Journal of Heuristics **9** (2003), pp. 353-370

A Robust Heuristic for Batting Order Optimization Under Uncertainty

Joel S. Sokol

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205

ABSTRACT

Baseball teams are faced with a difficult scheduling problem every day: given a set of nine players, find the optimal sequence in which they should bat. Effective optimization can increase a team's win total by up to 3 wins per season, and 10% of all Major League teams missed the playoffs by 3 or less wins in 1998. Considering the recent $252 million contract given to one player, it is obvious that baseball is a serious business in which making the playoffs has large financial benefits. Using the insights gleaned from a Markov chain model of baseball, we propose a batting order optimization heuristic that performs 1,000 times faster than the previous best heuristic for this problem. Our algorithm generates batting orders that (i) are optimal or near-optimal, and (ii) remain robust under uncertainty in skill measurement.

Keywords: Markov chain, baseball batting order optimization

Introduction

In the baseball industry, teams are faced with a difficult scheduling problem every day: given a set of nine inputs (players), find the optimal sequence in which they should bat. The quality of a sequence is dependent on the interactions between the various batting skills of each player. The best known heuristic for this problem is sufficiently fast for optimizing over a single set of 9 players; however, it is prohibitively slow for comparing sets of players to not just determine the optimal batting order, but also answer the meta-question of which 9 players should play. In this paper, we propose a new, Markov-based batting order optimization heuristic that generates optimal or near-optimal solutions approximately 1,000 times faster than the previous best heuristic, allowing both questions to be answered in under 2 minutes. Moreover, we show that the solutions are empirically robust under reasonable uncertainty, an important property that had not been understood before.

Freeze (1974), Thorn and Palmer (1984), Pankin (1991), and Bukiet, Harold, and Palacios (1997) have previously studied ways of optimizing the baseball batting order. Freeze (1974), based on the results of a Monte Carlo simulation using league-average data for each batting order slot, proposed a batting order heuristic in which players were sequenced in descending order of batting skill. Thorn and Palmer (1984) concluded (also using league-average data and Monte Carlo simulation) that Freeze's heuristic generates batting orders that are slightly worse than the traditional batting order (in which the best hitters usually bat 3rd and 4th). However, neither Freeze nor Thorn and Palmer studied data for individual teams, which have much more variation in skill (and in combinations of skills) between the batters.

Because they worked with league-average data, the batting orders generated by Freeze, as well as Thorn and Palmer, are generally suboptimal (and often significantly so) when real players with significant variations in skill are considered. Pankin (1991) studied a limited set of batting orders (less than 0.5% of the total possible sequences) for individual teams, and performed regressions to determine which batting order slots were (on average) best-filled by batters with high or low values of certain traditional baseball statistics (batting average, home runs, etc.).

Bukiet, Harold, and Palacios (1997), working with real team data, were able to construct an algorithm that was more flexible. Their algorithm evaluates each of 987 batting orders (pre-selected according to ten set criteria) and chooses the one that scores the most runs per game. Thorn and Palmer (1984), James (1985), and others have proposed formulas that quantify the correlation between wins and runs (see Section 3.2), so selecting the highest-scoring batting order is equivalent to selecting the batting order that is expected to win the most games.

Bukiet, Harold, and Palacios' criteria-based algorithm is empirically near optimal. However, because it relies on evaluating 987 batting orders, it can be impractical for answering daily what-if questions faced by Major League managers, such as "if we use a slightly different set of players, how will it affect our expected run-scoring?" In this paper, we present an algorithm that gives similar results but evaluates just a single suggested batting order, and thus runs over 1,000 times faster, allowing for quick answers to these questions.

A second drawback of the criteria-based algorithm is that it relies on rules such as "the seventh best batter can bat either first or sixth through ninth". While these rules are useful for identifying an empirically good set of approximately 1,000 batting orders to test, they do not provide insight into *why* such batting orders are good. Why should the seventh best batter bat either first or sixth through ninth, but not in any of the other slots in the batting order? We provide answers to this type of question, by identifying and addressing an underlying problem – finding a useful way of measuring players' contributions to the run-scoring process.

Bukiet, Harold, and Palacios used D'Esopo and Lefkowitz's (1960) scoring index to rank players within each team. The scoring index estimates, for each player, the number of runs that would be scored by a batting order comprised of nine clones of that player. In other words, it measures not just the skill of the batter, but also the interaction between the skills of that player and others just like himself. A more important question for the batting order problem is how a player's skills interact with those of *other* players. There are many ways by which players can contribute offensively to their team, and we show that an appropriately-designed two-dimensional measure based on interactions with other players gives a much more complete view of player value and player utility.

All of the previous batting order heuristics assume that the probabilities inherent in player performance are known. We show that the batting orders generated by our heuristic are robust in the sense that they are generally near-optimal even if the true probabilities differ from the observed results.

The rest of this paper is organized as follows. In Section 1, we describe a Markov chain model of baseball, and derive the formulas that measure each player's interactions with others. Based on the intuition behind that metric, we propose a fast batting order heuristic in Section 2. In Section 3, we give computational results demonstrating that our heuristic is competitive with that of Bukiet, Harold, and Palacios while requiring much less computation time. In Section 4 we study the question of uncertainty in batting skills. We show that optimality is not a robust property under uncertainty, but near-optimality remains robust, so that our near-optimal heuristic will remain so even in the case where batting skill is not known exactly. In fact, we show that all batting orders, whether good or bad, exhibit some degree of robustness. Because this is obviously the case in real life, the robustness result is important for demonstrating the utility of batting order optimization heuristics.

1. A Process Model of Baseball

Baseball can be thought of as a production process. Teams produce runs by iteratively applying a sequence of inputs (batters) to a system. In each of nine innings, a team begins with no runners on base and zero outs. Batters successively change the state of the system by changing the positions of the runners on base and/or by being put out (or causing baserunners to be put out). In the course of these changes, runs may be scored until three outs are recorded in the inning.

There are many combinations of events in baseball that produce runs. With the exception of a home run (in which the batter causes himself to score), more than one batter is involved. Generally, one batter will reach base (becoming a baserunner), and a subsequent batter will drive the first batter home (so that the baserunner scores a run). Often, there are intermediate batters, some who might contribute by allowing the baserunner to advance from one base to another so that less effort is required to drive the runner home, and others who might hinder the run-scoring process by producing outs.

1.1. A Markov Chain Model

To model the run-scoring process, we use a Markov chain independently proposed by Bukiet, Harold, and Palacios (1997); Stern (1997); Pankin (1991); Thorn and Palmer (1984); Cook (1964); and Howard (1960). (Lindsey (1963) uses a different Markov chain model to predict win probabilities.) The states of the Markov chain we use correspond to the runners on base and the number of outs at any time during a half-inning of play. There are eight possibilities for the positions of runners on base (three bases, each of which might or might not be occupied), and in each case there are three possible numbers of outs (0, 1, or 2), for a total of 8 x 3 = 24 states. We denote states by (B,O), where B is the set of bases occupied and O is the number of outs. For example, (23,1) corresponds to "runners on second and third base, 1 out", and (0,0) corresponds to "no runners on base, 0 out". We also define a 25th state, "3 outs", for which the positions of runners on base do not matter because the inning is over and no more runs can be scored.

For this Markov chain model, the state transition probabilities p_{ij} are the chance that the current batter's plate appearance will change the state of the system from state i to state j. For example, if state i = (1,0) ("runner on first base, 0 out") and state j = (0,0) ("no runners on base, 0 out"), then the only ways to get from state i to state j in a single transition are for the batter to hit a home run, or for the defense to make one or more errors that allow both the runner and the batter to score. Thus, p_{ij} is equal to the probability of the batter hitting a home run, plus the probability of the defense making the necessary errors.

In addition to constructing a matrix P of transition probabilities, we can also construct a matrix T of transition values measured in terms of actual runs scored. For example, the transition from "runner on first base, nobody out" to "no runners on base, nobody out" involves two runs scoring, so this transition will have a value of 2. For any state transition, the batter and every runner on base in the initial state i will either score, be put out, or be on base in the new state j. Therefore,

$$T_{ij} = O_i + |B_i| + 1 - O_j - |B_j|, \qquad (1)$$

where $|B_k|$ denotes the number of baserunners in state k and O_k denotes the number of outs in state k. Transitions to the "3 outs" state will have a value of 0, except in the rare cases where a run scores and the third out is made on the same play. The run-scoring matrix T, unlike the transition matrix P, is the same for every batter.

1.2. Event Probabilities Considered

Every batter b will define a unique matrix P_b of transition probabilities. As Bukiet, Harold, and Palacios (1997) observed, the model is very flexible, in that a large amount of detail can be added to the transition matrix. For example, the probability of the defense making enough errors that the batter scores depends on who the fielders are. The probability of the batter hitting a home run is dependent on who the pitcher is. Factors such as the ballpark dimensions and elevation, wind speed and direction, and even who the home-plate umpire is can also affect the transition probabilities. Given sufficiently detailed data, we can construct a specialized transition matrix for every situation. However, since our objective is simply to sequence distinct batters, we need not go into such detail; we take into account each batter's characteristics and assume a league-average level of pitching, fielding, etc.

In this model, we implicitly assume that we can use statistics from previous baseball seasons to estimate a batter's transition matrix. Berry, Reese, and Larkey (1999) found the assumption that individuals' batting skill did not change much from year to year to be generally true, except for the very beginning and end of a batter's career. In Section 4, we will discuss the uncertainty in a player's transition matrix and show that our methodology is robust under reasonable amounts of uncertainty.

In determining transition probabilities, we take into account each batter's frequency of a number of positive offensive events (singles, doubles, triples, home runs, walks (plus times hit by pitches), and stolen bases) and negative events (strikeouts, flyball outs, groundball outs, times caught stealing (plus pickoffs), double plays, and sacrifice flies). We also take into account league average probabilities of events such as defensive errors, balks, wild pitches, and passed balls when creating transition matrices.

Omitted from the model are situational factors such as clutch hitting (the supposition that batters perform differently under pressure) and batting order protection (the hypothesis that a batter will be thrown easier pitches to hit if a good batter is next in the batting order). The existence of such factors would, in fact, invalidate the Markov model by violating the "memoryless" property. However, James (1985) and Grabiner (1993) have shown that clutch hitting is statistically insignificant leaguewide, and Cramer (1977) has shown it to be indistinguishable from random chance. Pankin (1993) and Grabiner (1997) have shown batting order protection to be similarly insignificant.

1.3. Derived Data

We created a transition matrix for the average batter using league total data. Given this league transition matrix P^L, we calculated the steady-state probabilities π (see Table 1) of the system using the Markov chain steady-state equations

$$\pi = P^L \pi, \tag{2}$$

$$\textstyle\sum_i \pi_i = 1. \tag{3}$$

For this calculation, we eliminate the "3 outs" state; incoming transitions to this state are instead directed back to state (0,0) for the start of a new inning.

The calculations make the implicit assumption that we can estimate the league's real steady-state parameters using an average transition matrix for non-identical batters. Although the true steady-state probabilities for the seasons we studied were unavailable, Thorn and Palmer (1984) showed that similar calculations for previous seasons have been good approximations for the actual values, as far back as Lindsey's (1959) study.

Baserunners	*0 Out*	*1 Out*	*2 Out*
0	.240	.172	.133
1	.056	.072	.074
2	.019	.030	.036
3	.002	.006	.013
12	.013	.024	.030
13	.006	.014	.020
23	.002	.006	.009
123	.003	.008	.011

Table 1. 1998 National League estimated steady-state probabilities π.

We can also calculate the expected run value v of each state (see Table 2); that is, the number of runs a team can expect to score from the time the system is in a state until the time the system reaches the three-out state (after which no more runs can be scored in the inning), again using the league averages. One way to think of these values is as the team's expected run potential for the rest of the inning.

To do this, we use the standard equations

$$v = r + P^L v, \tag{4}$$

$$v_{3out} = 0, \tag{5}$$

where $r_i = \sum_j P_{ij}^L T_{ij}$. Since every inning begins in state (0,0), $v_{(0,0)}$ should be an estimate of the number of runs scored per inning by the entire league. By multiplying the expected run value of state (0,0) by the number of innings played by the league, we can calculate the number of runs our model predicts should have been scored. Our model differed from the true value by less than 2%.

Baserunners	*0 Out*	*1 Out*	*2 Out*
0	.505	.270	.101
1	.898	.529	.223
2	1.105	.671	.309
3	1.349	.924	.356
12	1.520	.948	.453
13	1.787	1.182	.488
23	1.956	1.334	.571
123	2.431	1.650	.784

Table 2. 1998 National League estimated expected run values *v*.

1.4. Measuring Player Interactions

We can use the transition matrix P^L, the scoring matrix T, the steady-state probabilities π, and the expected run values v to calculate the effect u of various common baseball events on the run-scoring system. These effects of individual events are the basis for Thorn and Palmer's (1984) "linear weights" method of player evaluation. To evaluate a player or team, one can simply take the inner product of the player or team's probability of each event with the event values u to find the player or team's value, expressed in runs above average. Dividing by the player or team's number of plate appearances gives the value (compared to the average player) per plate appearance.

For every event e, let s_{ie} be the state of the system after event e occurs in state i. The value u_e of event e is then

$$u_e = \Sigma_i \,(T_{is_{ie}} + v_{s_{ie}} - v_i). \tag{6}$$

Rather than calculating the overall value of each baseball event, we find it much more insightful to split events into two basic components: potential value (*PV*) and realization value (*RV*). Potential value measures the expected change in run potential (see Table 2) as the result of transitions:

$$PV_e = \Sigma_i \,\pi_i(v_{s_{ie}} - v_i). \tag{7}$$

Realization value measures the expected number of runs that score as a result of transitions (in other words, the amount of potential that is realized):

$$RV_e = \Sigma_i \ \pi_i T_{is_{ie}} . \tag{8}$$

Every player's contribution to the run-scoring process can be divided into these two areas: realization value measures how much the player takes advantage of the situation created by previous batters, and potential value measures how well the player creates good situations for the following batters. Thus the two measures partition each player's ability into forward interactions and backward interactions within the batting order.

Event	*Realization value*	*Potential value*	*Total value*
Single	.22	.26	.47
Double	.46	.30	.76
Triple	.62	.39	1.02
Home Run	1.62	–.20	1.43
Walk	.02	.32	.35
Strikeout	.00	–.28	–.28
Flyball Out	.02	–.29	–.27
Groundball Out	.04	–.29	–.25
Double Play	.05	–.85	–.80
Stolen Base	.00	.14	.14
Caught Stealing	.00	–.38	–.38

Table 3. 1998 National League average event value breakdowns.

Based on this Markov chain model described, we next (Section 2) present a new heuristic for determining a good batting order. In Section 3, we will show that this heuristic produces optimal or near-optimal results much more quickly than previous near-optimal heuristics, and in Section 4 we demonstrate the robustness of the algorithm under uncertainty in transition probabilities.

2. A Heuristic for Optimizing the Batting Order

The observation that player abilities can be partitioned into forward and backward interactions yields insight into the strategy of creating a good batting order. In general, in order to maximize the value of their forward interactions, players who have high potential values should be placed ahead of players who have good realization values. Similarly, to maximize the value of their backward interactions, players who have high realization values should be placed after players who have good potential values. A player who excels in both categories should be placed after players with high potential values and before players with high realization values. In this way, all of these players' contributions to the team's run-scoring process will be maximized.

The process-based analysis provides a similar intuition for the placement of batters with low realization value and/or potential value. For example, after batters with high realization values and low potential values, the manager should place a batter who has a low realization value. This is because the high-realization, low-potential batters will tend to remove baserunners without replacing them, thereby minimizing effect of the next batter's low realization value.

We now formalize these notions into an algorithm that will generate an optimal or near-optimal batting order. To begin, we divide each team into four subsets:

- Set (R^+,P^+): All-around contributors – players with above-average realization value and above-average potential value,
- Set (R^+,P^-): Run producers – players with above-average realization value and below-average potential value,
- Set (R^-,P^+): Table setters – players with below-average realization value and above-average potential value, and
- Set (R^-,P^-): Weak hitters – players with below-average realization value and below-average potential value.

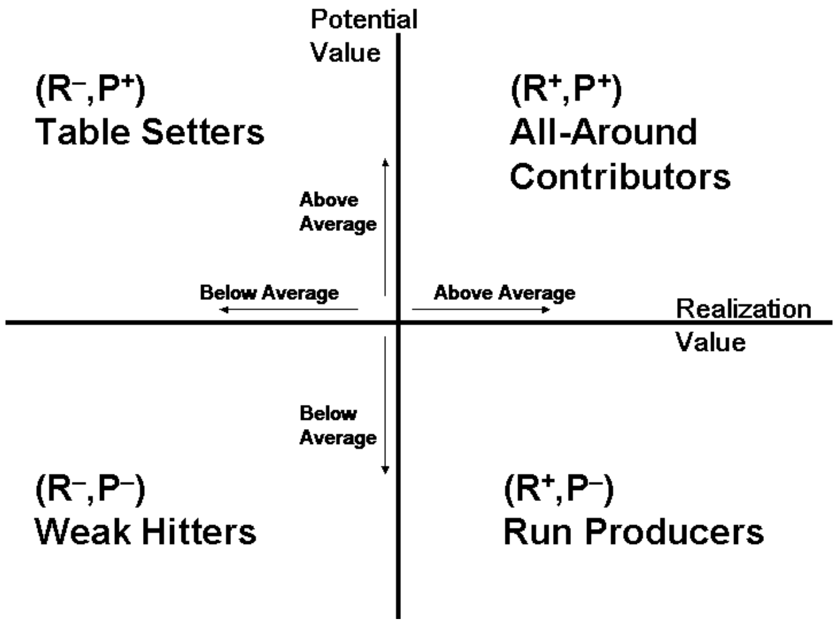

Figure 1. Partition of batters into subsets.

In creating an ordering of these four sets, we would like every set with an above-average realization value to immediately follow a set with an above-average potential value, and every set with an above-average potential value to immediately precede a set with an above-average realization value. The only way to do this is for set (R^-,P^+) to be followed by set (R^+,P^+), then set (R^+,P^-), and finally set (R^-,P^-). Since a batting order is circular, there are four possible orderings that satisfy this restriction. If the team were to bat in a large number of innings, then the starting point would eventually be evenly distributed among the nine batters and the number of runs scored per inning would be the same. However, in baseball every game begins with the first batter, and the number of innings (9) is relatively small. Lindsey (1961) has shown that after nine innings, the distribution is still not steady-state. Bukiet, Harold, and Palacios (1997) hypothesize that the lack of steady state is due to the changing of pitchers and batters in the latter innings of the game. Their model predicts that steady state will be reached before the end of a game. Both studies, however, predict that the deviation from steady-state will decrease as the game progresses. Therefore, it is best to front-load the batting order so that more runs will be scored early in the game before steady state is reached. We order the sets as follows:

1. Table-Setters (R^-,P^+),
2. All-Around Contributors (R^+,P^+),
3. Run Producers (R^+,P^-),
4. Weak Hitters (R^-,P^-).

Not all players fit nicely into one of these sets. Often, a player will be very close to average in one or both categories. To assign these players to sets, we take into account the assignments of other players on the team. If less than half of the batters are in the above-average realization value sets, then we round the best borderline realization value batter to above average. (Since half of 9 is 4.5, we round up to 5.) We repeat the process until either no more borderline realization value batters are unassigned or until we have 5 batters assigned to above-average realization value sets. Any remaining borderline realization value batters are rounded to below-average realization value sets. We perform a similar operation with potential value, using 4 above-average batters as our cutoff. The reason for the difference in cutoff levels is to attempt to maintain an equal number of above and below average assignments. We choose to use the higher cutoff level for realization value instead of for potential value because there is a greater variation among players in realization value than in potential value. The threshold for determining who is close to average can be adjusted; we found that defining it to include all batters within 5% of average works well.

There is no guarantee that all four of the sets will be nonempty. If either or both of the sets (R^+,P^+) or (R^-,P^-) is empty, our batting order strategy still holds; every above-average potential value group is still followed by an above-average realization value group, and every above-average realization value group is preceded by an above-average potential value group. In either case, we can proceed without any modifications.

If the table setter group is empty, we form one by reassigning a batter from either the (R^+,P^+) group or the (R^-,P^-) group. From among all batters in those groups, we pick whichever batter is most potential-oriented; that is, we take the batter i with the maximum value of $(PV_i - PV_{LG}) - (RV_i - RV_{LG})$, where PV_{LG} and RV_{LG} are the league average potential and realization values. On the other hand, if the run producer group is empty, we create one by reassigning the most realization-oriented batter, whoever has the maximum value of $(RV_i - RV_{LG}) - (PV_i - PV_{LG})$. Note that these two calculations will select the same batter as the one that maximizes $(PV - RV)$ or $(RV - PV)$.

Once the groups are defined, we determine the batting order within each group. Ideally, we would like to reach a potential value "peak" when the first of the R^+ batters comes to bat, and a potential value "valley" after the last of the R^+ batters has batted. Therefore, the batters in each group between the valley and the peak ((R^-,P^-) and (R^-,P^+)) should be ordered by increasing potential value, and the batters in each group between the peak and the valley ((R^+,P^+) and (R^+,P^-)) should be ordered by decreasing potential value. Of course, these are all within groups; all of the (R^+,P^+) batters should bat before any of the (R^+,P^-) batters, for example.

We apply three logical exceptions to these ordering rules. First, on occasion a team is overall not high in realization value but has a batter in the (R^-,P^-) group with an realization value that is not far below the average (but is too far below to have been a borderline case – in our model we use players within 10% of average). In such a case, that batter should hit first in the (R^-,P^-) group to finish taking advantage of the built-up run potential.

A second exception exploits the circularity of the batting order in light of the original objectives. The goal of front-loading the order for the first inning may not be met if the (R^-,P^+) group is too large. To guarantee that at least one of the (R^+,P^+) batters bats in the first inning, if the (R^-,P^+) group is larger than two batters, the first batter in the group should be shifted to 9th in the batting order, and everyone else moved up one slot. This maintains the same circular batting order, but shifts the starting point. For National League lineups in which the heuristic initially chooses the pitcher to bat 7th, we ignore this condition so that a poor-hitting pitcher does not bat as high as 6th in the batting order.

The third exception is similar to the second in attempting to get the best set of batters to start the game. If the table setter group has more than one batter and the first batter is significantly worse (a 20% or more difference in potential value) than the second batter, then it is often worthwhile to shift the first batter to 9th again, so long as the all-around contributor group has at least 4 members. This will increase the likelihood of the best above-average potential value batters batting in the first inning. Again, every other batter will move up one slot.

We note that, instead of having all batters in each group appear consecutively in the order, an alternative strategy might be to split each group to create two cycles of table-setter, all-around contributor, run-producer, and weak hitter (with the one duplicate, for a total of 9 batters). Our computational results (see Section 3.3) suggest that this is a bad idea, and we hypothesize that the reason is due to the frequency of outs. With rare exceptions, even the best batters make an out 55% of the time (or more). Therefore, it is helpful to have two or more good potential-value batters before an all-around contributor, so the chance of him batting with one or more runners on base is higher. The same is true of all-around contributors batting before run producers.

2.1. Solving a "Special Case"

We illustrate our algorithm by analyzing one team (1989 San Francisco; see Table 4) that seemed to be a special case for Bukiet, Harold, and Palacios (1997), but for which our algorithm finds the optimal solution in a straightforward manner. Bukiet, Harold, and Palacios suggested ten criteria for batting order generation, based on a ranking of the team's batters using D'Esopo and Lefkowitz's (1960) scoring index. The criteria defined 987 batting orders, and their algorithm enumerates and simulates each one and chooses the best among them. Bukiet, Harold, and Palacios selected their ten criteria by observing the optimal batting orders of the twelve National League teams of 1989. The criteria include the following two statements: "the fifth best batter should come up first or second, or fifth through seventh", and "the seventh best batter can bat either first or sixth through ninth". In fact, the fifth best batter (as ranked by scoring index) hit as low as seventh only once, and the seventh best batter hit first only once; both singular events occurred in the optimal San Francisco batting order.

Player	*Scoring Index*	*Realization value*	*Potential value*
Mitchell	.941	.180	–.071
Clark	.901	.138	–.050
Butler	.462	.084	–.073
Thompson	.425	.100	–.095
Williams	.411	.138	–.140
Maldonado	.369	.089	–.098
Kennedy	.349	.080	–.092
Uribe	.238	.066	–.107
Pitcher (Average)	.112	.054	–.149
League Average (1989)	N/A	.092	–.092

Table 4. San Francisco batters (1989).

The initial group assignments are (R^-,P^+) = {Butler}, (R^+,P^+) = {Mitchell, Clark}, (R^+,P^-) = {Williams}, and (R^-,P^-) = {Uribe, Pitcher}. Thompson (near-average potential value), Maldonado (near-average realization value), and Kennedy (near-average potential value) are initially unassigned.

Because there are only 4 batters in the above-average realization value groups, Maldonado is rounded up to the (R^+,P^-) group. There are only 3 batters in the above-average potential value groups, so Kennedy (with a better potential value than Thompson) is rounded up to the (R^-,P^+) group. Now there are 4 batters in the above-average potential value groups, so Thompson is rounded down to the (R^+,P^-) group.

Within the (R^-,P^+) group, the batters are ordered by increasing potential value; thus, Kennedy will bat first and Butler will bat second. Within the (R^+,P^+) group, the batters are ordered by decreasing potential value; thus, Clark will bat third and Mitchell will bat fourth. Within the (R^+,P^-) group, the batters are ordered by decreasing potential value; thus, Thompson will bat fifth, Maldonado will bat sixth, and Williams will bat seventh. Within the (R^-,P^-) group, the batters are ordered by increasing potential value; thus, the pitcher will hit eighth and Uribe will hit ninth. None of the post-ordering conditions are satisfied, so the final batting order is set: Kennedy, Butler, Clark, Mitchell, Thompson, Maldonado, Williams, Pitcher, Uribe.

By enumerating all 9! = 362,880 possible batting orders, Bukiet, Harold, and Palacios (1997) showed that this heuristic order is the optimal batting order. The intuition we have developed using the realization value and potential value analysis has led directly to this optimal batting order, and demonstrated that what seemed to be a special case is actually a straightforward application of our process-based methodology.

3. Computational Testing

3.1. Test Data

To test the speed and quality of our algorithm, we used 1998 and 1999 Major League data, as well as the same 1989 National League data used by Bukiet, Harold, and Palacios. The data was taken from Neft and Cohen (1990) and ESPN (1998, 1999).

The first step in collecting data was to determine which set of 9 players to consider for each team. Due to injuries, trades, and strategic issues, teams do not have the same nine players in the batting order for each game. In determining which 9 players would be used for our testing, we used Bukiet, Harold, and Palacios' (1997) convention of selecting the player who played most often at each position. American League teams can use the same 9 batters in every game, because they use a designated hitter who bats for the pitcher. (Pitchers must rotate – generally in a cycle of 4 or 5 games – because of fatigue.) For National League teams, we selected only 8 players, and used the totals of all pitchers as the 9th player.

Once the players were selected, we collected data on the frequency of events (home runs, strikeouts, etc.) for each player. Most of the data was readily available. Ratios of flyball outs to groundball outs for 1989 individuals and teams, as well as for 1998 and 1999 teams, were unavailable and had to be estimated based on league averages. Breakdowns of stolen bases and times caught stealing by base were also unavailable, and so we assume that all steals were of second base (by far the most common) and ignored the rarer steals of third base and home plate. Very rare events, such as catcher interference with a batter's swing, baserunners struck by batted balls, and triple plays, were omitted from the model.

For some events with more than one possible outcome, our model requires the frequency of each potential outcome. For example, when a batter hits a single with a baserunner on first base, we need to model the probability that the baserunner will (1) stop at second base, (2) advance to third base, or (3) be thrown out attempting to advance to third base. This data was estimated based on the work of Pankin (1993).

In addition to a model based on these estimates of baserunner advancement, we also use the simplified model of D'Esopo and Lefkowitz (1960) and of Bukiet, Harold, and Palacios (1997) when comparing our results with theirs. Although they find that their model underestimates the number of runs scored by approximately 7%, it gives us a way to compare our batting order generation method with theirs. The simplified model was used only for comparisons with Bukiet, Harold, and Palacios' computations with 1989 data.

In addition to comparing our heuristic with that of Bukiet, Harold, and Palacios, we benchmarked our batting orders against those used by Major League teams. If teams currently use near-optimal batting orders, then they would not benefit much from optimization.

Since teams do not use the same players in each game, they also do not use the same batting order for each game. Over the course of a full season, most players bat in more than one slot in the batting order. Given the set of the nine players who played most often at each position for a specific team, we estimate a team's standard batting order by solving the following assignment problem. Let l_{ij} be the fraction of plate appearances in which player i batted in slot j, and x_{ij} be a zero/one variable that will be equal to one if and only if player i is in slot j in the standard batting order. Then, we solved the assignment problem

$$\text{Maximize} \quad \sum_{i,j} l_{ij}\, x_{ij}, \tag{9}$$

$$\text{Subject to} \quad \sum_{j} x_{ij} = 1, \quad \text{for all players } i = 1,\ldots,9, \tag{10}$$

$$\sum_{i} x_{ij} = 1, \quad \text{for all batting order slots } j = 1,\ldots,9, \tag{11}$$

$$x_{ij} \geq 0, \quad \text{for all } i,j \in \{1,\ldots,9\} \times \{1,\ldots,9\} \tag{12}$$

to determine the team's standard batting order. In most cases, the solution to the assignment problem was obvious; each player had most of his plate appearances in the same batting order slot, and that player was the most frequent on the team to bat in that slot.

The batting order slot frequencies l_{ij} were not available for the 1989 data, so comparisons involving standard batting orders were made only for the 1998 and 1999 data.

3.2 Performance Metrics

Because lineup quality is measured in runs and team performance is measured in the number of games won, we need a method of converting runs to wins.

Let *WP*, *RS*, and *RA* denote a team's winning percentage (wins divided by games played), the number of runs scored by the team, and the number of runs allowed by the team. James (1985) has shown that a team's winning percentage is well-approximated by $WP \approx (RS)^2 / [\ (RS)^2 + (RA)^2\ ]$, a formula he calls the Pythagorean method. For example, St. Louis scored 810 runs and allowed 782 runs in 1998. Their estimated number of wins in a 162 game season is equal to 162 times their approximate winning percentage, or $162 \times (810)^2 / [\ (810)^2 + (782)^2\ ] = 83.85$. In fact, St. Louis won 83 out of 162 games in 1998.

Because *RA* is known for every team (and is independent of batting order), we can use James' (1985) formula to convert runs to wins in our computational tests. Notice that *WP* is a strictly increasing function of *RS*, so (logically) more runs scored leads to more wins.

3.3. Results

Overall, our algorithm generated the optimal batting order for 6 of the 12 teams in the 1989 National League. Moreover, for the other 6 teams, the heuristic batting order was no more than 1.1 runs per season (162 games) worse than the optimal batting order. The heuristic order was approximately one third of a run per season from optimal on average. This performance is approximately the same as that of Bukiet, Harold, and Palacios' (1997) algorithm on data which was not used in generating their criteria. According to the Pythagorean formula, it is unlikely that one or two additional runs per season will affect the number of games that a team will win; therefore, in terms of games won, the batting orders generated by our heuristic, as well as Bukiet, Harold, and Palacios' heuristic, are as good as the optimal batting orders.

For 1998 and 1999 data, our algorithm's performance was nearly as good (see Table 5). The three run differential columns report the average run differentials between (i) the optimal and worst batting orders, (ii) the optimal and heuristic batting orders, and (iii) the heuristic and standard (as defined in Section 3.1) batting order. The league-average difference between the optimal and worst batting orders was between 34 and 55 runs per season, while the league-average difference between the optimal and heuristic batting orders was less than a single run. More importantly, the league-average difference between the heuristic and standard batting orders was 13-15 runs.

We used the Pythagorean method to convert the run differentials to an estimate of the effects on team won/lost records (see Table 6). We report the maximum, average, and minimum differences for each season. The heuristic lineups, even in the worst cases, differed from the optimal lineups by less than ½ of a win per season. Of equal importance was the observation that the heuristic lineups beat the standard lineups by an average of almost 1.5 wins per season; for some teams the difference was in the 2.5 to 3 win range.

	Average Run Differentials		
League	*Optimal – Worst*	*Optimal – Heuristic*	*Heuristic – Standard*
1989 National	40	0.3	N/A
1998 National	55	0.5	14
1999 National	53	0.4	13
1998 American	34	0.4	14
1999 American	35	0.6	15

Table 5. Run differentials for Major League data.

	Average Win Differentials (Pythagorean Estimate)								
League	*Optimal – Worst*			*Optimal – Heuristic*			*Heuristic – Standard*		
	Max	*Avg.*	*Min*	*Max*	*Avg.*	*Min*	*Max*	*Avg.*	*Min*
1989 National	6.5	4.8	3.8	0.3	0.1	0.0	N/A	N/A	N/A
1998 National	6.3	4.9	3.7	0.4	0.1	0.0	2.8	1.4	0.5
1999 National	6.3	4.9	3.5	0.2	0.1	0.0	2.1	1.3	0.5
1998 American	4.6	3.4	2.3	0.3	0.1	0.0	2.4	1.4	0.3
1999 American	4.5	3.2	2.2	0.4	0.1	0.0	2.5	1.4	0.2

Table 6. Win differentials (Pythagorean estimate) for Major League data.

The running time of our algorithm is much faster than either Bukiet, Harold, and Palacios' (algorithm or the exact algorithm of complete enumeration. The complete enumeration algorithm evaluates all 9! = 362,880 batting orders. Bukiet, Harold, and Palacios' algorithm narrows the selections down to 987 possible batting orders, and evaluates all of those. Our algorithm generates only a single batting order, and therefore does not even need to evaluate even one batting order except for the purposes of what-if comparisons. Bukiet, Harold, and Palacios (1997) report computation times of approximately 5.5 days of CPU time for full enumeration, 20 minutes for the 987-order algorithm, and 1.3 seconds for a single batting order evaluation (the equivalent running time of our algorithm), all on a Sparc 2 workstation. Using a C code written for a 400 MHz Pentium II PC with a Linux operating system, we achieved computation times of approximately 2 hours for full enumeration, 20 seconds for the 987-order algorithm, and 0.02 seconds for our algorithm to select an order and perform a single batting order evaluation. Regardless of the platform used, our algorithm is approximately 1,000 times faster than the criteria-based algorithm.

Because our algorithm runs so quickly, it would be useful not only for suggesting optimal batting orders on a day-to-day basis, but also as an analysis tool for selecting which set of nine players a team should use in each game. Many teams have players of nearly-equivalent value at the same position, but very different skills. Against a certain pitcher, would a team be better off using a left fielder with a higher potential value or a higher realization value? Questions like these could easily be answered; the batting order optimization heuristic can evaluate both possibilities, and calculate the expected difference in run-scoring, in well under one second. In fact, because Major

League teams generally have no more than 15 position players on their active roster, there are at most 15!/(9!6!) = 5005 sets of nine players; with a running time of 0.02 seconds per optimization, our algorithm could evaluate all possibilities in less than 2 minutes (see Table 7). For National League teams, there would be at most 15!/(8!7!) = 6435 sets, with a running time of approximately 2 minutes.

Task	*Approximate Running Times (Seconds, 400 MHz PC)*		
	Complete Enumeration	*Criteria-Based Heuristic*	*Markov-Based Heuristic*
Optimize lineup for a single set of 9 given players	7200	20	0.02
Optimize lineup for 6 given players, 3 either/or choices (8 possibilities)	57,600	160	0.16
Optimize over all 5005 combinations of 15 players (American League)	36,000,000	100,000	100
Optimize over all 6435 combinations of 15 players (National League)	46,700,000	130,000	130

Table 7. Approximate running times of the algorithms.

We note that, when selecting a set of players, *RA* is not independent of the player set. Although much research has been done on evaluating the offensive contributions of players (see Cover and Keilers (1977), Cramer (1980), Pankin (1978), Thorn and Palmer (1984), James (1984), and many others), comparatively little has been done on evaluating defense (see, for example, Thorn and Palmer (1984) and James (2002)). However, because defense is independent of batting order, any defensive rating system can be used to calculate the difference to *RA* of a combination of players – this will need to be done at most 5005 or 6435 times. For the current best models of evaluating defense, all these calculations would add a negligible amount to the overall running time.

4. Batting Order Robustness Under Uncertainty

Thus far, we have made the assumption that the transition probabilities for each batter are known exactly. Of course, this is not the case in real life. While our heuristically-generated batting orders are optimal or near-optimal under the assumption that event probabilities are equal to observed data (seasonal statistics), for our optimization heuristic to be valuable the heuristically-generated batting orders must be *robust*. That is, even if the data is not a true representation of the probabilities, the heuristic recommendation should be near-optimal under the real probabilities. No previous batting order study we are aware of has addressed this issue.

To test robustness, we selected a single team and generated 200 scenarios using a bootstrapping method similar to that described by Efron and Tibshirani (1993). We used each player's observed data as a probability distribution from which we drew the 200 scenarios. For example, a player who hit 20 home runs in 400 real-life plate appearances would have a 20/400 probability of hitting a home run in each draw. In a given scenario, if we happened to draw 23 home runs in 400 trials for the player, then the player's home run probability in that scenario would be 23/400. To make robustness more difficult to achieve, we selected a test team (1998 St. Louis) which, due to injuries and trades, had a significant number of players with many fewer plate appearances than average. The smaller sample size for those players created a wider variance in the scenarios.

For each of the 200 scenarios, we evaluated every one of the 362,880 possible batting orders. We found that the property of optimality, being the batting order with the highest expected number of runs scored, was not robust at all. No batting order was optimal (or tied for optimal) in more than 14 of the 200 scenarios. Furthermore, in the 200 scenarios, more than 100 batting orders were optimal at least once. Therefore, we conclude that it makes no sense to search for an "optimal" batting order, because the transition probabilities have enough uncertainty that no matter which order we choose, it most likely will not be optimal for the true probabilities.

However, our tests showed that the property of *near-optimality* is robust. For each of the 200 scenarios, we calculated the difference in expected runs scored between each batting order and the optimal batting order for that scenario. The batting order that performed best was the one we had determined to be optimal when observed data was used as the probability distribution; on average, it was just 3.4 runs lower than each scenario's optimum, with a standard deviation of 2.9 runs below optimality. Moreover, the batting order suggested by our heuristic (which was 0.3 runs worse than optimal for observed data) was, on average, only 3.7 runs per season worse than each scenario's optimum, with a standard deviation of 2.6 runs below optimality per scenario. In other words, even though the optimal batting orders for each of the scenarios were widely diverse, the batting order that was optimal for the real (deterministic) data and the batting order suggested by our heuristic (again using real data) were both consistently near-optimal in the 200 scenarios.

In fact, the performance of every one of the 362,880 batting orders was at least somewhat robust under uncertainty in the 1998 St. Louis data. Although some batting orders were, on average, as far as 63.3 runs per season from optimal, the largest standard deviation was just 8.6 runs below optimality per scenario, or less than one expected win. In other words, good batting orders tended to be good for all 200 scenarios, and bad batting orders tended to be bad for all 200 scenarios.

This observation supports the utility of our heuristic. Not only was the heuristic batting order near-optimal in all 200 of the scenarios, but in fact searching for a batting order that is any better than near-optimal is probably fruitless since the optimal batting order is so sensitive to uncertainty.

5. Conclusion

Starting with an oft-proposed Markov chain model, we have considered baseball as a run-scoring process. We have demonstrated how to characterize players by their forward and backward interactions during the run-scoring process. In doing so, we quantified the traditional baseball concepts of "table setters" and "run producers" using two measures, potential value and realization value.

Using the insight gained from the process model, we proposed an intuitive heuristic for generating batting orders. We split batters into four categories (Table Setters, All-Around Contributors, Run Producers, Weak Hitters) based on their expected contribution to the team's run-scoring potential and to converting potential runs into actual runs. Within each group, we order the batters by potential value, in increasing order when the group's purpose is to build potential and in decreasing order when the group's purpose is to convert potential to actual runs. The result is an algorithm that performs as well as the previous best heuristic algorithm for the batting order problem, but runs approximately 1,000 times faster, allowing it to be used to answer what-if questions regarding issues such as player selection. Moreover, it provides an explanation for some of the findings of Bukiet, Harold, and Palacios (1997), and showed that what appeared to be a special case for their algorithm is actually a very straightforward instance of the problem.

Our batting order heuristic (or any batting order heuristic, for that matter) is valuable only if it can be used successfully in real life. Therefore, it must perform well under uncertainty. We used a bootstrapping technique to create 200 simulated seasons from one season of real data, and found that the heuristic batting order remains near-optimal and is robust under uncertainty in transition probabilities. The batting order suggested by our heuristic when the probabilities are assumed to be known (and equal to real-season data) is likely to also be optimal or near-optimal for any set of true but unknown transition probabilities.

Using James' Pythagorean method of relating a team's run-scoring to the number of games it wins, we estimated that the difference between optimized batting orders and the batting orders currently used by Major League teams was as high as three wins per season. Considering that 10% of all Major League teams missed the playoffs by 3 games or less in 1998, batting order optimization could make the difference between a team having a successful season and a disappointing season.

References

Berry, S., C. S. Reese, and P. D. Larkey (1999). A Sports Time Machine. INFORMS Cincinnati, May 1999.

Bukiet, B., E. R. Harold, and J. Palacios (1997). "A Markov Chain Approach to Baseball," *Operations Research* **45**, 14-23.

Cook, E. (1964). *Percentage Baseball*. MIT Press, Cambridge, MA.

Cover, T. M., and C. W. Keilers (1977). "An Offensive Earned-Run Average for Baseball," *Operations Research* **25**, 729-740.

Cramer, R. D. (1977). "Do Clutch Hitters Exist?" *Baseball Research Journal* **6**, pp. 74-79.

Cramer, R. D. (1980). "Average Batting Skill Through Major League History," *Baseball Research Journal* **9**, pp. 167-172.

D'Esopo, D. A., and B. Lefkowitz (1960). The Distribution of Runs in the Game of Baseball. SRI Internal Report.

Efron, B., and R. J. Tibshirani (1993). *An Introduction to the Bootstrap*. Chapman and Hall, New York.

ESPN (1998, 1999). ESPN Major League Baseball web site. sports.espn.go.com/mlb

Freeze, R. A. (1974). "An Analysis of Baseball Batting Order by Monte Carlo Simulation," *Operations Research* **22**, 728-735.

Grabiner, D. (1993). Does Clutch Hitting Exist? Unpublished.

Grabiner, D. (1997). Can You Protect a Hitter? Unpublished.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.

James, B. (1984). *The Bill James Baseball Abstract 1984*. Ballantyne Books, New York.

James, B. (1985). *The Bill James Historical Baseball Abstract*. Villard Books, New York.

James, B. (2002). *Win Shares*. STATS, Inc.

Lindsey, G. R. (1959). "Statistical Data Useful for the Operation of a Baseball Team," *Operations Research* **7**, 197-207.

Lindsey, G. R. (1961). "The Progress of the Score During a Baseball Game," *American Statistical Association Journal* **56**, 703-728.

Lindsey, G. R. (1963). "An Investigation of Strategies in Baseball," *Operations Research* **11**, 477-501.

Neft, D. S., and R. M. Cohen (1990). *The Sports Encyclopedia: Baseball.* St. Martin's Press, New York.

Pankin, M. D. (1978). "Evaluating Offensive Performance in Baseball," *Operations Research* **26**, 610-619.

Pankin, M. D. (1991). Finding Better Batting Orders. SABR XXI, New York.

Pankin, M. D. (1993). Subtle Aspects of the Game. SABR XXIII, San Diego.

Stern, H. S. (1997). "A Statistician Reads the Sports Page: Baseball by the Numbers," *Chance* **10**, p. 38.

Thorn, J., and P. Palmer (1984). *The Hidden Game of Baseball: A Revolutionary Approach to Baseball and Its Statistics.* Doubleday, Garden City, New York.