December 2014

A Markov Model for Baseball with Applications

Daniel Joseph Ursin
University of Wisconsin-Milwaukee

A MARKOV MODEL FOR BASEBALL WITH APPLICATIONS

by

Daniel Ursin

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

December 2014

ABSTRACT

A MARKOV MODEL FOR BASEBALL WITH APPLICATIONS

by

Daniel Ursin

The University of Wisconsin-Milwaukee, 2014
Under the Supervision of Professor Mukul Goyal, Ph.D.

In this work we confirm a Markov chain model of baseball for 2013 Major League Baseball batting data. We describe the transition matrices for individual player data and their use in generating single and nine-inning run distributions for a given lineup. The run distribution is used to calculate the expected number of runs produced by a lineup over nine innings. We discuss batting order optimization heuristics to avoid computation of distributions for the $9! = 362,880$ distinct lineups for 9 players. Finally, we describe an implementation of the algorithms and review their performance against actual game data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction and Definitions

The game of baseball has been studied by statisticians for decades to varying degrees, and today's teams are more interested in accurate data models than ever before. In this work, we confirm the 2013 Major League Baseball batting records with a Markov chain model of baseball developed by Bukiet [3]. A *Markov chain* is a stochastic process with the Markov property. We say that a stochastic process has the *Markov property* if the probability distribution of future states, conditional on the past and present values of the states, depends only on the current state, and not on any previous state. A Markov chain is completely determined by its transition matrix and initial distribution.

The progression of a baseball game is very easily understood in terms of a Markov chain. Each state represents the number of outs so far recorded in the inning and the current locations of base-runners, and there is a transition each time a new batter's at-bat comes to an end. In the game of baseball, at the start of every inning, there are no outs and no runners on the bases. This no-out, no-runner state is the start-state each inning of every game. In general, when a baseball player comes to the plate, he is inevitably in one of 24 possible situations:

- There are 0, 1 or 2 outs (3 possibilities)
- There are runners distributed on the bases in 8 possible configurations
  - The bases are empty (1 way)
  - There is exactly one runner on first, second, or third (3 ways)
  - There are exactly two runners on first, second, or third (3 ways)
  - The bases are loaded (1 way)

Finally, we have a three-out absorbing state giving a total of $(3 \times 8) + 1 = 25$ states. In baseball, when three outs are recorded, the inning is over and the next inning begins in the no-runner, no-out state for the opposing team.

For each player, we construct a transition probability matrix based on widely available batting statistics. Suppose for example that there are no outs and no runners on base. The probability that after the current at-bat is complete the system will be in the no-out, runner-on-second state is the probability that the batter hit a double. The probability that after the current at-bat is complete the system will be in the 1-out state with no runners is the probability of that player

recording an out by strikeout, fly-out, etc. These probabilities are easily calculated from available batting records.

Once the probabilities have been entered, we normalize each row to ensure that the matrix is stochastic. Determining the probability of being in a particular state after a number of batters can be calculated by simply multiplying the transition probability matrices of each player in the lineup in the order in which they bat and left-multiplying the initial distribution vector $v_0 =< 1, 0, ..., 0 >$. That is, we begin each inning in the no-out, no-runner state. There are a few subtleties in keeping track of runs scored but this is the essence of the Markov model.

1.1. **Offensive Possibilities.** In order to determine how many runs are scored given a particular transition, we assume that at each at-bat, the following six actions are possible [3]:

- Out - Runners do not advance. Outs increase by one.
- Walk - Batter moves to first base. Other base runners advanced when forced.
- Single - Batter moves to first base. Runner at first advances to second. Runners at second or third score.
- Double - Batter moves to second base. Runner at first advances to third. Runners at second or third score.
- Triple - Batter moves to third base. All other runners score.
- Home Run - All base runners and batter score, clearing the bases.

The assumption that runners cannot advance on an out may be concerning to baseball fans. It is possible in baseball for a runner to advance on an out during a sacrifice fly or successful bunt. However, the simplicity of the model makes computation efficient and the results are very accurate over the course of a season, even without such concern.

With regard to a walk, a base runner is said to be forced if the advancement of the previous runner would cause two players to be at the same base. Suppose for example there are runners on first and third and the batter is issued a walk. Then the runner on first base is forced to advance to second, since the batter is advancing to first. The runner on third does not advance because the base runner now at second has already advanced, and is not in danger of occupying the same base as the runner at third.

Runners on second or third are traditionally said to be in *scoring position*, meaning that any base hit (single, double, triple, or home run) will score them.

One of the most popular aspects of baseball is the base-clearing home run. A home run scores a minimum of one run (the batter), and a maximum of four runs (the batter and base runners on first, second, and third). A four-run home run is called a *Grand Slam* and is fairly rare due to a variety of reasons, such as pitcher's strategy, rarity of having the bases loaded, and the unavoidable low percentage of hitting a home run in the first place. We make the assumption that the probability of a player recording any of the above offensive possibilities is constant throughout the game and does not depend on the position or quantity of the base runners.

2. Previous Work

The usefulness of Markov chain models to analyze baseball has been evident for many years. Bellman [1] described the basic framework of such models and gave a sketch of a dynamic programming approach in 1976 which would allow a manager to optimize his run production based on a defined decision policy. In Bellman's Markov model there are 2592 states describing the possible situations a batter can face.

The number $2592 = 3 \times 4 \times 3 \times 8 \times 9$ comes from the 3 outs, 4 balls, 3 strikes, 8 base-runner configurations, and 9 available batters that are possible at the pitch-by-pitch level. While very interesting and certainly an attractive resource for a baseball manager, implementing such a system is complex and requires respectable computing power.

In 1960, D'Esopo and Lefkozwitz [7] employed 25 states and used an averaging technique over all players on a team or league and used a single, constant transition matrix for their calculations and then studied the steady-state behavior of the transition matrix.

Later authors, such as Bukiet [3] reduce the state space to a mere 25, a far more manageable number of possibilities. This is the basis of the model that we employ in this work. Remarkably, even with this small number of states, highly useful models can be developed. Bukiet used batting data from the 1989 National League teams to create individual player transition matrices. The run distribution was then used to calculate the expected number of wins in a season and to find optimal batting orders.

We implemented the Markov model and ran simulations for all of the teams in Major League Baseball for 2013. The hypothesis that we investigate for the model is that the results of the model simulations closely approximates actual season data for 2013.

3. Player Transition Matrices

The 25×25 transition matrix can be simplified through the use of blocks. In particular, we have the following structure:

$$\mathrm{P} = \begin{pmatrix} A_0 & B_0 & C_0 & D_0 \\ \mathbf{0} & A_1 & B_1 & E_1 \\ \mathbf{0} & \mathbf{0} & A_2 & F_2 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{pmatrix}$$

where the A, B, and C matrices are 8×8 blocks containing transition probabilities which update the number of outs from zero to zero, one, and two outs respectively. The D, E, and F matrices are 8×1 column vectors which hold probabilities that take the inning to the three-out state. D_0 for example holds the probabilities that an at-bat begins with zero outs and transitions into the three-out state. That is, the batter hits into a triple play. Similarly, E_1 holds the transition probabilities of hitting into a double play from a 1-out state, and F_2 holds the transition probabilities of moving from a two-out to a three-out state. Finally, the bottom row consists of three 1×8 row vectors and a 1×1 matrix holding the single value, unity. This represents that the three-out state is an absorbing one. If we are in a three-out state, we will stay in this state with certainty. The block 8×8 zero matrices indicate that it is impossible to transition into a state with fewer outs. Given the batting record of a player, we construct a 25×25 transition matrix which associates the available offensive options listed above with the individual season statistics of the player. We can visualize the first block column as transitions to states with no outs, and the second, third and fourth columns as holding transitions to states with one, two and three outs respectively.

3.1. **A - Matrices.** Here and below, O represents the number of outs in the block. The ordered pair $(O, S), O \in \{0, 1, 2, 3\}, S \in \{0, 1, 2, 3, 12, 13, 23, 123\}$ represents the state with O outs and with runners in configuration S. A-Matrices have the following form before normalization:

$$
\begin{array}{c}
 \\
(O,0) \\
(O,1) \\
(O,2) \\
(O,3) \\
(O,12) \\
(O,13) \\
(O,23) \\
(O,123)
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
(O,0) & (O,1) & (O,2) & (O,3) & (O,12) & (O,13) & (O,23) & (O,123)
\end{array} \\
\left(\begin{array}{cccccccc}
p_{HR} & (p_{1B}+p_{BB}) & p_{2B} & p_{3B} & 0 & 0 & 0 & 0 \\
p_{HR} & 0 & 0 & p_{3B} & (p_{1B}+p_{BB}) & 0 & p_{2B} & 0 \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & p_{BB} & 0 & 0 & 0 \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & p_{BB} & 0 & 0 \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & 0 & 0 & p_{BB} \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB}
\end{array}\right)
\end{array}
$$

$$p_{1B} = \frac{Hits-(Doubles+Triples+HomeRuns)}{AtBats},$$

$$p_{2B} = \frac{Doubles}{AtBats}, p_{3B} = \frac{Triples}{AtBats},$$

$$p_{BB} = \frac{Walks}{AtBats}, p_{Hr} = \frac{HomeRuns}{AtBats}$$

These are transitions that do not increase the number of outs. Traditionally, baseball statistics include the number of total hits, doubles, triples and home runs. They do not typically include singles specifically. To obtain this number, we simply complement from the total number of hits.

3.2. **B - Matrices.** B-Matrices have the following form before normalization:

$$B = p_{Out} \cdot I_8$$

That is, B is an 8×8 matrix with p_{Out} along the diagonal and zero elsewhere.

These are transitions from a no-out state to a one-out state, or from a one-out state to a two-out state. Note that this assumes that runners do not advance on an out. There are situations in baseball where a runner may score on a play in which there is an out. A force-out at first can occur while a runner scores from third, a runner may tag up and run home after a fly ball is caught deep in the outfield, and many other scenarios are possible. This assumption frees us from having to consider base stealing and other hard-to-describe events. To consider stealing, we would need to maintain states that encode which players are at which bases, so that we could calculate the probability that that player will attempt to steal and either succeed or fail. This would greatly increase the number of necessary states.

3.3. **C - Matrices.** C-Matrices have the following form before normalization:

$$
\begin{array}{c} \\ (0,0) \\ (0,1) \\ (0,2) \\ (0,3) \\ (0,12) \\ (0,13) \\ (0,23) \\ (0,123) \end{array}
\begin{array}{c}
\begin{array}{cccccccc} (2,0) & (2,1) & (2,2) & (2,3) & (2,12) & (2,13) & (2,23) & (2,123) \end{array} \\
\left(\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array} \right)
\end{array}
$$

$$p_{DP} = \frac{DoublePlays}{AtBats}$$

These are transitions from a no-out state to a two-out state. The first entry is zero since at least one other runner besides the batter is required to record two outs. The next three are simply the probability that the current batter hits into a double play, an available statistic. Note that when this happens, both players are eliminated as base runners. Other transitions are not populated as the location of the runners after such a play can vary from play to play. If one were interested, placing positive probabilities in other transitions would be as simple as deciding what statistic to use and what transitions would result from such a play.

3.4. **D - Matrices.** D-Matrices have the following form before normalization:

$$D = p_{TP} \cdot < 0,0,0,0,1,1,1,1 >^T$$

$$p_{TP} = \frac{AllTimeTriplePlays}{AtBats} \approx \frac{692}{14{,}000{,}000} \approx .000049$$

There have been only 692 triple plays in major league baseball history, and there have been more than 14 million at-bats. We simply use a constant and assume that it is equal for all players.

3.5. **E - Matrices.** E-Matrices have the following form before normalization:

$$E = p_{DP} \cdot < 0,1,1,1,1,1,1,1 >^T$$

$$p_{DP} = \frac{DoublePlays}{AtBats}$$

Similar to the C - Matrices, the first entry here is zero since there must be at least one other runner besides the batter to record two outs.

3.6. **F - Matrices.** F-Matrices have the following form before normalization:

$$F = p_{Out} \cdot < 1,1,1,1,1,1,1,1 >^T$$

$$p_{Out} = \frac{Outs}{AtBats}$$

When a player is batting in a two-out state, any out will cause a transition to the three-out state.

3.7. **Full Transition Matrix.** The full 25×25 player transition matrix is made up of the A, B, C, D, E, and F matrices and can be seen in full in the Figure 19.

4. Computing Run Distributions

In order to determine the distribution of runs for a particular lineup of 9 players, we need to know the player transition matrices, as well as the transitions that will cause 0, 1, 2, 3, or 4 runs to score during the course of one at-bat. We have seen how to construct a player's transition matrix, and we turn to the construction of the scoring matrices which sum to the player transition matrix. Since runners do not advance in our model, we need only consider the A-type matrix and its transitions. The following states could have zero, one or two outs. The number of runs scored on transition from state i to state j is

$$\mathrm{R}_{i,j} = 1 + (Number\ of\ Base\ Runners(i) - Number\ of\ Base\ Runners(j))$$

Thus for each transition in the A-matrices, the following number of runs will be scored

$$\mathrm{R} = \begin{array}{c} \\ (0) \\ (1) \\ (2) \\ (3) \\ (12) \\ (13) \\ (23) \\ (123) \end{array} \begin{array}{c} \begin{array}{cccccccc} (0) & (1) & (2) & (3) & (12) & (13) & (23) & (123) \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 2 & 2 & 2 & 1 & 1 & 1 & 0 \\ 3 & 2 & 2 & 2 & 1 & 1 & 1 & 0 \\ 3 & 2 & 2 & 2 & 1 & 1 & 1 & 0 \\ 4 & 3 & 3 & 3 & 2 & 2 & 2 & 1 \end{pmatrix} \end{array}$$

We break R into $R = R_0 + R_1 + R_2 + R_3 + R_4$ where R_i is zero except where $R_{j,k} = i$. Then, we create five matrices, $P_0, ..., P_4$ representing the transitions from the transition matrix P where i runs are scored for $i \in \{0, 1, 2, 3, 4\}$. We determine P_1, P_2, P_3, P_4 by using R_1, R_2, R_3, R_4. P_0, the matrix which holds transitions that do not score a run is calculated as the complement of the P_i from P. That is, $P_0 = P - (P_1 + P_2 + P_3 + P_4)$. Now we have written $P = P_0 + P_1 + P_2 + P_3 + P_4$, and are prepared to calculate the number of runs scored during transitions.

4.1. **One-Inning Run Distribution.** Below we present the algorithm for generating a one-inning distribution for 9 batters. We follow the convention by Bukiet [3] that the maximum number of runs scored in an inning is 20. This is reasonable

since the highest scoring inning in modern baseball history (1900 - present) scored 19 runs in 1953 in the 7^{th} inning of a game between the Boston Red Sox and Detroit Tigers. Later we will limit the number of runs in an entire game to 20 runs. This is an arbitrary limit on the number of runs that keeps the computations within a reasonable scope and run-time.

In the algorithm, we iterate over the player transition matrices in the given lineup until the probability of three outs is within ϵ of unity. In our calculations, we used $\epsilon = 0.0001$. We use the 25×25 player transition matrices, as well as a non-stochastic, 21×25 score-keeping matrix, which we denote U. U has 21 rows since a team can score between 0 and 20 runs in a single game and is initialized to all zeroes except for unity as its first entry. This represents that before any batters have come to bat, the game will be in a state with no runs, no outs, and no runners aboard with certainty. There are 25 columns since the game is played over 25 possible states.

Consider the beginning of the game, before any batters have come to bat and no transitions have taken place. At this time, U is zero everywhere except a one in the first entry. We proceed to build a temporary score-keeping matrix, *temp*, which will become U after the first batter has completed his at-bat. Row zero of *temp* is the product of the first row of U, $u_0 := < 1, 0, \ldots, 0 >$, with the scoring matrix, P_0, since the P_0 matrix contains transitions that do not score any runners. This includes outs, and any hit which does not cause a runner to score. Row one of this new matrix is the product of u_0 with the scoring matrix P_1, since the P_1 matrix contains transitions that score one runner. Rows two, three, and four are the products of u_0 with P_2, P_3 and P_4, respectively and are zero because this is the first batter to come to bat, so it is impossible to score two or more runs in this at-bat. In general, a score-keeping matrix will only have positive entries in rows 0 through n, where n is the number of batters who have had an at-bat this inning. This is simply because you cannot score more runs than you have had batters up to bat.

Whereas for the first batter, U will have positive entries only in the first row (meaning no runs have scored), for the second batter U can have positive entries in the first *or* second rows. We build *temp* from U and the scoring matrices of the current batter, row by row as before. The final column of U is still all zeroes because

it is impossible to be in a three-out state where there has been only one batter. Only after three batters can there be any positive entries in the final column of U, since the final column represents transitions to the three-out state. We proceed this way for each player in the lineup, until the sum of the 25th column is within ϵ of 1. This column vector is the single-inning run distribution for the lineup.

Algorithm: $SingleInningRunDistribution(T, \text{ref } Idx)$
Data: Player Transition Matrices: T, Starting Batter Index: Idx
Result: One-inning run distribution
$U \leftarrow empty\ 21 \times 25\ matrix$
$U[0,0] \leftarrow 1$
while $\|U.FinalColumnSum - 1\| > \epsilon$ **do**
  $Idx \leftarrow (Idx\ \%\ T.Length)$
  $P_0, P_1, P_2, P_3, P_4 \leftarrow ToScoringMatrices(T.ElementAt(Idx))$
  $temp \leftarrow empty\ 21 \times 25\ matrix$
  for $rowIdx \leftarrow 0$ **to** 20 **do**
    $row \leftarrow U.GetRow(rowIdx) \times P_0$
      $+ U.GetRow(rowIdx - 1) \times P_1$
      $+ U.GetRow(rowIdx - 2) \times P_2$
      $+ U.GetRow(rowIdx - 3) \times P_3$
      $+ U.GetRow(rowIdx - 4) \times P_4$
    /* Of course, if $rowIdx - i < 0$, that addend is zero */
    $temp.SetRow(rowIdx, row)$
  end
  $U \leftarrow temp$
  $Idx \leftarrow Idx + 1$
end
return $U.FinalColumn$

Algorithm 1: Compute a Single-Inning Run Distribution for a Lineup

4.2. **Nine Inning Run Distributions.** To compute the nine-inning run distribution for a lineup of 9 batters, we simply run the one-inning run distribution algorithm 9 times, keeping track of the first batter index so we truly cycle through the available batters. The result will be nine one-inning distributions, which when treated as independent random variables, can be summed to obtain the nine-inning run distribution. The sum of two independent random variables is simply the convolution of their probability vectors, and we sum all nine single-inning distributions to obtain a single stochastic vector holding the probability that a lineup scores i runs for $i \in \{0, 1, ..., 20\}$.

Algorithm: $NineInningRunDistribution(T)$
Data: Player Transition Matrices: T
Result: Nine-inning run distribution
Create empty list L of vectors
$Idx \leftarrow 0$
for $inning \leftarrow 1$ **to** 9 **do**
  $distribution \leftarrow SingleInningRunDistribution(T, ref\ Idx)$
  $L.Add(distribution)$
end
return $SumDistributions(L)$

Algorithm 2: Compute the Nine-Inning distribution for a Lineup

5. Applications of the 9-Inning Run Distribution

Once a nine-inning run distribution has been computed for a particular lineup, we turn to applications of that stochastic vector. Firstly, by having the probabilities that a team scores i runs, we can immediately state the expected number of runs produced in a single game.

5.1. **Expected Number of Runs.** Given a run distribution, D, for a particular lineup, we compute the expected number of runs as

$$E(D) = \sum_{i=0}^{20} i \cdot p\,[i]$$

This is a simple calculation that can be done as soon as a run distribution has been calculated. In the results section, we look at the accuracy of the model for the American League versus the National League for the 2013 Major League Baseball season.

5.2. **Expected Number of Wins over a Series/Season.** While simply calculating the expected number of runs produced by a lineup of batters is a useful way to measure strength of a lineup, we are often interested in the performance of a team over the course of an entire, 162 game, season. By computing the run distributions for each opponent in your team's schedule, you can then compute the probability of each team winning the game. Of course, in order to win the game a team must score more runs than its opponent, and we have all of the information we need about run scoring from the pre-computed run distributions. If D_1 and D_2 are the run distributions of Teams 1 and 2 respectively, then the probability of Team 1 scoring more runs than Team 2 and hence winning the game is

$$P_{Win} = \sum_{i=1}^{20} \left[D_1[i] \cdot \sum_{j=0}^{i-1} D_2[j] \right]$$

Throughout a season, teams will play series of between 2 and 4 games against one opponent. The number of expected wins over an n-game series is $E_{Wins} = n \cdot P_{Win}$. By doing this calculation for every series your team plays over a season, you can determine the expected number of wins in your 162 game season. An important assumption that we make is that the lineup remains the same in every game throughout the season. In calculating the expected number of wins over a season, we are assuming that a player's transition matrix is also constant throughout the season.

5.3. **Optimal Batting Orders.** Traditionally, a batting order is crafted using a few basic rules:

- 1^{st}: Called the *leadoff*, a player with a high on-base percentage
- 2^{nd}: A player who can move base-runners by contact-hitting, bunt, or sacrifice
- 3^{rd}: Called the *three-hole*, typically one of the best batters in the lineup
- 4^{th}: Called the *clean-up* position, usually a power hitter
- $5^{th} - 7^{th}$*: Decreasing order of batting ability*
- 8^{th} *or* 9^{th}*: The pitcher (or worst batter) typically bats in the ninth position*

The first advantage of having your best players at the top of the batting order is that they will get an extra at-bat during the game. Batters high in the order attempt to get on base and allow a power hitter (the clean-up batter) to drive them home with power. The worst batter is placed at the bottom of the order to attempt to contain the low performance of the batter. From time to time, baseball managers experiment with different ordering strategies but much of this decision comes down to superstition and accepted baseball wisdom. An advantage of our model is that it allows one to quickly test the difference between two lineups to determine which is superior.

In the worst case, there are $9! = 362880$ permutations of the nine batters in the lineup. In our simulations, calculation of the run distribution for a single lineup takes roughly 100 milliseconds. At that rate, the calculation of all 9! possible orders takes more than 10 hours of computation time. Bukiet [3] and Pankin [4] both give criteria-based methods for assigning batting positions. For Pankin, batting positions are determined using common batting statistics such as on-base percentage, slugging percentage, and ability to draw a walk or steal. This approach is very simple and produces results comparable to the guidelines presented above. Bukiet placed batters by considering their *scoring index*, computed by determining

the expected number of runs produced with a lineup consisting of nine copies of the player in question.

For a real baseball manager, the placement of many of the batters is already set and one has only to decide where to place a few batters. In this case, exhaustively calculating the expected run productions of each lineup is well within modern computing abilities. For example, if the task is to simply place a single player into a lineup, calculating these 9 possibilities takes less than a second at the rate of 100 milliseconds per lineup.

6. Results and Data

The model enjoys a high level of accuracy for the run distribution and expected number of runs produced but does not accurately predict the number of wins that a team will enjoy during their season. We used data from Sean Lahman's 2013 [6] baseball database in all calculations. For each of the 30 teams in the MLB in 2013, we constructed lineups of the top 9 players by at-bats and ordered the batters from most at-bats to least. This is a simple way to construct a representative lineup for the sake of a season-long simulation.

As in previous models, we found that the expected number of runs produced by a team was on average lower than the actual game data reflects. Authors, Pankin [4] and Bukiet [3] suggest that the conservative offensive possibilities contributes to this effect. In particular, the fact that runners do not advance on an out in our model is a simplification that may be a part of the difference. The 2013 results for runs scored by league are presented in figures 1 and 2.

The simulated data closely approximates the actual run production of the American and National League teams for 2013, thus confirming our hypothesis. We find that the model can be used to accurately compute the expected number of runs produced by a particular lineup of nine batters.

In the American League, the model predicts the most runs for Detroit and Boston. These two teams were ultimately in contention for the World Series in 2013. High runs per game were also predicted for St. Louis and the LA Dodgers (top teams in the National League), but were not as striking as Boston and Detroit.

6.1. **Implementation of the Model.** To run simulations and interact with the massive amount of player data, a web application was built using .NET MVC 4 and Microsoft SQL Server [8]. To this end, a user can create a lineup of nine batters using player batting data from any year that the player has been active. This way, one can quickly determine the expected number of runs produced by a particular lineup. A lineup can be reordered and tested for superiority over another lineup.

The data access layer is provided through Entity Framework 5.0 and accesses the SQL tables provided by Sean Lahman [6] as well as other tables created for use with the application. The data is static and contains records from every year of Major League baseball since the late 1800s up to and including the 2013 season. An improvement would be to run the simulations using live data where a player's probabilities could change as a result of actual at-bat results. This kind of socket connection is available for a high price and is used by the Major League's own websites and mobile applications. Note that the application built for this work is available only for specific IP addresses for security.

7. Conclusions and Future Work

This model provides the mechanics for creating accurate transition matrices for every player, as soon as data is available. We were able to determine the expected number of runs expected for a particular lineup of 9 players. This is a useful, high-level analysis of the game provides accurate results when applied at a per-series or per-season level. We compared the predicted number of runs produced by a lineup with the full-season data for 2013 and examined differences between the results in the American League and National League.

An attractive improvement would be the application of the decision process described by Bellman [1]. The difficulty of this approach is the vast number of states that are involved when inning number, pitch count, and base-runner positions are included in the state definitions. The model proposed by Bellman is a pitch-by-pitch model. A reasonable relaxation in requirements would be to use the beginning of each at-bat as the decision epochs, rather than the beginning of each pitch. These stats would be more widely available and be able to ignore the pitch count in the calculation. Statistics on players stealing bases and performing with batters

in particular positions is becoming more available but is still costly for real-time information.

By removing the constraint that a team cannot score on an out, more realistic game data my be possible. One of the most attractive aspects of this model is the flexibility of the player transition matrix. Whenever more up-to-date batting data is available, it can be added to the player transition matrix. Base running data such as steals could be added to the transition matrix to give positive value to the probability of a runner advancing on an out. There are other situations that runners may advance or even score on an out, such as an error, sacrifice, bunt, or deeply hit fly ball. Determining the probabilities that runners advance to particular bases on events like steals and errors would not be based on widely-available statistics and is hence not implemented in this model.

Though many modern baseball teams employ statisticians to collect and analyze player data, baseball managers are still the last word in strategy decisions. With the methods in this work, it is hoped that more applications will be built to implement the lineup analysis. By providing more data to teams at trade deadlines or substitutions late in the game, this model allows a manager to quickly determine if he is playing his players in the proper order.

TABLE 1. 2013 National League Run Production

National League			
Team	Actual RPG	Simulated RPG	Difference
Washington	4.056	4.275	0.219
Atlanta	4.253	4.123	-0.13
Miami	3.173	3.035	-0.138
NY Mets	3.827	3.662	-0.165
Philadelphia	3.772	3.756	-0.015
Milwaukee	3.957	3.704	-0.252
St. Louis	4.84	4.051	-0.788
Pittsburgh	3.92	3.867	-0.053
Cincinnati	4.315	3.86	-0.454
Chicago Cubs	3.698	3.041	-0.657
LA Dodgers	4.012	4.391	0.378
San Francisco	3.889	3.966	0.077
San Diego	3.821	3.657	-0.164
Arizona	4.235	3.911	-0.324
Colorado	4.364	4.515	0.151
AVERAGE	4.009	3.854	-0.154
STDEV	0.362	0.406	0.304

TABLE 2. 2013 American League Run Production

American League			
Team	Actual RPG	Simulated RPG	Difference
Baltimore	4.605	4.106	-0.499
NY Yankees	4.019	3.469	-0.55
Toronto	4.401	4.123	-0.278
Tampa Bay	4.301	3.989	-0.311
Boston	5.272	4.548	-0.723
Kansas City	4.006	3.519	-0.487
Detroit	4.92	4.761	-0.159
Cleveland	4.605	3.796	-0.809
Chicago Sox	3.722	3.527	-0.195
Minnesota	3.796	3.675	-0.121
LA Angels	4.531	4.091	-0.44
Oakland	4.741	4.086	-0.655
Seattle	3.858	3.907	0.049
Houston	3.772	3.529	-0.243
Texas	4.485	3.821	-0.663
AVERAGE	4.336	3.929	-0.406
STDEV	0.448	0.365	0.241

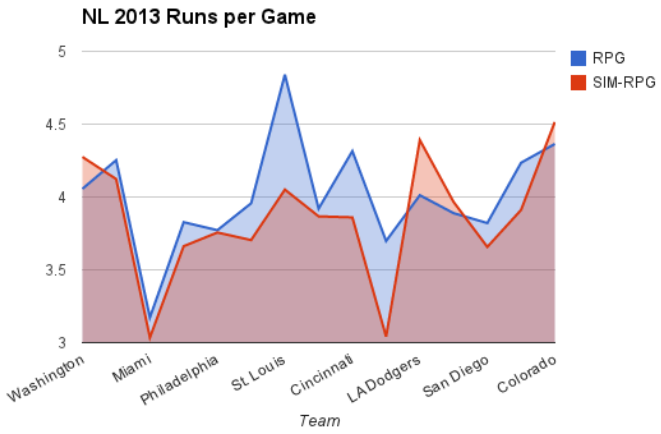

FIGURE 1. National League Runs per Game

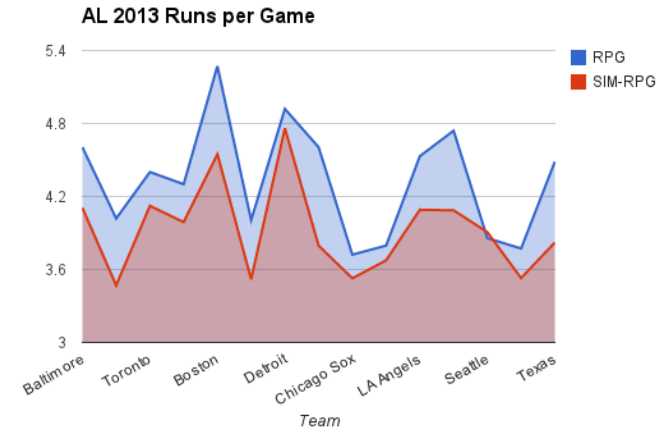

FIGURE 2. American League Runs per Game

FIGURE 3. AL - 2013 Baltimore and Boston Run Distributions. All run outcomes are discrete.

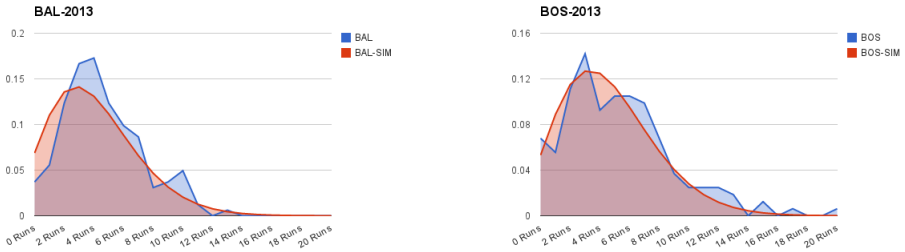

FIGURE 4. AL - 2013 Chicago and Cleveland Run Distributions. All run outcomes are discrete.

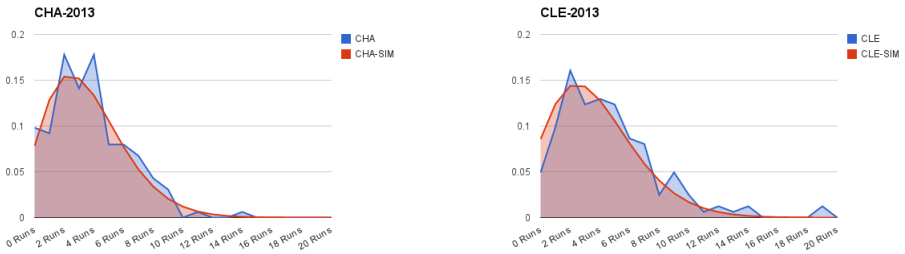

FIGURE 5. AL - 2013 Detroit and Houston Run Distributions. All run outcomes are discrete.

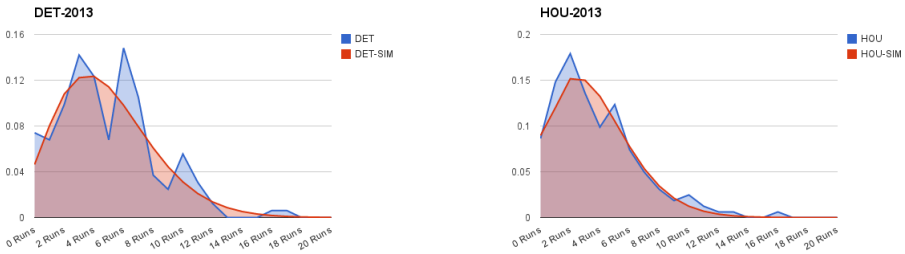

FIGURE 6. AL - 2013 Kansas City and Los Angeles Run Distributions. All run outcomes are discrete.

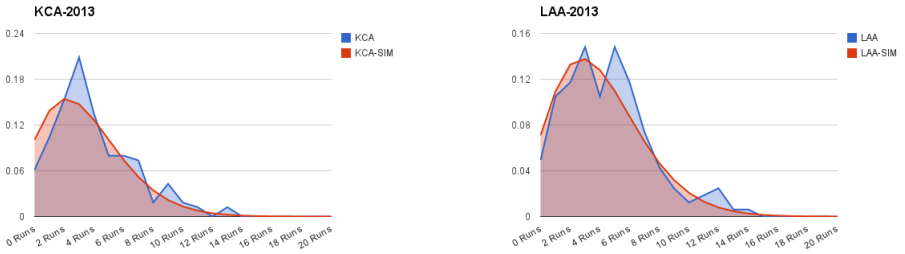

FIGURE 7. AL - 2013 Minnesota and New York Run Distributions. All run outcomes are discrete.

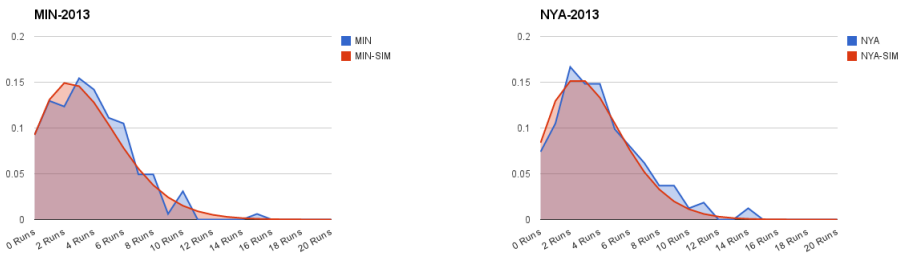

FIGURE 8. AL - 2013 Oakland and Seattle Run Distributions. All run outcomes are discrete.

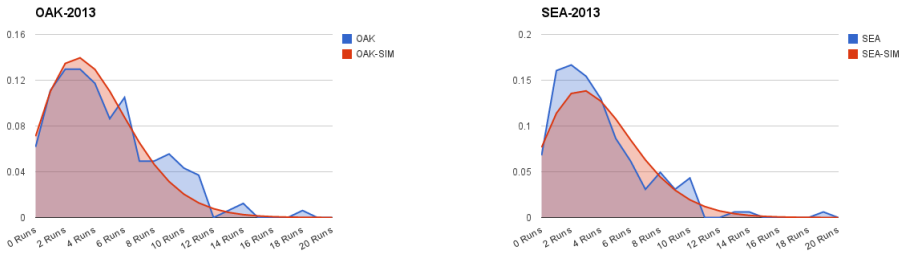

FIGURE 9. AL - 2013 Tampa Bay and Texas Run Distributions. All run outcomes are discrete.

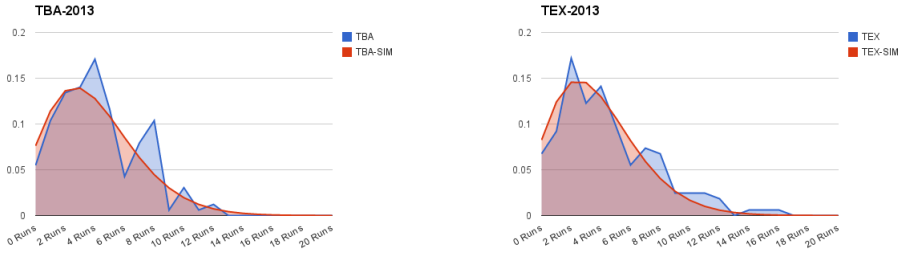

FIGURE 10. AL - 2013 Toronto Run Distribution. All run outcomes are discrete.

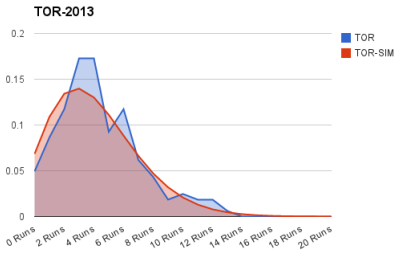

FIGURE 11. NL - 2013 Arizona and Atlanta Run Distributions. All run outcomes are discrete.

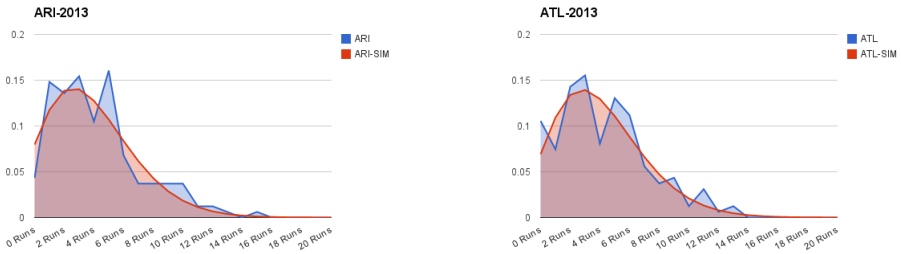

FIGURE 12. NL - 2013 Chicago and Cincinatti Run Distributions. All run outcomes are discrete.

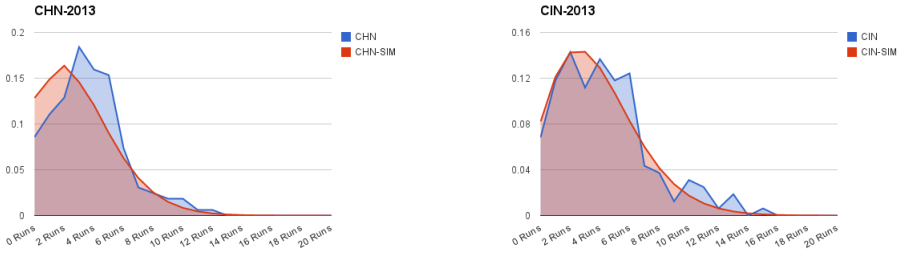

FIGURE 13. NL - 2013 Colorado and Los Angeles Run Distributions. All run outcomes are discrete.

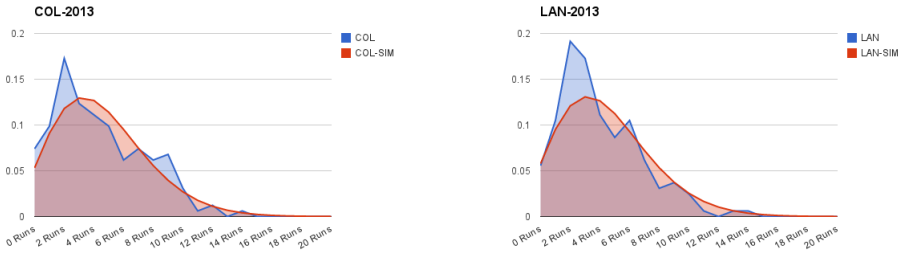

FIGURE 14. NL - 2013 Miami and Milwaukee Run Distributions. All run outcomes are discrete.

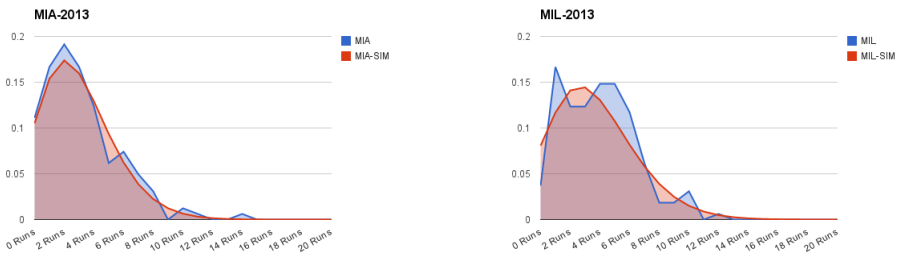

FIGURE 15. NL - 2013 New York and Philidelphia Run Distributions. All run outcomes are discrete.

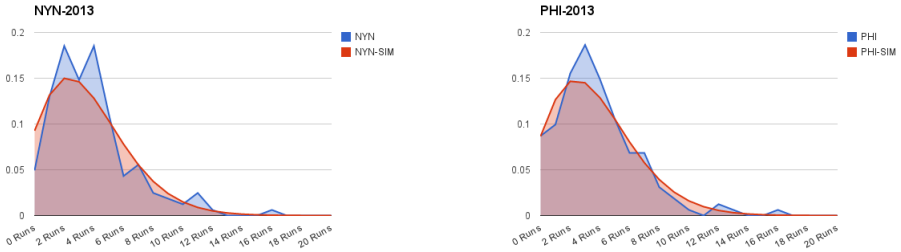

FIGURE 16. NL - 2013 Pittsburgh and San Diego Run Distributions. All run outcomes are discrete.

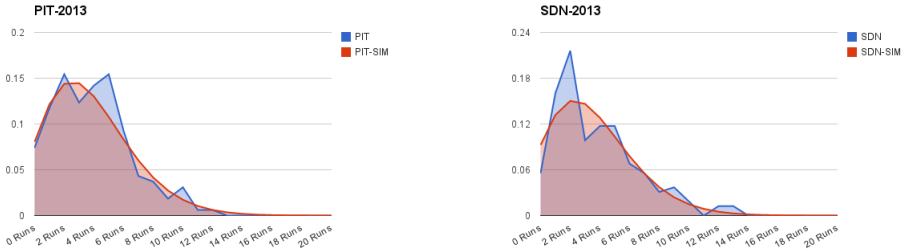

FIGURE 17. NL - San Fransisco and St. Louis Run Distributions. All run outcomes are discrete.

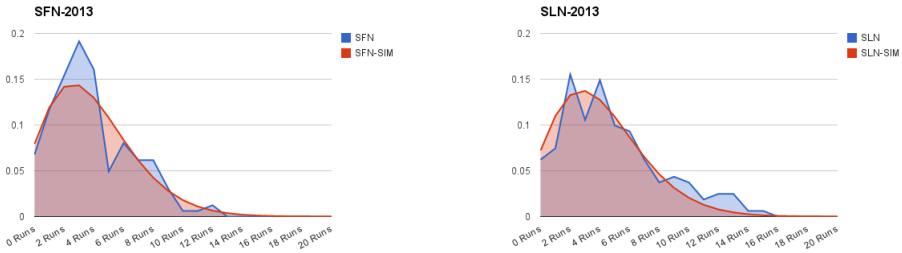

FIGURE 18. NL - 2013 Washington Run Distribution. All run outcomes are discrete.

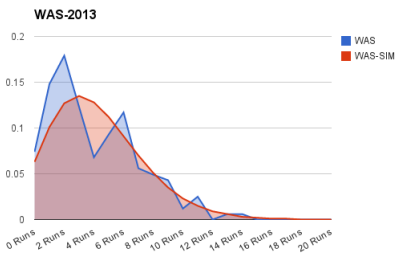

FIGURE 19. Full Transition Matrix

$$
\begin{array}{cccccccccccccccccccccccc}
p_{HR} & p_{1B}+p_{BB} & p_{2B} & p_{3B} & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B}+p_{BB} & 0 & p_{2B} & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & p_{BB} & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & p_{DP} & 0 & 0 & 0 & 0 & 0 & 0 & p_{TP} \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{TP} \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{TP} \\
p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & 0 & 0 & p_{BB} & 0 & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{TP} \\
p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{TP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B}+p_{BB} & p_{2B} & p_{3B} & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B}+p_{BB} & 0 & p_{2B} & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & p_{BB} & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & 0 & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & p_{Out} & 0 & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & p_{Out} & 0 & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & 0 & 0 & p_{BB} & 0 & 0 & 0 & 0 & 0 & 0 & p_{Out} & p_{DP} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & 0 & 0 & 0 & 0 & 0 & 0 & p_{Out} & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B}+p_{BB} & p_{2B} & p_{3B} & 0 & 0 & 0 & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B}+p_{BB} & 0 & p_{2B} & 0 & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & p_{BB} & 0 & 0 & 0 & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & p_{BB} & 0 & 0 & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & p_{1B} & p_{2B} & p_{3B} & 0 & 0 & 0 & p_{BB} & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{HR} & 0 & 0 & p_{3B} & p_{1B} & 0 & p_{2B} & p_{BB} & p_{Out} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

References

[1] Bellman, R., *Dynamic Programming and Markovian Decision Processes, with Application to Baseball*, Departments of Electrical Engineering, Mathematics, and Medicine UCLA, 1976

[2] Puterman, M.L. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc. 2005

[3] Bukiet, B., Harold, E., Palacios, J. *A Markov chain approach to baseball* Operations Research 45.1 (1997): 14-23

[4] Pankin, M., *Finding Better Batting Orders*, http://www.pankin.com/markov/btn1191.htm

[5] Sokol, J.S., *An Intuitive Markov Chain Lesson from Baseball*, INFORMS Transactions on Education 5:1(47-55) 2004

[6] Lahman, S., *Sean Lahman Baseball Database 2013*, http://www.seanlahman.com/baseball-archive/statistics

[7] D. A. D'Esopo and B. Lefkowitz, *The Distribution of Runs in the Game of Baseball*, SRI, Internal Report, 1960

[8] Ursin, D., *SabermetricMS*, http://www.sabermetricms.elasticbeanstalk.com, 2014