# Batting Order Optimization by Genetic Algorithm

Sen Han
525 13th St. S. Apt. 302,
St. Cloud, MN 56301
001-320-339-7110
hase0701@stcloudstate.edu

## ABSTRACT
Baseball has been widely studied in various ways, including math and statistics. In a baseball game, an optimized batting order helps the team achieves greater number of runs in a season. This paper introduces a method that combines a genetic algorithm with a statistical simulation to identify a non-optimal batting order. The biggest issue is how we evaluate a batting order. There are past works using dynamic programming to calculate the plate appearance and using Markov Chain to evaluate a batting order. These two algorithms summarize all past data to deliver an optimal batting order. The GA described here applies an evaluation function using a baseball game simulation. Thus the GA is more like a helping tool that can be incorporated into the decision making process rather than a deterministic tool. The simulation defines the baseball game as a set of events. By using only a subset of the event set, the decision maker can pursue a customized batting order.

## Categories and Subject Descriptors
J. Computer Applications. J.6 [COMPUTER-AIDED ENGINEERING] Subjects: Computer-aided design (CAD)

## Keywords
Genetic Algorithm, Batting Order, Simulation, Baseball Game

## 1. INTRODUCTION
In a baseball game, the offensive team has nine players bat in an order that is specified in advance. This order cannot be changed. The official baseball rules say the following: "The first batter in each inning after the first inning shall be the player whose name follows that of the last player who legally completed his time at bat in the preceding inning. " [3].

In what order should the lineup bat? Various methods have been proposed to solve this problem. Palacios [1] used Markov chains to model the baseball games and find the best batting order. The Markov chain modeled the game a sequence of states and defined transitions between states. Freeze [2] simulated the game by Monte Carlo Simulation. Batting orders are permutations of a set of batters. It is the nature of EA to attack this sort of problem. And it is natural to use the order of nine symbols to represent a batting order.


*GECCO'12 Companion,* July 7–11, 2012, Philadelphia, PA, USA.


Section 2, introduces the baseball simulation program used to evaluate each batting order. Section 3 presents the design of the EC. Finally, a discussion of reliability of the system is given on section 4.

## 2. BATTING ORDER EVALUATION
The evaluation function is the core of a Genetic Algorithm. Each chromosome represents a unique batting order. The evaluation of a batting order can be accomplished in various ways. The optimal batting order is the one that leads to the max number runs in a game. Several methods, to predict the number of runs, have been proposed. In Chen [5], the number of runs achieved is based on dynamic programming. The mechanism of the method can be presented by the following.

$$PA(bat, out) = \begin{cases} 1, \text{ if } bat = 1 \text{ and } out = 0 \\ 0, \text{ if } out \geq 27 \text{ or } out < 0 \\ PA(bat-1, out-1)*(1-OBP(bat-1)) + PA(bat-1,out) \\ *OBP(bat - 1), \text{ else} \end{cases}$$

$$TBA(batters, outs) = \sum PA(bat, out), \text{ where batter} \equiv bat \pmod 9 \text{ and outs} \equiv out \pmod 3$$

Run(batter, outs) =

BB(batter) * Runner(batter, 1,z outs, 0) +
1B(batter) * Runner(batter, 1, outs, 0) +
2B(batter) * Runner(batter, 2, outs, 0) +
3B(batter) * Runner(batter, 3, outs 0) +
HR(batter)

Runner(batter, base, outs, emptybases) =

$$\begin{cases} 0, & \text{if outs} >= 3 \\ 1, & \text{if outs} < 3 \text{ and base} >= 4 \\ Out(nextbatter)*Runner(nextbatter,base,outs+1,emptybases)+BB(nextbatter)*Runner(nextbatter,base,outs, emptybases1)+1B(nextbatter)*Runner(nextbatter,base+1,outs, emptybases)+2B(nextbatter)*Runner(nextbatter, base+2,outs,emptybases)+HR(nextbatter) & \text{if emptybases} > 0 \\ Out(nextbatter)*Runner(nextbatter,base,outs+1,emptybases)+BB(nextbatter)*Runner(nextbatter,base+1,outs,emptybases)+1B(nextbatter)*Runner(nextbatter,base+1,outs,emptybases)+2B(nextbatter)*Runner(nextbatter,base+2,outs,emptybases)+HR(nextbatter) & \text{if emptybases} = 0 \end{cases}$$

Palacios used a Markov chain was used to model the game [1]. To understand this conceptually, a game can be thought as a sequence of transitions related with each player's plate appearance.

> There are eight possibilities for the distribution of runners on base. There either is or isn't a runner on each of first, second, and third bases (23 = 8). Furthermore, there are zero, one, or two outs (3 x 8 = 24). The half inning ends with the third out, which we treat as a twenty-fifth state.

Both of these methods are based on statistical accumulated data. The accuracy of the program depends on these data. Although the number of runs achieved in a game is an important measure of a batting order, we can develop the GA to be more powerful and practical by extending the evaluation itself. Primarily, *the program should be more like a helping tool, which can be incorporated into the decision making process before the game, rather than a deterministic decision maker*. We want the program to produce informational results for different real world cases rather than just summarize the past data and deliver a single batting order and claim this is the "best" one. Therefore, we can summarize our expectations of the GA as following:

(1) Getting More Runs.
(2) Customized Batting Order Searching. The GA should be able to be directed into various directions. The coach may want to see batting orders that are best for certain events or the batting order in which an event does not happen.
(3) Statistically Oriented. All searching should be based on past accumulated data.
(4) Consider Season Performance. In the evaluation, the GA should consider the number of runs over the season. Indeed, we should separately consider the batting order that is more likely to perform well in a game or in a whole season.
(5) Return a set of solutions. The GA should deliver a set of solutions that can let the coach further consider.

In this GA, we use a game simulation to be the evaluation function [4]. The evaluation function simulates a season of games and compares batting orders by both the *total number of runs achieved* and *runs distribution in the season*. In the simulation, 3 outs is an inning; 9 innings is a game; 120 games is a season.

## 2.1 Events Table

In the simulation, a game is defined by a finite set of events which form an events table (E). The events table matrix contains 4 columns and a variable number of rows. Each row is an event and each column represents a base on the diamond in that event. $E_i$ indicates the event i, the $i_{th}$ row of the event table. The entry $E_{(i,j)}$ Indicates what happens for event i on base j, if there is a person on base j. Of course, if there is no runner on base j, the entry does not mean anything. The following is an example event.

| Out | 2 | 3 | H |
|---|---|---|---|

This is a row in the event table. Entry 0 is 'Out'. It says that if there is a person on base 0 (home plate), he out. Every time the program encounters an 'out', it checks the total number of outs. The program will continue if the total number of outs is less than 3. Entry 1 is 2; if there is a person on base 1 he goes to base 2. Entry 2 is 3; if there is a person on base 2 he goes to base 3. Entry 3 is H. H represents 'home' which means a run is scored. The total number of runs increment 1.

At run time, we generate random numbers to decide which events happen, thus the game is simulated.

## 2.2 Batters Table

The events table represents the game. Then the batters table (B) associates the accumulated data of each batter with each event defined in events table. If the event table has m rows, then the events table is an m × 9 matrix. Each column represents a batter in the game. The entry $B_{(i,j)}$ ($i_{th}$ row, $j_{th}$ column) indicates the probability of event i for batter j.

The probability is based on accumulated statistical Data. For any event i and player j, we need to count the total occurrences of i for both j and all other batters. It also can be represented by the following formula.

---

It also can be represented as the following flowchart:

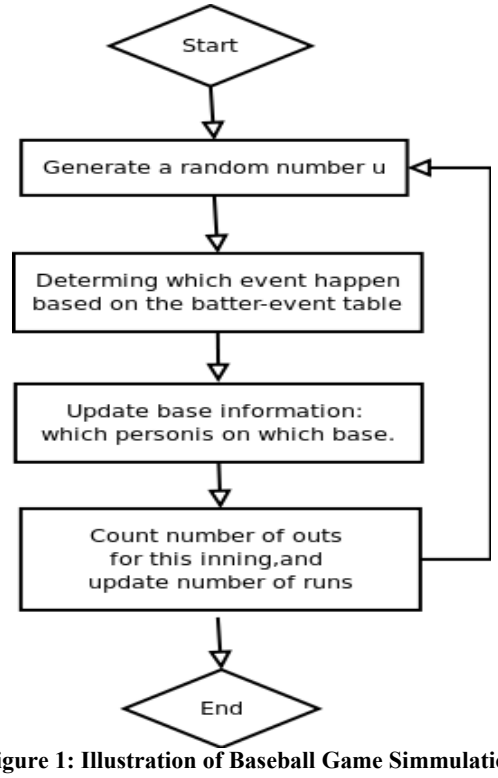


**Figure 1: Illustration of Baseball Game Simmulation**

The simulation contains 120 games. 9 innings is a game. 3 outs is an inning. Each bat is like a roulette which using a random number to determine which event happens. By the event, four bases, Scores, Outs updates. The goes to the next bat.

The simulation algorithm of baseball game as following:

```
for ( i = 0; i <120; i++) // 120 Games is an season
{
   for ( j = 0; j < 9; j ++) // 9 inning is a game
   {
      for ( k = 0; k < 3; k++) // 3 outs is a game
     {
          for ( b = 0; b < 9; b ++) // 9 batters
          u ← generating a random number from 0 to 1
          e ← getEvent(u, b) // compute which event happen
                                based on the batters table
          RcordBases(e) // update bases, count outs and runs
                           based on events table.
     }
   }
}
```

In order to search for a customized batting order, the simulation can only use a subset of the whole events set. By only allowing desired events happen, the GA will search for those batting orders which are more intend to let these events happen. The two tables used in the testing runs of the GA are given at the Appendix.

# 3. GENETIC ALGORITHM DESIGN

A batting order contains 9 batters. So the search space contains 9! = 362880 candidate orders. This is small for a GA, but we want to take advantage of the simulation. The more games we simulate, the more accurate the GA is.

## 3.1 Termination and Overall Structure

The EC terminates after a certain number of generations. And this is determined by an EC parameter. The overall structure of the EC is given as following pseudo-code:

```
while(i smaller than NUM_OF_GENERATIONS)
{

        Do the following λ times{
        Select parents and produce a child
        Push the chid back to the population pool.
        }

Select best μ chromosomes to form next generation's population.

i++
}
```

## 3.2 Candidate Solution Representation

Candidate solutions are permutations of 9 batters. It is nature that encodes each candidate solution as a permutation of nine symbols. So we number nine batters and represent each order as a string of nine numbers.

The GA's initial population consists of a random permutation of batters.

## 3.3 Selection and Survival Mode

Tournament Selection with tournament size 10 has been used in this EC. The tournament algorithm as following:

Randomly select 10 chromosomes form the population

Return the best one out of the 10 chromosomes.

The EC used simple (μ, λ) selection mode. For each generation, the system generates 50 children chromosomes which are all pushed back to the population pull. Finally, the system selects best 100 of them to form the population pool for next generation.

## 3.4 Variation Operators

A Swap mutation operator has been used. EC randomly generate 2 numbers in the range of 0 to 8 and swap the two numbers corresponding of these 2 numbers as index in the chromosome. And the number of swaps done for a mutation is determined by the mutation step size(mutation probability) EC parameter.

Example:

0 1 2 3 4 5 6 7 8 => 0 1 3 2 4 5 6 7 8

C1 Crossover Operator (Reeves [6]) has been used as following:

1. Pick a random number i from 0 to 8

2. Copy all symbols whose index smaller or equal to i from one parent to child chromosome

3. Scan the other parent and append all symbols, that does not in child chromosome, to the child chromosome in the order of appearance in the second parent.

Example:

0 1 2 3 4 | 5 6 7 8

8 7 6 5 | 4 3 2 1 0

Child Chromosome: 0 1 2 3 4 9 8 7 6 5

## 3.5 EC Parameters

Children Pool Size: CHILDREN_SIZE (50)

Population Size: POP_SIZE (500)

Mutation Probability: P_M (0.001)

CrossOver Probability P_X (1/7)

Selection Probability: P_S (0.8)

Number of Generations:

NUM_OF_GENERATION 1000

## 4. RESULTS, DISCUSSION AND CONCLUSION

The significance of the simulation and the GA can be measured by: (1) the progress of the fitness value. We use the biggest fitness value of the first and last generation to measure the progress of the fitness value. We are happy to see a big difference of these two values. (2) The climbing process. The climbing process is how fast the fitness value increased as the generation goes.

The first randomly generated order got fitness value 310 which means 310 runs has been achieved in the simulation. And in the final generation, about more than 23 percent of chromosomes have fitness value more 500. This considers being a big progress as searching. The searching results are strong advice to decision makers. The final generation of the GA offers a great selecting pool for decision maker to finally impose human consideration on selecting the best batting order.

The evaluation function still can be further developed to search for the best lineup. This expands the searching space dramatically. We can further apply parallel skills or improve the hardware to avoid reducing simulation runs.

## 5. APPENDIX

The following two tables referenced Dr. Robinson's Probability Simulation Lecture notes, see reference page. [4] Note that the purpose of this paper is demonstration of the algorithm. So the following data are not real data. But still it is good enough to demonstrate the effectiveness of the algorithm. The following tables are two subset of events and batters table.

**Table 1: Events Table**

| Home | Base 1 | Base 2 | Base 3 |
|---|---|---|---|
| -1 | -1 | 2 | 3 |
| -1 | 1 | 2 | 3 |
| -1 | 1 | 2 | 0 |
| -1 | 1 | 3 | 0 |
| -1 | 2 | 3 | 0 |
| 1 | 2 | 3 | 0 |
| 1 | 2 | 0 | 0 |
| 1 | 3 | 0 | 0 |
| 2 | 3 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

**Table 2: Batters Table**

| Bat0 | Bat1 | Bat2 | Bat3 | Bat4 | Bat5 | Bat6 | Bat7 | Bat8 |
|---|---|---|---|---|---|---|---|---|
| 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 |
| 0.27 | 0.25 | 0.23 | 0.21 | 0.23 | 0.27 | 0.31 | 0.35 | 0.39 |
| 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |
| 0.04 | 0.05 | 0.06 | 0.07 | 0.06 | 0.05 | 0.04 | 0.03 | 0.02 |
| 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.04 | 0.03 | 0.02 | 0.01 |
| 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 |
| 0.08 | 0.07 | 0.06 | 0.05 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.08 | 0.07 | 0.06 | 0.05 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.02 | 0.03 | 0.03 | 0.04 | 0.03 | 0.03 | 0.02 | 0.02 | 0.01 |
| 0.02 | 0.02 | 0.03 | 0.03 | 0.03 | 0.02 | 0.02 | 0.01 | 0.01 |
| 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.04 | 0.03 | 0.02 | 0.01 |

## 6. REFERENCES


[1] Palacios(1994), A markov chain approach to baseball, *Operations Research*, Vol. 45, No. 1(Jan. – Feb., 1997), pp. 14-23

[2] Freeze(1974), An Analysis of Baseball Batting Order by Monte Carlo Simulation, *Operations Research*, Vol. 22, No. 4(Jul. – Aug., 1974), pp. 728-735

[3] Official Baseball Game Rules

[4] Dr. Robinson's Probability Simmulation Lecture notes http://www.stcloudstate.edu/statistics/faculty.asp

[5] Chen, Batting Order Optimization using Evolutionary Computation. Not published. http://www.csie.nctu.edu.tw/~chenyy/FAAB/Lineup/report.pdf

[6] Reences C. (1996) Hybrid Genetic Algorithm for Bin-Packing and Related Problems. *Annals of OR* 1996; 63:371 – 396.