fileName = String.Format("{0}_{1}.csv",
                          settings.ConferenceResultsOutputFile, group);
        break;
    case RatingObject.Division:
        fileName = String.Format("{0}_{1}.csv",
                          settings.DivisionResultsOutputFile, group);
        break;
}

try
{
    sw = new StreamWriter(fileName);
    sw.WriteLine("Results for group {0}", group);

    if (settings.ComputeStandardRatings)
    {
        headerLine = String.Format("{0},{1}", headerLine,
                                   StandardRating.RatingName);

        sortedKeys = from k in stdRatingVector.Keys
                     orderby stdRatingVector[k] descending
                     select k;
    }

    if (settings.ComputeHomefieldAdvantageRatings)
    {
        headerLine = String.Format("{0},{1}", headerLine,
                                   HomeFieldAdvantageRating.RatingName);

        sortedKeys = from k in hfaRatingVector.Keys
                     orderby hfaRatingVector[k] descending
                     select k;
    }

    if (settings.ComputeMaxPointDifferentialRatings)
    {
        headerLine = String.Format("{0},{1}", headerLine,
                                 MaxPointDifferentialRating.RatingName);

        sortedKeys = from k in mpdRatingVector.Keys
                     orderby mpdRatingVector[k] descending
                     select k;
    }
```

```
    if (settings.ComputeHensleyRatings)
    {
        headerLine = String.Format("{0},{1}", headerLine,
                                   HensleyRating.RatingName);

        sortedKeys = from k in hensleyRatingVector.Keys
                     orderby hensleyRatingVector[k] descending
                     select k;
    }

    sw.WriteLine(headerLine);

    if (homeFieldAdvantage != 0)
    {
        sw.WriteLine("Home Field Advantage,{0}", homeFieldAdvantage);
    }

    foreach (int key in sortedKeys)
    {
        switch (ratingObject)
        {
            case RatingObject.Team:
                detailLine = WriteTeamResult(key);
                break;

            case RatingObject.Conference:
                detailLine = WriteConferenceResult(key, games);
                break;

            case RatingObject.Division:
                detailLine = WriteDivisionResult(key, games);
                break;
        }

        sw.WriteLine(detailLine);
        i++;
    }
    Console.WriteLine("DONE");
    sw.Close();
}
catch (Exception ex)
{
    Console.WriteLine("\n{0}", ex.Message);
}

try
{
    Console.Write("Writing results to database...");
    entities.SaveChanges();
    Console.Write("DONE\n");
}
catch (Exception ex)
{
    Console.WriteLine("\n{0}", ex.Message);
}
```

```
}

/// <summary>
/// Writes a record to the team result table in the database.
/// </summary>
/// <param name="key">The ID of the team</param>
/// <returns>A comma delimited representation of the data</returns>
private string WriteTeamResult(int key)
{
    Team team = entities.Teams.First(t => t.ID == key);
    TeamResult teamResult;
    String detailLine = String.Format("{0} ({1}-{2})", team.Name, team.Wins,
                                      team.Losses);

    if (!team.TeamResultReference.IsLoaded)
    {
        team.TeamResultReference.Load();
    }
    if (team.TeamResult == null)
    {
        teamResult = new TeamResult();
        teamResult.Team = team;
        entities.TeamResults.AddObject(teamResult);
    }
    else
    {
        teamResult = team.TeamResult;
    }

    teamResult.Wins = team.Wins;
    teamResult.Losses = team.Losses;

    if (settings.ComputeStandardRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                   Math.Round(stdRatingVector[key],
                                   RoundDecimals),
                                   stdRatingVector.Where(k => k.Value >
                                   stdRatingVector[key]).Count() + 1);
        teamResult.StandardRating = stdRatingVector[key];
    }

    if (settings.ComputeHomefieldAdvantageRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                   Math.Round(hfaRatingVector[key],
                                   RoundDecimals),
                                   hfaRatingVector.Where(k => k.Value >
                                   hfaRatingVector[key]).Count() + 1);
        teamResult.HomefieldAdvantageRating = hfaRatingVector[key];
    }

    if (settings.ComputeMaxPointDifferentialRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                  Math.Round(mpdRatingVector[key],
```

```
                                    RoundDecimals),
                                    mpdRatingVector.Where(k => k.Value >
                                    mpdRatingVector[key]).Count() + 1);
        teamResult.MaxPointDifferentialRating = mpdRatingVector[key];
    }

    if (settings.ComputeHensleyRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                     Math.Round(hensleyRatingVector[key],
                                     RoundDecimals),
                                     hensleyRatingVector.Where(k => k.Value >
                                     hensleyRatingVector[key]).Count() + 1);
        teamResult.HensleyRating = hensleyRatingVector[key];
    }

    return detailLine;
}

/// <summary>
/// Writes a record to the conference result table in the database.
/// </summary>
/// <param name="key">The ID of the conference</param>
/// <param name="interConferenceGames">The games being processes</param>
/// <returns>A comma delimited representation of the data</returns>
private string WriteConferenceResult(int key,
                                       IEnumerable<Game> interConferenceGames)
{
    Conference conference = entities.Conferences.First(c => c.ID == key);
    ConferenceResult conferenceResult;
    int wins = entities.GetWins(interConferenceGames, conference);
    int losses = entities.GetLosses(interConferenceGames, conference);
    String detailLine = String.Format("{0} ({1}-{2})", conference.Name,
                                        wins, losses);

    if (!conference.ConferenceResultReference.IsLoaded)
    {
        conference.ConferenceResultReference.Load();
    }
    if (conference.ConferenceResult == null)
    {
        conferenceResult = new ConferenceResult();
        conferenceResult.Conference = conference;
        entities.ConferenceResults.AddObject(conferenceResult);
    }
    else
    {
        conferenceResult = conference.ConferenceResult;
    }

    conferenceResult.Wins = wins;
    conferenceResult.Losses = losses;

    if (settings.ComputeStandardRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
```

```
                                      Math.Round(stdRatingVector[key],
                                      RoundDecimals),
                                      stdRatingVector.Where(k => k.Value >
                                      stdRatingVector[key]).Count() + 1);
        conferenceResult.StandardRating = stdRatingVector[key];
    }

    if (settings.ComputeHomefieldAdvantageRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                      Math.Round(hfaRatingVector[key],
                                      RoundDecimals),
                                      hfaRatingVector.Where(k => k.Value >
                                      hfaRatingVector[key]).Count() + 1);
        conferenceResult.HomefieldAdvantageRating = hfaRatingVector[key];
    }

    if (settings.ComputeMaxPointDifferentialRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                      Math.Round(mpdRatingVector[key],
                                      RoundDecimals),
                                      mpdRatingVector.Where(k => k.Value >
                                      mpdRatingVector[key]).Count() + 1);
        conferenceResult.MaxPointDifferentialRating = mpdRatingVector[key];
    }

    if (settings.ComputeHensleyRatings)
    {
        detailLine = String.Format("{0},{1} ({2})", detailLine,
                                      Math.Round(hensleyRatingVector[key],
                                      RoundDecimals),
                                      hensleyRatingVector.Where(k => k.Value >
                                      hensleyRatingVector[key]).Count() + 1);
        conferenceResult.HensleyRating = hensleyRatingVector[key];
    }

    return detailLine;
}

/// <summary>
/// Writes a record to the division result table in the database.
/// </summary>
/// <param name="key">The ID of the division</param>
/// <param name="interConferenceGames">The games being processes</param>
/// <returns>A comma delimited representation of the data</returns>
private string WriteDivisionResult(int key,
                                      IEnumerable<Game> interDivisionGames)
{
    Division division = entities.Divisions.First(c => c.ID == key);
    DivisionResult divisionResult;
    int wins = entities.GetWins(interDivisionGames, division);
    int losses = entities.GetLosses(interDivisionGames, division);
    String detailLine = String.Format("{0} ({1}-{2})", division.Name, wins,
                                         losses);
```

```
if (!division.DivisionResultReference.IsLoaded)
{
    division.DivisionResultReference.Load();
}
if (division.DivisionResult == null)
{
    divisionResult = new DivisionResult();
    divisionResult.Division = division;
    entities.DivisionResults.AddObject(divisionResult);
}
else
{
    divisionResult = division.DivisionResult;
}

divisionResult.Wins = wins;
divisionResult.Losses = losses;

if (settings.ComputeStandardRatings)
{
    detailLine = String.Format("{0},{1} ({2})", detailLine,
                               Math.Round(stdRatingVector[key],
                               RoundDecimals),
                               stdRatingVector.Where(k => k.Value >
                               stdRatingVector[key]).Count() + 1);
    divisionResult.StandardRating = stdRatingVector[key];
}

if (settings.ComputeHomefieldAdvantageRatings)
{
    detailLine = String.Format("{0},{1} ({2})", detailLine,
                               Math.Round(hfaRatingVector[key],
                               RoundDecimals),
                               hfaRatingVector.Where(k => k.Value >
                               hfaRatingVector[key]).Count() + 1);
    divisionResult.HomefieldAdvantageRating = hfaRatingVector[key];
}

if (settings.ComputeMaxPointDifferentialRatings)
{
    detailLine = String.Format("{0},{1} ({2})", detailLine,
                               Math.Round(mpdRatingVector[key],
                               RoundDecimals),
                               mpdRatingVector.Where(k => k.Value >
                               mpdRatingVector[key]).Count() + 1);
    divisionResult.MaxPointDifferentialRating = mpdRatingVector[key];
}

if (settings.ComputeHensleyRatings)
{
    detailLine = String.Format("{0},{1} ({2})", detailLine,
                               Math.Round(hensleyRatingVector[key],
                               RoundDecimals),
                               hensleyRatingVector.Where(k => k.Value >
                               hensleyRatingVector[key]).Count() + 1);
    divisionResult.HensleyRating = hensleyRatingVector[key];
```

```
            }

            return detailLine;
        }
    }
}

/* GaussJordanElimination.cs
 *
 * Joel Hensley
 * January 16, 2010
 *
 * This class is used to perform the Gauss-Jordan elimination
 * method on a matrix.
 */

namespace RatingSystem
{
    class GaussJordanElimination
    {
        /// <summary>
        /// Swaps row1 with row2 in the passed matrix
        /// </summary>
        /// <param name="matrix">A two dimensional array</param>
        /// <param name="row1">The number of the row in the matrix</param>
        /// <param name="row2">The number of another row in the matrix</param>
        private void SwitchRows(double[][] matrix, int row1, int row2)
        {
            double[] temp;

            temp = matrix[row1];
            matrix[row1] = matrix[row2];
            matrix[row2] = temp;
        }

        /// <summary>
        /// Performs Gauss-Jordan elimination on the system of linear equations
        /// </summary>
        /// <param name="matrix">
        /// The matrix representing the system of linear equations
        /// </param>
        /// <param name="rowCount">The number of rows in the matrix</param>
        /// <param name="colCount">The number of coloumns in the matrix</param>
        /// <returns>The solved variables</returns>
        public double[] PerformElimination(double[][] matrix, int rowCount,
                                           int colCount)
        {
            bool isLastRow;
            double temp;
            double[] solutions = new double[rowCount];

            for (int row = 0; row < rowCount; row++)
            {
                isLastRow = (row == (rowCount - 1));

                if (isLastRow && matrix[row][row] == 0)
```

```
            {
                // We are on the last row in the matrix and the value in the
                // bottom-right corner is 0. Therefore, there is no solution
                return null;
            }
            else if (matrix[row][row] == 0)
            {
                SwitchRows(matrix, row, row + 1);
            }

            if (matrix[row][row] == 0.0)
            {
                return null;
            }
            else
            {
                // Get a 1 in the diaganol value of the current row
                temp = matrix[row][row];
                for (int col = 0; col < colCount; col++)
                {
                    matrix[row][col] /= temp;
                }
            }

            // Subtract multiples of the current row to remove the diaganol
            // value
            for (int rowsAhead = row + 1; rowsAhead < rowCount; rowsAhead++)
            {
                temp = matrix[rowsAhead][row];
                for (int col = 0; col < colCount; col++)
                {
                    matrix[rowsAhead][col] -= temp * matrix[row][col];
                }
            }
        }

        // Perform backwards substitution
        for (int row = rowCount - 1; row >= 0; row--)
        {
            for (int rowsBehind = row - 1; rowsBehind >= 0; rowsBehind--)
            {
                temp = matrix[rowsBehind][row];
                for (int col = 0; col < colCount; col++)
                {
                    matrix[rowsBehind][col] -= temp * matrix[row][col];
                }
            }
        }

        for (int row = 0; row < rowCount; row++)
        {
            solutions[row] = matrix[row][colCount - 1];
        }

        return solutions;
    }
```

```
    }
}

/* StandardRating.cs
 *
 * Joel Hensley
 * January 18, 2010
 *
 * This class forms the base for the other rating methods. It
 * applies no adjusts to the point different and does not
 * account for any home field advantage.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace RatingSystem
{
    /// <summary>
    /// Specifies whether the rating is applied for a team,
    /// conference, or division.
    /// </summary>
    public enum RatingObject
    {
        Team = 0,
        Conference = 1,
        Division = 2
    }

    public class StandardRating
    {
        public static string RatingName = "Standard Rating";
        protected Dictionary<int, int> teamIDMapping;
        protected Dictionary<int, int> conferenceIDMapping;
        protected Dictionary<int, int> divisionIDMapping;
        protected CollegeFootballEntities entities;
        protected int teamCount;
        protected int conferenceCount;
        protected int divisionCount;
        protected RatingSettings ratingSettings;
        protected int group;

        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        /// <param name="_group">The group being processed</param>
        public StandardRating(CollegeFootballEntities _entities, int _group)
        {
            entities = _entities;
            group = _group;
            ratingSettings = new RatingSettings();
            CreateTeamIDMapping();
            CreateConferenceIDMapping();
            CreateDivisionIDMapping();
```

```
}

/// <summary>
/// Solves a matrix using Gauss-Jordan elimination and returns
/// the solution vector mapped to each rating object's primary ID.
/// </summary>
/// <param name="matrix">The matrix to be solved.</param>
/// <param name="ratingObject">The type of object being processed.</param>
/// <returns>
/// The solution vector mapped to each rating object's primary ID.
/// </returns>
/// <remarks>
/// This method can be overridden in subclasses to account
/// for differences in the number of rows, columns, or values stored in the
/// matrix based on the rating system applied.
/// </remarks>
public virtual Dictionary<int, double> GetRatings(double[][] matrix,
                                                  RatingObject ratingObject)
{
    Dictionary<int, double> ratingDictionary = new Dictionary<int,
                                                           double>();
    GaussJordanElimination gj = new GaussJordanElimination();
    double[] ratingVector;
    int rowCount;

    rowCount = matrix.Count();
    ratingVector = gj.PerformElimination(matrix, rowCount, rowCount + 1);

    for (int row = 0; row < rowCount; row++)
    {
        switch (ratingObject)
        {
            case RatingObject.Team:
                ratingDictionary.Add(teamIDMapping.First(t => t.Value ==
                                     row).Key, ratingVector[row]);
                break;

            case RatingObject.Conference:
                ratingDictionary.Add(conferenceIDMapping.First(c => c.Value
                                            == row).Key, ratingVector[row]);
                break;

            case RatingObject.Division:
                ratingDictionary.Add(divisionIDMapping.First(d => d.Value ==
                                            row).Key, ratingVector[row]);
                break;
        }
    }

    return ratingDictionary;
}

/// <summary>
/// Creates a dictionary to map a team's ID to its row number in the matrix.
/// </summary>
protected void CreateTeamIDMapping()
```

```
{
    teamIDMapping = new Dictionary<int, int>();
    teamCount = 0;

    foreach (Team team in entities.Teams.Where(t => t.Group ==
                                              group).OrderBy(t => t.ID))
    {
        teamIDMapping.Add(team.ID, teamCount);
        teamCount++;
    }
}

/// <summary>
/// Creates a dictionary to map a conference's ID to its row number in the
/// matrix.
/// </summary>
protected void CreateConferenceIDMapping()
{
    conferenceIDMapping = new Dictionary<int, int>();
    conferenceCount = 0;

    foreach (Conference conference in entities.Conferences.Where(c =>
             c.Group == group).OrderBy(d => d.ID))
    {
        conferenceIDMapping.Add(conference.ID, conferenceCount);
        conferenceCount++;
    }
}

/// <summary>
/// Creates a dictionary to map a division's ID to its row number in the
/// matrix.
/// </summary>
protected void CreateDivisionIDMapping()
{
    divisionIDMapping = new Dictionary<int, int>();
    divisionCount = 0;

    foreach (Division division in entities.Divisions.Where(d => d.Group ==
             group).OrderBy(d => d.ID))
    {
        divisionIDMapping.Add(division.ID, divisionCount);
        divisionCount++;
    }
}

/// <summary>
/// Populates the rating matrix to be solved for teams.
/// </summary>
/// <returns>The populated rating matrix.</returns>
/// <remarks>
/// This method can be overridden in subclasses to account for differences
/// in the number of rows, columns, or values stored in the matrix based on
/// the rating system applied.
/// </remarks>
public virtual double[][] CreateTeamMatrix()
```

```
        {
            int rowCount = 0;
            int gameCount = 0;
            double[][] matrix = new double[teamCount][];

            foreach (Team team in entities.Teams.Where(t => t.Group ==
                         group).OrderBy(t => t.ID))
            {
                gameCount = team.Games.Count();
                matrix[rowCount] = new double[teamCount + 1];

                // Diaganol value is the number of games played
                matrix[teamIDMapping[team.ID]][teamIDMapping[team.ID]] = gameCount;

                // Right hand side of every line is total score margin
                matrix[rowCount][teamCount] = team.PointDifferential;
                rowCount++;
            }

            foreach (Game game in entities.GetGames(group))
            {
    matrix[teamIDMapping[game.HomeTeamID]][teamIDMapping[game.AwayTeamID]] -= 1;
    matrix[teamIDMapping[game.AwayTeamID]][teamIDMapping[game.HomeTeamID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < teamCount; col++)
            {
                matrix[teamCount - 1][col] = 1;
            }
            matrix[teamCount - 1][teamCount] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for conferences.
        /// </summary>
        /// <param name="interConferenceGames">
        /// All interconference games played.
        /// </param>
        /// <returns>The populated rating matrix.</returns>
        /// <remarks>
        /// This method can be overridden in subclasses to account for differences
        /// in the number of rows, columns, or values stored in the matrix based on
        /// the rating system applied.
        /// </remarks>
        public virtual double[][] CreateConferenceMatrix(
                         IEnumerable<Game> interConferenceGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            double[][] matrix = new double[conferenceCount][];
            IEnumerable<Game> conferenceGames;

            foreach (Conference conference in entities.Conferences.Where(c =>
```

```
                  c.Group == group).OrderBy(c => c.ID))
            {
                conferenceGames = entities.GetGames(conference);
                gameCount = conferenceGames.Count();

                matrix[rowCount] = new double[conferenceCount + 1];

                // Diaganol value is the number of games played
matrix[conferenceIDMapping[conference.ID]][conferenceIDMapping[conference.ID]] =
gameCount;

                // Right hand side of every line is total score margin
                matrix[rowCount][conferenceCount] =
                        entities.GetPointDifferential(conferenceGames, conference);
                rowCount++;
            }

            foreach (Game game in interConferenceGames)
            {
matrix[conferenceIDMapping[game.HomeConferenceID]][conferenceIDMapping[game.AwayConf
erenceID]] -= 1;
matrix[conferenceIDMapping[game.AwayConferenceID]][conferenceIDMapping[game.HomeConf
erenceID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < conferenceCount; col++)
            {
                matrix[conferenceCount - 1][col] = 1;
            }
            matrix[conferenceCount - 1][conferenceCount] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for divisions.
        /// </summary>
        /// <param name="interDivisionGames">All interdivision games played.</param>
        /// <returns>The populated rating matrix.</returns>
        /// <remarks>
        /// This method can be overridden in subclasses to account for differences
        /// in the number of rows, columns, or values stored in the matrix based on
        /// the rating system applied.
        /// </remarks>
        public virtual double[][] CreateDivisionMatrix(
                                      IEnumerable<Game> interDivisionGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            double[][] matrix = new double[divisionCount][];
            IEnumerable<Game> divisionGames;

            foreach (Division division in entities.Divisions.Where(d => d.Group ==
                  group).OrderBy(d => d.ID))
            {
```

```
                    divisionGames = entities.GetGames(division);
                    gameCount = divisionGames.Count();

                    matrix[rowCount] = new double[divisionCount + 1];

                    // Diaganol value is the number of games played
matrix[divisionIDMapping[division.ID]][divisionIDMapping[division.ID]] = gameCount;

                    // Right hand side of every line is total score margin
                    matrix[rowCount][divisionCount] =
                                entities.GetPointDifferential(divisionGames, division);
                    rowCount++;
                }

                foreach (Game game in interDivisionGames)
                {
matrix[divisionIDMapping[game.HomeDivisionID]][divisionIDMapping[game.AwayDivisionID
]] -= 1;
matrix[divisionIDMapping[game.AwayDivisionID]][divisionIDMapping[game.HomeDivisionID
]] -= 1;
                }

                // Last row should be all ones and then a zero for RHS
                for (int col = 0; col < divisionCount; col++)
                {
                    matrix[divisionCount - 1][col] = 1;
                }
                matrix[divisionCount - 1][divisionCount] = 0;

                return matrix;
            }
        }
}

/* HomeFieldAdvantageRating.cs
 *
 * Joel Hensley
 * January 18, 2010
 *
 * This class derives from the standard rating class and
 * adjusts the ratings to account for a computed home field
 * advantage factor.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace RatingSystem
{
    public class HomeFieldAdvantageRating : StandardRating
    {
        public static new string RatingName = "Home Field Advantage Ratings";
        protected double homeFieldAdvantage = 0;

        /// <summary>
```

```
/// The computed home field advantage factor.
/// </summary>
public double HomeFieldAdvantage
{
    get
    {
        return homeFieldAdvantage;
    }
}

/// <summary>
/// Default constructor
/// </summary>
/// <param name="_entities">The database object</param>
/// <param name="_group">The group being processes</param>
public HomeFieldAdvantageRating(CollegeFootballEntities _entities,
                                int _group) : base(_entities, _group)
{
}

/// <summary>
/// Solves a matrix using Gauss-Jordan elimination and returns
/// the solution vector mapped to each rating object's primary ID.
/// </summary>
/// <param name="matrix">The matrix to be solved.</param>
/// <param name="ratingObject">The type of object being processed.</param>
/// <returns>
/// The solution vector mapped to each rating object's primary ID.
/// </returns>
public override Dictionary<int, double> GetRatings(double[][] matrix,
                                                   RatingObject ratingObject)
{
    Dictionary<int, double> ratingDictionary = new Dictionary<int,
                                                               double>();
    GaussJordanElimination gj = new GaussJordanElimination();
    double[] ratingVector;
    int rowCount;

    rowCount = matrix.Count();
    ratingVector = gj.PerformElimination(matrix, rowCount, rowCount + 1);

    // The last row is the homefield advantage
    for (int row = 0; row < rowCount - 1; row++)
    {
        switch (ratingObject)
        {
            case RatingObject.Team:
                ratingDictionary.Add(teamIDMapping.First(t => t.Value ==
                                     row).Key, ratingVector[row]);
                break;

            case RatingObject.Conference:
                ratingDictionary.Add(conferenceIDMapping.First(c => c.Value
                                     == row).Key, ratingVector[row]);
                break;
```

```
                case RatingObject.Division:
                    ratingDictionary.Add(divisionIDMapping.First(d => d.Value ==
                                         row).Key, ratingVector[row]);
                    break;
            }
        }

        homeFieldAdvantage = ratingVector[rowCount - 1];

        return ratingDictionary;
    }

    /// <summary>
    /// Populates the rating matrix to be solved for teams.
    /// </summary>
    /// <returns>The populated rating matrix.</returns>
    public override double[][] CreateTeamMatrix()
    {
        int rowCount = 0;
        int gameCount = 0;
        double[][] matrix = new double[teamCount + 1][];
        IEnumerable<Game> groupGames = entities.GetGames(group);

        matrix[teamCount] = new double[teamCount + 2];
        foreach (Team team in entities.Teams.Where(t => t.Group ==
                                                  group).OrderBy(t => t.ID))
        {
            gameCount = team.Games.Count();
            matrix[rowCount] = new double[teamCount + 2];

            // Diaganol value is the number of games played
            matrix[rowCount][teamIDMapping[team.ID]] = gameCount;

            // Second to last column is home game differential
            matrix[rowCount][teamCount] = team.HomeGameDifferential;
            matrix[teamCount][rowCount] = team.HomeGameDifferential;

            // Right hand side of every line is total score margin
            matrix[rowCount][teamCount + 1] = team.PointDifferential;
            rowCount++;
        }
        matrix[teamCount][teamCount] = entities.GetHomeGameCount(groupGames);
        matrix[teamCount][teamCount + 1] =
                           entities.GetHomeGamePointDifferential(groupGames);

        foreach (Game game in groupGames)
        {
matrix[teamIDMapping[game.HomeTeamID]][teamIDMapping[game.AwayTeamID]] -= 1;
matrix[teamIDMapping[game.AwayTeamID]][teamIDMapping[game.HomeTeamID]] -= 1;
        }

        // Last row should be all ones and then a zero for homefield advantage
        // and a 0 for the RHS
        for (int col = 0; col < teamCount; col++)
        {
            matrix[teamCount - 1][col] = 1;
```

```
            }
            matrix[teamCount - 1][teamCount] = 0;
            matrix[teamCount - 1][teamCount + 1] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for conferences.
        /// </summary>
        /// <param name="interConferenceGames">
        /// All interconference games played.
        /// </param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateConferenceMatrix(
                                        IEnumerable<Game> interConferenceGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            int homeGameDifferential;
            double[][] matrix = new double[conferenceCount + 1][];
            IEnumerable<Game> conferenceGames;

            matrix[conferenceCount] = new double[conferenceCount + 2];
            foreach (Conference conference in entities.Conferences.Where(c =>
                     c.Group == group).OrderBy(c => c.ID))
            {
                conferenceGames = entities.GetGames(conference);
                gameCount = conferenceGames.Count();

                matrix[rowCount] = new double[conferenceCount + 2];

                // Diaganol value is the number of games played
matrix[conferenceIDMapping[conference.ID]][conferenceIDMapping[conference.ID]] =
gameCount;

                // Second to last column is home game differential
                homeGameDifferential =
                      entities.GetHomeGameDifferential(conferenceGames, conference);
                matrix[rowCount][conferenceCount] = homeGameDifferential;
                matrix[conferenceCount][rowCount] = homeGameDifferential;

                // Right hand side of every line is total score margin
                matrix[rowCount][conferenceCount + 1] =
                         entities.GetPointDifferential(conferenceGames, conference);
                rowCount++;
            }
            matrix[conferenceCount][conferenceCount] =
                                        entities.GetHomeGameCount(interConferenceGames);
            matrix[conferenceCount][conferenceCount + 1] =
                        entities.GetHomeGamePointDifferential(interConferenceGames);

            foreach (Game game in interConferenceGames)
            {
matrix[conferenceIDMapping[game.HomeConferenceID]][conferenceIDMapping[game.AwayConf
erenceID]] -= 1;
```

```
matrix[conferenceIDMapping[game.AwayConferenceID]][conferenceIDMapping[game.HomeConf
erenceID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < conferenceCount; col++)
            {
                matrix[conferenceCount - 1][col] = 1;
            }
            matrix[conferenceCount - 1][conferenceCount] = 0;
            matrix[conferenceCount - 1][conferenceCount + 1] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for divisions.
        /// </summary>
        /// <param name="interDivisionGames">All interdivision games played.</param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateDivisionMatrix(
                                          IEnumerable<Game> interDivisionGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            int homeGameDifferential;
            double[][] matrix = new double[divisionCount + 1][];
            IEnumerable<Game> divisionGames;

            matrix[divisionCount] = new double[divisionCount + 2];
            foreach (Division division in entities.Divisions.Where(d => d.Group ==
                     group).OrderBy(d => d.ID))
            {
                divisionGames = entities.GetGames(division);
                gameCount = divisionGames.Count();

                matrix[rowCount] = new double[divisionCount + 2];

                // Diaganol value is the number of games played
matrix[divisionIDMapping[division.ID]][divisionIDMapping[division.ID]] = gameCount;

                // Second to last column is home game differential
                homeGameDifferential =
                          entities.GetHomeGameDifferential(divisionGames, division);
                matrix[rowCount][divisionCount] = homeGameDifferential;
                matrix[divisionCount][rowCount] = homeGameDifferential;

                // Right hand side of every line is total score margin
                matrix[rowCount][divisionCount + 1] =
                             entities.GetPointDifferential(divisionGames, division);
                rowCount++;
            }
            matrix[divisionCount][divisionCount] =
                                        entities.GetHomeGameCount(interDivisionGames);
            matrix[divisionCount][divisionCount + 1] =
                          entities.GetHomeGamePointDifferential(interDivisionGames);
```

```
            foreach (Game game in interDivisionGames)
            {
matrix[divisionIDMapping[game.HomeDivisionID]][divisionIDMapping[game.AwayDivisionID
]] -= 1;
matrix[divisionIDMapping[game.AwayDivisionID]][divisionIDMapping[game.HomeDivisionID
]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < divisionCount; col++)
            {
                matrix[divisionCount - 1][col] = 1;
            }
            matrix[divisionCount - 1][divisionCount] = 0;
            matrix[divisionCount - 1][divisionCount + 1] = 0;

            return matrix;
        }
    }
}

/* MaxPointDifferentialRating.cs
 *
 * Joel Hensley
 * January 23, 2010
 *
 * This class derives from the standard rating class and
 * adjusts the ratings to account for a user-defined max
 * point differential in each game. If no max point differential
 * is passed, the default value is 28.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace RatingSystem
{
    public class MaxPointDifferentialRating : StandardRating
    {
        public static new string RatingName = "Max Point Differential Ratings";

        private int maxPointDifferential = 28;

        /// <summary>
        /// The maximum point differential allowed in a given game.
        /// </summary>
        public int MaxPointDifferential
        {
            get
            {
                return maxPointDifferential;
            }
        }
```

```
        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        /// <param name="_group">The group being processes</param>
        public MaxPointDifferentialRating(CollegeFootballEntities _entities,
                                          int _group) : base(_entities, _group)
        {
        }

        /// <summary>
        /// User-defined constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        /// <param name="_group">The group being processes</param>
        /// <param name="_maxPointDifferential">
        /// The maximum point differential
        /// </param>
        public MaxPointDifferentialRating(CollegeFootballEntities _entities,
                int _group, int _maxPointDifferential) : base(_entities, _group)
        {
            maxPointDifferential = _maxPointDifferential;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for teams.
        /// </summary>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateTeamMatrix()
        {
            int rowCount = 0;
            int gameCount = 0;
            double[][] matrix = new double[teamCount][];

            foreach (Team team in entities.Teams.Where(t => t.Group ==
                     group).OrderBy(t => t.ID))
            {
                gameCount = team.Games.Count();
                matrix[rowCount] = new double[teamCount + 1];

                // Diaganol value is the number of games played
                matrix[rowCount][teamIDMapping[team.ID]] = gameCount;

                // Right hand side of every line is the calculated score margin
                matrix[rowCount][teamCount] =
                                 team.GetPointDifferential(maxPointDifferential);
                rowCount++;
            }

            foreach (Game game in entities.GetGames(group))
            {
matrix[teamIDMapping[game.HomeTeamID]][teamIDMapping[game.AwayTeamID]] -= 1;
matrix[teamIDMapping[game.AwayTeamID]][teamIDMapping[game.HomeTeamID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
```

```
            for (int col = 0; col < teamCount; col++)
            {
                matrix[teamCount - 1][col] = 1;
            }
            matrix[teamCount - 1][teamCount] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for conferences.
        /// </summary>
        /// <param name="interConferenceGames">
        /// All interconference games played.
        /// </param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateConferenceMatrix(
                                            IEnumerable<Game> interConferenceGames)
        {
            int rowCount = 0;
            double[][] matrix = new double[conferenceCount][];
            IEnumerable<Game> conferenceGames;

            foreach (Conference conference in entities.Conferences.Where(c =>
                     c.Group == group).OrderBy(c => c.ID))
            {
                conferenceGames = entities.GetGames(conference);

                matrix[rowCount] = new double[conferenceCount + 1];

                // Diaganol value is the number of games played
matrix[conferenceIDMapping[conference.ID]][conferenceIDMapping[conference.ID]] =
conferenceGames.Count();

                // Right hand side of every line is total score margin
                matrix[rowCount][conferenceCount] =
entities.GetPointDifferential(conferenceGames, conference, maxPointDifferential);
                rowCount++;
            }

            foreach (Game game in interConferenceGames)
            {
matrix[conferenceIDMapping[game.HomeConferenceID]][conferenceIDMapping[game.AwayConf
erenceID]] -= 1;
matrix[conferenceIDMapping[game.AwayConferenceID]][conferenceIDMapping[game.HomeConf
erenceID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < conferenceCount; col++)
            {
                matrix[conferenceCount - 1][col] = 1;
            }
            matrix[conferenceCount - 1][conferenceCount] = 0;

            return matrix;
```

```
        }

        /// <summary>
        /// Populates the rating matrix to be solved for divisions.
        /// </summary>
        /// <param name="interDivisionGames">All interdivision games played.</param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateDivisionMatrix(
                                          IEnumerable<Game> interDivisionGames)
        {
            int rowCount = 0;
            double[][] matrix = new double[divisionCount][];
            IEnumerable<Game> divisionGames;

            foreach (Division division in entities.Divisions.Where(d => d.Group ==
                     group).OrderBy(d => d.ID))
            {
                divisionGames = entities.GetGames(division);

                matrix[rowCount] = new double[divisionCount + 1];

                // Diaganol value is the number of games played
matrix[divisionIDMapping[division.ID]][divisionIDMapping[division.ID]] =
divisionGames.Count();

                // Right hand side of every line is total score margin
                matrix[rowCount][divisionCount] =
                                 entities.GetPointDifferential(divisionGames,
                                                 division, maxPointDifferential);
                rowCount++;
            }

            foreach (Game game in interDivisionGames)
            {
matrix[divisionIDMapping[game.HomeDivisionID]][divisionIDMapping[game.AwayDivisionID
]] -= 1;
matrix[divisionIDMapping[game.AwayDivisionID]][divisionIDMapping[game.HomeDivisionID
]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < divisionCount; col++)
            {
                matrix[divisionCount - 1][col] = 1;
            }
            matrix[divisionCount - 1][divisionCount] = 0;

            return matrix;
        }
    }
}

/* HensleyRating.cs
 *
 * Joel Hensley
 *  March 4, 2010
```

```
 *
 * This class derives from the home field advantage rating class.
 * Instead of using the points differential, this class comptues
 * the win or loss value using a function based on the winning
 * score and losing score. A user can define the upper and lower
 * bounds of this value if desired; otherwise, it defaults to
 * 4.0 and 1.0, respectively.
 */

using System.Collections.Generic;
using System.Linq;
using DatabaseLayer;

namespace RatingSystem
{
    public class HensleyRating : HomeFieldAdvantageRating
    {
        public static new string RatingName = "Hensley Rating";

        private double upperBounds = 4.0F;
        /// <summary>
        /// The upper bounds to the win-loss value function.
        /// </summary>
        public double UpperBounds
        {
            get
            {
                return upperBounds;
            }
        }

        private double lowerBounds = 1.0F;
        /// <summary>
        /// The lower bounds to the win-loss value function.
        /// </summary>
        public double LowerBounds
        {
            get
            {
                return lowerBounds;
            }
        }

        /// <summary>
        /// Default constructor
        /// </summary>
        /// <param name="_entities">The database object</param>
        /// <param name="_group">The group being processed</param>
        public HensleyRating(CollegeFootballEntities _entities, int _group)
            : base(_entities, _group)
        {
        }

        /// <summary>
        /// User-defined constructor
        /// </summary>
```

```
        /// <param name="_entities">The database object</param>
        /// <param name="_group">The group being processed</param>
        /// <param name="_lowerBounds">
        /// The lower bounds to the win-loss value function
        /// </param>
        /// <param name="_upperBounds">
        /// The upper bounds to the win-loss value function
        /// </param>
        public HensleyRating(CollegeFootballEntities _entities, int _group,
                 double _lowerBounds, double _upperBounds) : base(_entities, _group)
        {
            upperBounds = _upperBounds;
            lowerBounds = _lowerBounds;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for teams.
        /// </summary>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateTeamMatrix()
        {
            int rowCount = 0;
            int gameCount = 0;
            double[][] matrix = new double[teamCount + 1][];
            IEnumerable<Game> groupGames = entities.GetGames(group);

            matrix[teamCount] = new double[teamCount + 2];
            foreach (Team team in entities.Teams.Where(t => t.Group ==
                     group).OrderBy(t => t.ID))
            {
                gameCount = team.Games.Count();
                matrix[rowCount] = new double[teamCount + 2];

                // Second to last column is home game differential
                matrix[rowCount][teamCount] = team.HomeGameDifferential;
                matrix[teamCount][rowCount] = team.HomeGameDifferential;

                // Diaganol value is the number of games played
                matrix[rowCount][teamIDMapping[team.ID]] = gameCount;

                // Right hand side of every line is the calculated score margin
                matrix[rowCount][teamCount + 1] =
                        team.GetHensleyPointDifferential(lowerBounds, upperBounds);
                rowCount++;
            }
            matrix[teamCount][teamCount] = entities.GetHomeGameCount(groupGames);
            matrix[teamCount][teamCount + 1] =
                           entities.GetHomeGameHensleyPointDifferential(groupGames,
                                                              lowerBounds, upperBounds);

            foreach (Game game in groupGames)
            {
matrix[teamIDMapping[game.HomeTeamID]][teamIDMapping[game.AwayTeamID]] -= 1;
matrix[teamIDMapping[game.AwayTeamID]][teamIDMapping[game.HomeTeamID]] -= 1;
            }
```

```
            // Last row should be all ones and then a zero for homefield advantage
            // and a 0 for the RHS
            for (int col = 0; col < teamCount; col++)
            {
                matrix[teamCount - 1][col] = 1;
            }
            matrix[teamCount - 1][teamCount] = 0;
            matrix[teamCount - 1][teamCount + 1] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for conferences.
        /// </summary>
        /// <param name="interConferenceGames">
        /// All interconference games played.
        /// </param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateConferenceMatrix(
                                         IEnumerable<Game> interConferenceGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            int homeGameDifferential;
            double[][] matrix = new double[conferenceCount + 1][];
            IEnumerable<Game> conferenceGames;

            matrix[conferenceCount] = new double[conferenceCount + 2];
            foreach (Conference conference in entities.Conferences.Where(c =>
                     c.Group == group).OrderBy(c => c.ID))
            {
                conferenceGames = entities.GetGames(conference);
                gameCount = conferenceGames.Count();

                matrix[rowCount] = new double[conferenceCount + 2];

                // Diaganol value is the number of games played
matrix[conferenceIDMapping[conference.ID]][conferenceIDMapping[conference.ID]] =
gameCount;

                // Second to last column is home game differential
                homeGameDifferential =
                                   entities.GetHomeGameDifferential(conferenceGames,
                                                                    conference);
                matrix[rowCount][conferenceCount] = homeGameDifferential;
                matrix[conferenceCount][rowCount] = homeGameDifferential;

                // Right hand side of every line is total score margin
                matrix[rowCount][conferenceCount + 1] =
                               entities.GetHensleyPointDifferential(conferenceGames,
                                           conference, lowerBounds, upperBounds);
                rowCount++;
            }
            matrix[conferenceCount][conferenceCount] =
                               entities.GetHomeGameCount(interConferenceGames);
```

```
            matrix[conferenceCount][conferenceCount + 1] =
                  entities.GetHomeGameHensleyPointDifferential(interConferenceGames,
                                                        lowerBounds, upperBounds);

            foreach (Game game in interConferenceGames)
            {
matrix[conferenceIDMapping[game.HomeConferenceID]][conferenceIDMapping[game.AwayConf
erenceID]] -= 1;
matrix[conferenceIDMapping[game.AwayConferenceID]][conferenceIDMapping[game.HomeConf
erenceID]] -= 1;
            }

            // Last row should be all ones and then a zero for RHS
            for (int col = 0; col < conferenceCount; col++)
            {
                matrix[conferenceCount - 1][col] = 1;
            }
            matrix[conferenceCount - 1][conferenceCount] = 0;
            matrix[conferenceCount - 1][conferenceCount + 1] = 0;

            return matrix;
        }

        /// <summary>
        /// Populates the rating matrix to be solved for divisions.
        /// </summary>
        /// <param name="interDivisionGames">All interdivision games played.</param>
        /// <returns>The populated rating matrix.</returns>
        public override double[][] CreateDivisionMatrix(
                                        IEnumerable<Game> interDivisionGames)
        {
            int rowCount = 0;
            int gameCount = 0;
            int homeGameDifferential;
            double[][] matrix = new double[divisionCount + 1][];
            IEnumerable<Game> divisionGames;

            matrix[divisionCount] = new double[divisionCount + 2];
            foreach (Division division in entities.Divisions.Where(d => d.Group ==
                     group).OrderBy(d => d.ID))
            {
                divisionGames = entities.GetGames(division);
                gameCount = divisionGames.Count();

                matrix[rowCount] = new double[divisionCount + 2];

                // Diaganol value is the number of games played
matrix[divisionIDMapping[division.ID]][divisionIDMapping[division.ID]] = gameCount;

                // Second to last column is home game differential
                homeGameDifferential =
                          entities.GetHomeGameDifferential(divisionGames, division);
                matrix[rowCount][divisionCount] = homeGameDifferential;
                matrix[divisionCount][rowCount] = homeGameDifferential;

                // Right hand side of every line is total score margin
```

```
                    matrix[rowCount][divisionCount + 1] =
                                    entities.GetHensleyPointDifferential(divisionGames,
                                                    division, lowerBounds, upperBounds);
                    rowCount++;
                }
                matrix[divisionCount][divisionCount] =
                                          entities.GetHomeGameCount(interDivisionGames);
                matrix[divisionCount][divisionCount + 1] =
                       entities.GetHomeGameHensleyPointDifferential(interDivisionGames,
                                                              lowerBounds, upperBounds);

                foreach (Game game in interDivisionGames)
                {
matrix[divisionIDMapping[game.HomeDivisionID]][divisionIDMapping[game.AwayDivisionID
]] -= 1;
matrix[divisionIDMapping[game.AwayDivisionID]][divisionIDMapping[game.HomeDivisionID
]] -= 1;
                }

                // Last row should be all ones and then a zero for RHS
                for (int col = 0; col < divisionCount; col++)
                {
                    matrix[divisionCount - 1][col] = 1;
                }
                matrix[divisionCount - 1][divisionCount] = 0;
                matrix[divisionCount - 1][divisionCount + 1] = 0;

                return matrix;
            }
        }
    }
```

Rating Evaluator Project

```
/* Program.cs
 *
 * Joel Hensley
 * March 29, 2010
 *
 * This class is used to evaluate the different rating systems, write the results
 * to CSV files as well as the console.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using DatabaseLayer;
using RatingSystem;

namespace RatingEvaluator
{
    class Program
    {
```

```
private EvaluatorSetting settings;
private CollegeFootballEntities entities;
private Dictionary<int, int> stdRatingRankDictionary;
private Dictionary<int, int> hfaRatingRankDictionary;
private Dictionary<int, int> mpdRatingRankDictionary;
private Dictionary<int, int> hensleyRatingRankDictionary;
private int stdRatingMistakeCount;
private int hfaRatingMistakeCount;
private int mpdRatingMistakeCount;
private int hensleyRatingMistakeCount;
private double stdRatingError;
private double hfaRatingError;
private double mpdRatingError;
private double hensleyRatingError;
private int ratingCount;
private double stdCorrelationCoefficientAPTop25;
private double hfaCorrelationCoefficientAPTop25;
private double mpdCorrelationCoefficientAPTop25;
private double hensleyCorrelationCoefficientAPTop25;
private double stdCorrelationCoefficientAvgFBSRanking;
private double hfaCorrelationCoefficientAvgFBSRanking;
private double mpdCorrelationCoefficientAvgFBSRanking;
private double hensleyCorrelationCoefficientAvgFBSRanking;

/// <summary>
/// Default constructor
/// </summary>
public Program()
{
    settings = new EvaluatorSetting();
    entities = new CollegeFootballEntities();
    ratingCount = 0;
    stdCorrelationCoefficientAPTop25 = 0;
    hfaCorrelationCoefficientAPTop25 = 0;
    mpdCorrelationCoefficientAPTop25 = 0;
    hensleyCorrelationCoefficientAPTop25 = 0;

    stdRatingMistakeCount = 0;
    hfaRatingMistakeCount = 0;
    mpdRatingMistakeCount = 0;
    hensleyRatingMistakeCount = 0;

    stdRatingError = 0;
    hfaRatingError = 0;
    mpdRatingError = 0;
    hensleyRatingError = 0;

    stdRatingRankDictionary = new Dictionary<int, int>();
    hfaRatingRankDictionary = new Dictionary<int, int>();
    mpdRatingRankDictionary = new Dictionary<int, int>();
    hensleyRatingRankDictionary = new Dictionary<int, int>();
}

/// <summary>
/// Calls run
/// </summary>
```

```
/// <param name="args"></param>
static void Main(string[] args)
{
    Program p = new Program();

    try
    {
        p.Run();
    }
    catch (Exception)
    {
        Console.WriteLine("An error occurred during execution.");
    }

    Console.WriteLine("\n\nPress any key to continue");
    Console.ReadKey();
}

/// <summary>
/// Evaluates the four different rating systems and then writes the results
/// to CSV files as well as the console.
/// </summary>
public void Run()
{
    bool isHomeTeamWinner;
    bool isMistake;
    double error;
    int homeRank;
    int awayRank;

    Console.Write("Populating rank dictionaries...");
    PopulateRatingRankDictionaries();
    Console.WriteLine("DONE");

    Console.Write("Evaluating ranks...");
    foreach (Game game in entities.Games)
    {
        isHomeTeamWinner = (game.HomeScore > game.AwayScore);

        // Standard Rating
        homeRank = stdRatingRankDictionary[game.HomeTeamID];
        awayRank = stdRatingRankDictionary[game.AwayTeamID];
        error = GetError(homeRank, awayRank);
        isMistake = IsMistake(isHomeTeamWinner, homeRank, awayRank);

        if (isMistake)
        {
            stdRatingMistakeCount++;
            stdRatingError += error;
        }

        // Home Field Advantage Rating
        homeRank = hfaRatingRankDictionary[game.HomeTeamID];
        awayRank = hfaRatingRankDictionary[game.AwayTeamID];
        error = GetError(homeRank, awayRank);
        isMistake = IsMistake(isHomeTeamWinner, homeRank, awayRank);
```

```
            if (isMistake)
            {
                hfaRatingMistakeCount++;
                hfaRatingError += error;
            }

            // Max Point Differential Rating
            homeRank = mpdRatingRankDictionary[game.HomeTeamID];
            awayRank = mpdRatingRankDictionary[game.AwayTeamID];
            error = GetError(homeRank, awayRank);
            isMistake = IsMistake(isHomeTeamWinner, homeRank, awayRank);

            if (isMistake)
            {
                mpdRatingMistakeCount++;
                mpdRatingError += error;
            }

            // Hensley Rating
            homeRank = hensleyRatingRankDictionary[game.HomeTeamID];
            awayRank = hensleyRatingRankDictionary[game.AwayTeamID];
            error = GetError(homeRank, awayRank);
            isMistake = IsMistake(isHomeTeamWinner, homeRank, awayRank);

            if (isMistake)
            {
                hensleyRatingMistakeCount++;
                hensleyRatingError += error;
            }
        }

        ComputeAPTop25CorrelationCoefficient();
        ComputeAvgFBSRankingCorrelationCoefficient();
        Console.WriteLine("DONE");

        OutputResults();
    }

    /// <summary>
    /// Determines whether a lower ranked team beat a higher ranked team.
    /// </summary>
    /// <param name="isHomeTeamWinner">
    /// Whether or not the home team won the game
    /// </param>
    /// <param name="homeRank">The rank of the home team</param>
    /// <param name="awayRank">The rank of the away team</param>
    /// <returns>True if the lower ranked team won; otherwise, false.</returns>
    /// <remarks>
    /// The lower the rank, the better the team. The higher the rank,
    /// the worse the team.
    /// </remarks>
    private bool IsMistake(bool isHomeTeamWinner, int homeRank, int awayRank)
    {
        bool isMistake;
```

```
        // The lower the rank, the better the team
        // The higher the rank, the worse the team

        if (isHomeTeamWinner)
        {
            isMistake = homeRank > awayRank;
        }
        else
        {
            isMistake = awayRank > homeRank;
        }

        return isMistake;
    }

    /// <summary>
    /// Gets the size of the mistake using Potemkin's equation.
    /// </summary>
    /// <param name="homeRank">The rank of the home team</param>
    /// <param name="awayRank">The rank of the away team</param>
    /// <returns>The size of the mistake</returns>
    private double GetError(int homeRank, int awayRank)
    {
        double importance;
        double size;
        double error;

        size = ratingCount * Math.Abs(homeRank - awayRank) * 1.0
               /  (homeRank * awayRank);
        importance = ratingCount * (homeRank + awayRank)
               / (homeRank * awayRank);
        error = size * importance;

        return error;
    }

    /// <summary>
    /// Computes the correlation coefficient against the final AP poll
    /// </summary>
    private void ComputeAPTop25CorrelationCoefficient()
    {
         ComputeCorrelationCoefficient(
                                 GetRankingDictionary(settings.Top25Filename),
                                 true);
    }

    /// <summary>
    /// Computes the correlation coefficient against the average FBS computer
    /// ranking
    /// </summary>
    private void ComputeAvgFBSRankingCorrelationCoefficient()
    {
        Division fbsDivision = entities.Divisions.OrderBy(d => d.ID).First();
        IEnumerable<Team> nonFbsTeams = from t in entities.Teams
                    join c in entities.Conferences on t.ConferenceID equals c.ID
                    where c.DivisionID != fbsDivision.ID
```

```
                select t;

        // Remove all non-FBS teams from the rank dictionary
        foreach (Team nonFbsTeam in nonFbsTeams)
        {
            stdRatingRankDictionary.Remove(nonFbsTeam.ID);
            hfaRatingRankDictionary.Remove(nonFbsTeam.ID);
            mpdRatingRankDictionary.Remove(nonFbsTeam.ID);
            hensleyRatingRankDictionary.Remove(nonFbsTeam.ID);
        }

        ReassignPlaces(stdRatingRankDictionary);
        ReassignPlaces(hfaRatingRankDictionary);
        ReassignPlaces(mpdRatingRankDictionary);
        ReassignPlaces(hensleyRatingRankDictionary);

        ComputeCorrelationCoefficient(
                           GetRankingDictionary(settings.AvgFBSRankingFilename),
                           false);
    }

    /// <summary>
    /// Reassign the rank of all the teams since non-FBS schools were removed
    /// </summary>
    /// <param name="ratingRankDictionary">
    /// The dictionary being reassigned
    /// </param>
    private void ReassignPlaces(Dictionary<int, int> ratingRankDictionary)
    {
        int newPlace = 1;
        foreach (int key in ratingRankDictionary.OrderBy(kv =>
                 kv.Value).Select(kv => kv.Key).ToList())
        {
            ratingRankDictionary[key] = newPlace;
            newPlace++;
        }
    }

    /// <summary>
    /// Computes the correlation coefficient for a given ranking dictionary
    /// </summary>
    /// <param name="rankDictionary">
    /// The ranking dictionary being evaluated
    /// </param>
    /// <param name="isAPTop25">
    /// True if this for the final AP poll; otherwise, false.
    /// </param>
    private void ComputeCorrelationCoefficient(
                     Dictionary<int, int> rankDictionary, bool isAPTop25)
    {
        double sumX = 0;
        double sumX2 = 0;
        double sumStdRank = 0;
        double sumHfaRank = 0;
        double sumMpdRank = 0;
        double sumHensleyRank = 0;
```

```
double sumStdRank2 = 0;
double sumHfaRank2 = 0;
double sumMpdRank2 = 0;
double sumHensleyRank2 = 0;
double sumStdRankX = 0;
double sumHfaRankX = 0;
double sumMpdRankX = 0;
double sumHensleyRankX = 0;
int elements = rankDictionary.Count;

foreach (KeyValuePair<int, int> kvPair in rankDictionary)
{
    sumX += kvPair.Value;
    sumX2 += kvPair.Value * kvPair.Value;

    sumStdRank += stdRatingRankDictionary[kvPair.Key];
    sumHfaRank += hfaRatingRankDictionary[kvPair.Key];
    sumMpdRank += mpdRatingRankDictionary[kvPair.Key];
    sumHensleyRank += hensleyRatingRankDictionary[kvPair.Key];

    sumStdRank2 += Math.Pow(stdRatingRankDictionary[kvPair.Key], 2);
    sumHfaRank2 += Math.Pow(hfaRatingRankDictionary[kvPair.Key], 2);
    sumMpdRank2 += Math.Pow(mpdRatingRankDictionary[kvPair.Key], 2);
    sumHensleyRank2 += Math.Pow(hensleyRatingRankDictionary[kvPair.Key],
                                2);

    sumStdRankX += stdRatingRankDictionary[kvPair.Key] * kvPair.Value;
    sumHfaRankX += hfaRatingRankDictionary[kvPair.Key] * kvPair.Value;
    sumMpdRankX += mpdRatingRankDictionary[kvPair.Key] * kvPair.Value;
    sumHensleyRankX += hensleyRatingRankDictionary[kvPair.Key]
                       * kvPair.Value;
}

if (isAPTop25)
{
    stdCorrelationCoefficientAPTop25 =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                                 sumStdRank, sumStdRank2, sumStdRankX);
    hfaCorrelationCoefficientAPTop25 =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                                 sumHfaRank, sumHfaRank2, sumHfaRankX);
    mpdCorrelationCoefficientAPTop25 =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                                 sumMpdRank, sumMpdRank2, sumMpdRankX);
    hensleyCorrelationCoefficientAPTop25 =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                 sumHensleyRank, sumHensleyRank2, sumHensleyRankX);
}
else
{
    stdCorrelationCoefficientAvgFBSRanking =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                                 sumStdRank, sumStdRank2, sumStdRankX);
    hfaCorrelationCoefficientAvgFBSRanking =
            GetCorrelationCoefficient(elements, sumX, sumX2,
                                 sumHfaRank, sumHfaRank2, sumHfaRankX);
```

```
            mpdCorrelationCoefficientAvgFBSRanking =
                        GetCorrelationCoefficient(elements, sumX, sumX2,
                                          sumMpdRank, sumMpdRank2, sumMpdRankX);
            hensleyCorrelationCoefficientAvgFBSRanking =
                        GetCorrelationCoefficient(elements, sumX, sumX2,
                              sumHensleyRank, sumHensleyRank2, sumHensleyRankX);
        }
    }

    /// <summary>
    /// Gets the correlation coefficient value
    /// </summary>
    /// <param name="elements">The number of elements</param>
    /// <param name="sumX">The sum of all the X values</param>
    /// <param name="sumX2">The sum of all the X^2 values</param>
    /// <param name="sumY">The sum of all the Y values</param>
    /// <param name="sumY2">The sum of all the Y^2 values</param>
    /// <param name="sumXY">The sum of all the X*Y values</param>
    /// <returns>The correlation coefficient</returns>
    private double GetCorrelationCoefficient(int elements, double sumX,
                        double sumX2, double sumY, double sumY2, double sumXY)
    {
        double r;

        r = (sumXY - sumX * sumY / elements)
            / Math.Sqrt((sumX2 - Math.Pow(sumX, 2) / elements)
                        * (sumY2 - Math.Pow(sumY, 2) / elements));
        return r;
    }

    /// <summary>
    /// Creates a ranking dictionary for a given CSV file
    /// </summary>
    /// <param name="fileName">The path and name of the CSV file</param>
    /// <returns>The ranking dictionary</returns>
    /// <remarks>
    /// The CSV file should be in the following format: Place,TeamName
    /// </remarks>
    private Dictionary<int, int> GetRankingDictionary(string fileName)
    {
        Dictionary<int, int> rankDictionary = new Dictionary<int, int>();
        string line;
        string teamName;
        string[] values;
        int place = 0;
        Team team;

        try
        {
            StreamReader sr = new StreamReader(fileName);

            while (!sr.EndOfStream)
            {
                line = sr.ReadLine();
                values = line.Split(',');
                teamName = values[1];
```

```
                try
                {
                    place = Convert.ToInt32(values[0]);
                }
                catch (Exception)
                {
                    Console.WriteLine("Invalid place, {0}, in {1}", values[0],
                                      fileName);
                    continue;
                }

                team = entities.Teams.FirstOrDefault(t => t.Name == teamName);

                if (team == null)
                {
                    Console.WriteLine("Invalid team name, {0}, in {1}",
                                      teamName, fileName);
                    continue;
                }

                rankDictionary.Add(team.ID, place);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("An error occurred during the correlation
                              coefficient calculations.");
        }

        return rankDictionary;
    }

    /// <summary>
    /// Populates the four different rank dictionaries for the different methods
    /// </summary>
    private void PopulateRatingRankDictionaries()
    {
        IEnumerable<Team> teamsOrderByStdRating;
        IEnumerable<Team> teamsOrderByHfaRating;
        IEnumerable<Team> teamsOrderByMpdRating;
        IEnumerable<Team> teamsOrderByHensleyRating;

        teamsOrderByStdRating = entities.TeamResults.OrderByDescending(
             tr => tr.StandardRating).Select(tr => tr.Team);
        teamsOrderByHfaRating = entities.TeamResults.OrderByDescending(
             tr => tr.HomefieldAdvantageRating).Select(tr => tr.Team);
        teamsOrderByMpdRating = entities.TeamResults.OrderByDescending(
             tr => tr.MaxPointDifferentialRating).Select(tr => tr.Team);
        teamsOrderByHensleyRating = entities.TeamResults.OrderByDescending(
             tr => tr.HensleyRating).Select(tr => tr.Team);

        ratingCount = teamsOrderByStdRating.Count();
        for (int i = 1; i <= ratingCount; i++)
        {
            stdRatingRankDictionary.Add(teamsOrderByStdRating.ElementAt(
```

```
                                                                    i – 1).ID, i);
                hfaRatingRankDictionary.Add(teamsOrderByHfaRating.ElementAt(
                                                                    i - 1).ID, i);
                mpdRatingRankDictionary.Add(teamsOrderByMpdRating.ElementAt(
                                                                    i - 1).ID, i);
                hensleyRatingRankDictionary.Add(teamsOrderByHensleyRating.ElementAt(
                                                                    i - 1).ID, i);
            }
        }

        /// <summary>
        /// Outputs the results to the screen and the CSV files
        /// </summary>
        private void OutputResults()
        {
            int gameCount = entities.Games.Count();

            OutputToScreen(StandardRating.RatingName, stdRatingMistakeCount,
                           stdRatingError, stdCorrelationCoefficientAPTop25,
                           stdCorrelationCoefficientAvgFBSRanking, gameCount);
            OutputToScreen(HomeFieldAdvantageRating.RatingName,
                           hfaRatingMistakeCount, hfaRatingError,
                           hfaCorrelationCoefficientAPTop25,
                           hfaCorrelationCoefficientAvgFBSRanking, gameCount);
            OutputToScreen(MaxPointDifferentialRating.RatingName,
                           mpdRatingMistakeCount, mpdRatingError,
                           mpdCorrelationCoefficientAPTop25,
                           mpdCorrelationCoefficientAvgFBSRanking, gameCount);
            OutputToScreen(HensleyRating.RatingName, hensleyRatingMistakeCount,
                           hensleyRatingError, hensleyCorrelationCoefficientAPTop25,
                           hensleyCorrelationCoefficientAvgFBSRanking, gameCount);

            try
            {
                StreamWriter sw = new StreamWriter(settings.OutputFilename);
                sw.WriteLine("Rating Name,Number of Mistakes,Total Error,Correlation
                             To Top 25,Correlation to Average FBS Computer
                             Rankings");
                WriteResultLine(sw, StandardRating.RatingName,
                                stdRatingMistakeCount, stdRatingError,
                                stdCorrelationCoefficientAPTop25,
                                stdCorrelationCoefficientAvgFBSRanking, gameCount);
                WriteResultLine(sw, HomeFieldAdvantageRating.RatingName,
                                hfaRatingMistakeCount, hfaRatingError,
                                hfaCorrelationCoefficientAPTop25,
                                hfaCorrelationCoefficientAvgFBSRanking, gameCount);
                WriteResultLine(sw, MaxPointDifferentialRating.RatingName,
                                mpdRatingMistakeCount, mpdRatingError,
                                mpdCorrelationCoefficientAPTop25,
                                mpdCorrelationCoefficientAvgFBSRanking, gameCount);
                WriteResultLine(sw, HensleyRating.RatingName,
                                hensleyRatingMistakeCount, hensleyRatingError,
                                hensleyCorrelationCoefficientAPTop25,
                                hensleyCorrelationCoefficientAvgFBSRanking,
                                gameCount);
                sw.Close();
```

```
    }
    catch (Exception)
    {
        Console.WriteLine("An error occurred during the writing of the
                           results.");
    }

}

/// <summary>
/// Outputs the results to the screen
/// </summary>
/// <param name="ratingName">The name of the rating system</param>
/// <param name="mistakeCount">The number of mistakes</param>
/// <param name="totalError">The total error of the mistakes</param>
/// <param name="correlationCoefficientAPTop25">
/// The correlation coefficient to the final AP poll
/// </param>
/// <param name="correlationCoefficientAvgFBSRanking">
/// The correlation coefficient to the average FBS ranking
/// </param>
/// <param name="gameCount">The number of games played</param>
private void OutputToScreen(string ratingName, int mistakeCount,
    double totalError, double correlationCoefficientAPTop25,
    double correlationCoefficientAvgFBSRanking, int gameCount)
{
    Console.WriteLine("Results for {0}", ratingName);
    Console.WriteLine("-----------------------------------------");
    Console.WriteLine("Number of mistakes = {0} ({1}%)", mistakeCount,
                      Math.Round(mistakeCount * 100.0 / gameCount, 1));
    Console.WriteLine("Total error = {0}", Math.Round(totalError));
    Console.WriteLine("Correlation coefficient to Top 25 = {0}",
                      Math.Round(correlationCoefficientAPTop25, 2));
    Console.WriteLine("Correlation coefficient to Avg FBS Ranking = {0}",
                      Math.Round(correlationCoefficientAvgFBSRanking, 2));
    Console.WriteLine();
}

/// <summary>
/// Outputs the results to the screen
/// </summary>
/// <param name="sw">The CSV file being written to</param>
/// <param name="ratingName">The name of the rating system</param>
/// <param name="mistakeCount">The number of mistakes</param>
/// <param name="totalError">The total error of the mistakes</param>
/// <param name="correlationCoefficientAPTop25">
/// The correlation coefficient to the final AP poll
/// </param>
/// <param name="correlationCoefficientAvgFBSRanking">
/// The correlation coefficient to the average FBS ranking
/// </param>
/// <param name="gameCount">The number of games played</param>
private void WriteResultLine(StreamWriter sw, string ratingName,
    int mistakeCount, double totalError,
    double correlationCoefficientAPTop25,
    double correlationCoefficientAvgFBSRanking, int gameCount)
```

```
            {
                sw.WriteLine("{0},{1} ({2}%),{3},{4},{5}", ratingName, mistakeCount,
                             Math.Round(mistakeCount * 100.0 / gameCount, 1),
                             Math.Round(totalError),
                             Math.Round(correlationCoefficientAPTop25, 2),
                             Math.Round(correlationCoefficientAvgFBSRanking, 2));
            }
        }
    }
```