Algorithmic Maximization of Automated Golfball Collection

Zach Simon, Sam Vandello, Tanner Smith

August 22, 2013

1 Abstract

Golf is played all over the world and in all different forms, from the typical 18-hole round to driving ranges, Frisbee golf and putt-putt. Our problem considers an issue faced by every driving range in the business: finding an economically efficient way of collecting balls that have been hit by customers. To solve this issue, we have invented an automated robot vacuum that will travel the grounds and collect balls. However, even with such a golf ball collection robot, the question of what is the best route for the robot to travel that will maximize the number of balls collected is not obvious. We tested and analyzed multiple different strategies in an attempt to find the best such strategy for a generic driving range.

2 Motivation of the problem

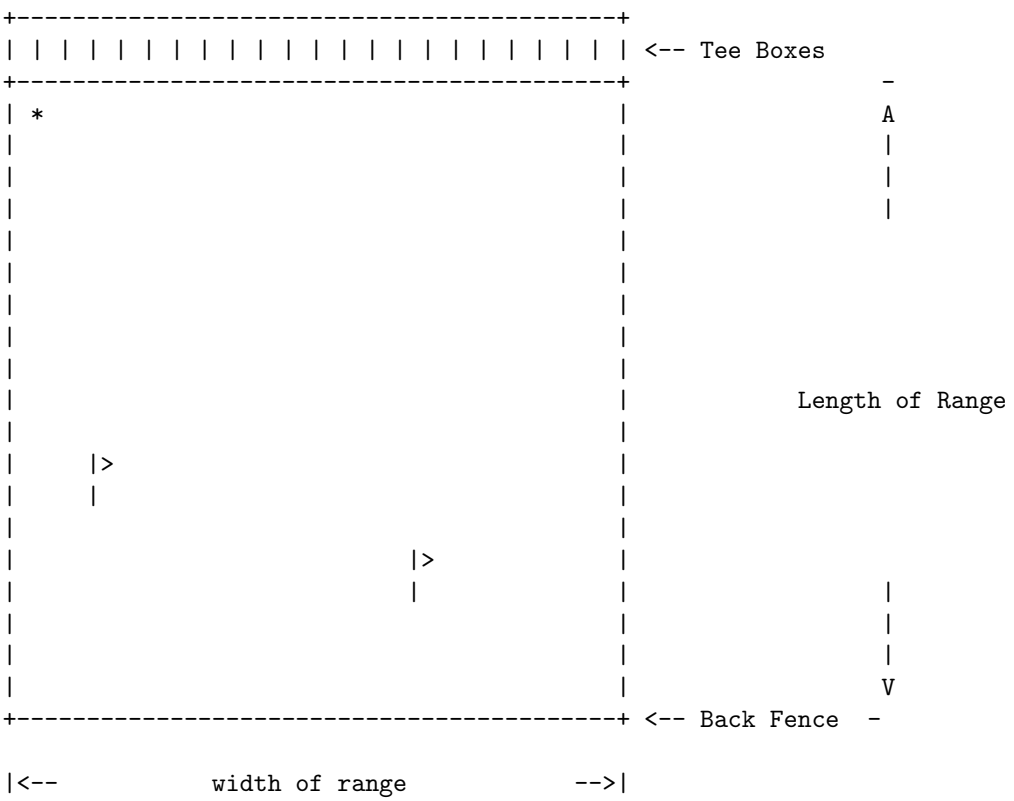
Suppose the owner of a driving range had grown tired of wasting labor on the trivial task of driving a golf ball collection tractor each day. He might decide that the most effective way to complete this task would be through an automated collection robot that swept the field without the help of a human. This robot wouldn't collect all of the balls on the range each day, but it would collect enough so that at the end of each day, the owner could drive his old human-controlled golf ball collector until he had collected the remaining balls. Thus, if the goal of the robot is to minimize the amount of labor that it takes to collect the remaining balls, the effectiveness of a robot is determined by the proportion of balls that it collects in a given day. However, even after having built such a robot to maximize the percentage of balls collected in a given path, the path is yet to be determined. In this paper, we seek to find a path that maximizes the number of balls that the robot drives over, which in turn should minimize the amount of time that the owner of a driving range would have to spend cleaning up after each day the driving range operates.

3 Simulations and Model

Our model was developed by viewing a driving range as a rectangle made up of rows and columns of three meter by three meter squares. At each timestep, a robot occupies exactly one such square and collects some number of balls from that square. After each timestep, the robot can move up, down, right or left to one of the squares adjacent to its previous square. If after a move, a robot is up against a fence and attempts to move further in the direction of the fence, it cannot move towards the fence for the next timestep.

3.1 Formulation of the Model

More formally, the driving range can be viewed as a matrix $A \in \mathbb{Z}^{m \times n}$. If the range is oriented such that when the range is viewed from above the tee boxes are at the top, then $A_{i,j}$ is the number of balls in the 3 m^2 square that is bounded by $3(i-1)$ above, $3i$ from below, $3(j-1)$ on the left and $3j$ on the right. In other words, $A_{i,j}$ is the square in the i^{th} row and j^{th} column of both the matrix A and the physical driving range. As this matrix evolves over time, it is convenient to denote both the element and the time when referring to an element, such as $A_{i,j,t}$.



$* = A_{1,1}$ (The upper left most cell in the Matrix is the upper left most cell of the driving range, when viewed in the orientation seen above).

At time $t = 0$, the robot starts in the upper left corner of the driving range, i.e. $A_{1,1}$. If at time t, the robot is at $A_{i,j}$, unless the robot is up against a fence in one or more directions,

- Moving left will result in the robot moving to $A_{i-1,j}$ for timestep $t + 1$
- Moving right will result in the robot moving to $A_{i+1,j}$ for timestep $t + 1$
- Moving up will result in the robot moving to $A_{i,j-1}$ for timestep $t + 1$
- Moving down will result in the robot moving to $A_{i,j+1}$ for timestep $t + 1$

Thus for each simulation, we can define a function, $L : \mathbb{Z} \mapsto \mathbb{Z}^2$ which takes in t as an input and outputs the row and column of the robot at time t. That is, $1 \leq L(t)_1 \leq m$ gives the row the robot is in at time t and $1 \leq L(t)_2 \leq n$ gives the column the robot is in at time t.

During each timestep, k$\sim$Pois(λ_b) balls are hit out on to the range. Although the driving range is assumed to have infinitely many golf balls (i.e. it can never run out of golf balls), in any one timestep, the maximum number of balls that can be hit out is $4\lambda_b$. The distance each ball travels can be expressed as the row, i, in which is lands, where i$\sim$Pois(λ_d). If $i > m$, then we let i = m since a ball traveling further than the length of the driving range will hit the back fence and land in the last row of the range. Each ball lands in column j, where j is a uniformly distributed variable between 1 and n.

If there are b balls in $A_{i,j}$ at time $t - 1$, and the robot moves into $A_{i,j}$ at time t, then the robot will collect $c = \lfloor \alpha b \rfloor$ balls, where $0 \leq \alpha \leq 1$, leaving $b - c$ balls in $A_{i,j}$ at time $t+1$, in addition to any new balls hit into $A_{i,j}$ during $t+1$. If a ball is hit into $A_{i,j}$ while the robot is in $A_{i,j}$, then that ball is added to b before the robot collects its balls. Therefore, each timestep can be thought of as three distinct steps: the distribution of new balls, the collection of balls from some square, and then the moving of the robot to its next square.

Each timestep represents 5 seconds, which, assuming an 8 hour day, makes for 5760 timesteps in one day. The model assumes that after the range closes for the day, management can shift workers to driving an old, non-automated golf ball collecting tractor to pick up the remaining balls on the range. Thus, the goal of the collection algorithms is to maximize the percentage of the balls collected after 8 hours, as this minimizes the labor required to pick up the remaining balls. The emptying of the range each night allows for two useful properties:

Unlimited Balls We assume that there is no limit to the number of balls hit out every day since if the driving range ran out of balls in a day, it would most likely purchase more balls before it would purchase a relatively expensive automated ball collector.

Limiting Behavior of Ball Collection This choice allows us to run our simulation for a large number of days as opposed to a large number of timesteps. This may seem like a trivial distinction, but it is not. The problem with running one very long simulation is that in order to have enough data points for any meaningful statistical analysis, the simulation must be run for a very long time. However, over that very long time, any reasonable algorithm will visit each cell in the matrix at least once every x moves, so that in the last x moves, every cell in the range has been collected from, where x is dependent upon the specific algorithm. But as the number of timesteps grows very large, the robot should expect to pick up about α of all of the balls hit out, since the proportion of the balls still currently in any cell gets smaller and smaller in relation to the total number of balls hit into that cell over the course of the simulation. Thus, we should expect with enough time that any robot that eventually visits every cell in the range will collect about α of all balls hit out. But since we want to see how well each algorithm does over a shorter period, it becomes necessary to break the simulation up into many different days.

Interestingly enough, since the lawnmower algorithm minimizes x, the number moves required to visit every single cell, we would expect it to do particularly well when run in shorter simulations. However, when the number of timesteps is less than x, the lawnmower algorithm does terribly because it doesn't focus its energy into the cells with higher expected values. For example, for a driving range with 1500 cells and timesteps < 1500, simulations run with the lawnmower algorithm collected about 16% of the balls, while the probabilistic greedy algorithm collected an average of 36% of the balls.

This also suggests that if the robot is only used for part of a day (which a reasonable driving range manager may well decide to do), the lawnmower robot would do relatively worse than our simulations show. Furthermore, this implies that if the robots had a finite capacity, resulting in a need to empty their balls during the middle of the day, the lawnmower would do relatively worse as well, since each robot would be sent back to $A_{i,j}$ at various points throughout the day to start over.

3.2 Solving for a Cell's Expected Value

Note that solving for the expected value of each cell is relatively easy at time t given we know the expected values at $t-1$. During timestep t, the expected number of balls hit into cell $A_{i,j}$ is given by

$$\mathbb{E}(A_{i,j,t}) = \lambda_b \left(\frac{1}{n}\right) \left(\frac{\lambda_d^i}{i!} e^{-\lambda_d}\right)$$

. Note that this value is not time-dependent, since the expected number of balls hit into $A_{i,j}$ is independent of t, however, we maintain the t index to keep

notation consistent. So if there is no robot present, the expected value of $A_{i,j}$ is given by the sum of the balls expected at each timestep, or:

$$t \times \lambda_b \left(\frac{1}{n}\right) \left(\frac{\lambda_d^i}{i!} e^{-\lambda_d}\right)$$

Define $\Theta(i,j)$ to be the last timestep in which the robot was in $A_{i,j}$, or 0 if the robot has not yet visited $A_{i,j}$, in other words,

$$\Theta(i,j) = \max_{k \geq 0} \{k | L(k) = \binom{i}{j}\} \cup \{0\}$$

Now, the expected value of each cell can be calculated at each timestep, which forms the backbone of some of our strategies. To do so, we initialize a matrix $E \in \mathbb{R}^{m \times n}$ to all zeros. That is, at time $t = 0$, we expect each cell to have 0 golf balls in it. Then, at each timestep we:

Account for the Balls Hit Out by Customers By incrementing each cell $E_{i,j}$ in E by $\mathbb{E}(A_{i,j,t})$, we are adding the expected number of balls hit into each cell during t

Account for Balls Collected by Robot During this Timestep Since we know what cell the robot is in (given by $L(t)$), and the rate at which it collects balls from that cell (given by α), at each timestep we set $A_{L(t)_1,L(t)_2} = A_{L(t)_1,L(t)_2} - \lfloor \alpha A_{L(t)_1,L(t)_2} \rfloor$ to account for the balls which the robot picked up.

Thus, the recursively defined expected value of $A_{i,j}$ is

$$\mathbb{E}(i,j,t) = \lceil (1-\alpha) A_{L(\Theta(i,j))_1, L(\Theta(i,j))_2, \Theta(i,j)} \rceil + (t - \Theta(i,j)) \times \lambda_b \left(\frac{1}{n}\right) \left(\frac{\lambda_d^i}{i!} e^{-\lambda_d}\right)$$

where $A_{L(\Theta(i,j))_1, L(\Theta(i,j))_2, \Theta(i,j)}$ is the number of balls in $A_{i,j}$ at the time when the robot last collected balls in $A_{i,j}$

3.3 Implementing the Monte Carlo Simulation

Our simulations implemented this model in Java, viewing the robot as an object that can query the simulation to gain information about the state of the driving range before returning its next move. Each simulation was run for d days before the datasets were imported to R for statistical analysis. Essentially, for each day, we ran code similar to that below:

```
for (each timestep){
    int b = nextPois(lambda_b); % generate random number of balls
                                % to be hit during this timestep
    for (each ball in b){
        int row = nextPois(lambda_d); % generates a random row
                                      % (i.e. distance) for a given ball
```

```
        int col = nextUnif(n); % generates a random column
                               % (i.e. width) for a given ball
        placeBall(row, column);
    }
    Direction move = Robot.getMove(); % ask the robot what its next move is
    moveRoomba(move);
}
```

For our simulations, we assigned the parameters as folows:

- $m = 75$ (thus, the driving range was 225 meters long)
- $n = 40$ (thus, the driving range was 120 meters wide)
- $\lambda_d = 50$
- $\lambda_b = 10$
- $d = 500$
- $\alpha = 0.8$
- timesteps = 5760

4 Assumptions and Simplifications

Due to the overall robustness of this problem and the many different variations that could be seen, we made a few assumptions while considering a solution. However, overall we feel that these assumptions adhere closely to reality, and any differences have a relatively small effect on our results. Furthermore, since we are looking for an algorithm that works well in a typical driving range, we need not consider extreme cases, such as arbitrarily large numbers of balls or arbitrarily large driving ranges, where these assumptions may not hold or seem plausible.

4.1 Robot has no collected ball capacity

One situation that we considered when developing our model, but did not implement in our simulations, was constraining robots to a fixed, finite ball capacity of c. Essentially, this capacity would force robots to move back to $A_{i,j}$ after they had collected c balls, where they would then empty their balls and continue collection. Clearly, this capacity would be realistic, and it might have an effect on our results, but we expect it to affect the naive algorithms (Lawnmower, Random) worse than the probabilistic algorithms (Towards Max, Probabilistic Greedy) because the probabilistic algorithms can continue to use the same expected value calculations after having to drop off balls, whereas the naive robots would simply have to continue as they did at the start of the day. Thus, the relative effectiveness of our algorithms would most likely not change much as a result of including this factor, even if the absolute collection rates changed somewhat.

4.2 Frequency of balls being hit and their respective landing places on the map

We assume that the particular driving range in which our robots operate are quite popular, and that there are always golfers occupying stalls. Our model assumes that the number of balls hit out during a timestep, and the row of the matrix which that ball lands in both follow a Poisson distribution. This alone introduces many assumption, none of which seem too implausible. Take for example the requirement that events be independent and identically distributed. The claim that each ball hit out from the driving range does not effect other balls is reasonable, and a worthwhile trade-off to allow our model to be able to use the Poisson distribution.

Furthermore, we must assume that the rate at which balls are being hit and the distance they are traveling is constant throughout the day and across days as well. It could be the case that at a given driving range, this assumption is not the case; maybe there is a mad rush at the end of the day by golfers with a ton of balls left to hit all of them before the range closes. However, since we don't have this data, we cannot assume that we know such information or include it in our model. Also, even if we did have this data about a specific driving range, incorporating it into the model may not serve to maximize the proportion of balls collected at all other ranges. Since our goal is to find a generalized formula that works for all driving ranges, it would not make sense to include such information in our calculations.

4.3 General capabilities of our robot and the driving range

Due to its high power motor and above average ball collecting capability (Thanks, mechanical engineers!), our robot is capable of collecting a proportion, α, of all balls on any one 5x5 square portion of our map in a time step of 5 second. In this interval, our robot collects said balls, chooses the next portion of the map it will visit, and moves itself there. As with most things in real life, this process is continuous: the robot does not quantum leap from one cell to the next. However, since the process of making decisions is inherently discrete, the choice to view the driving range as a matrix instead of a rectangle in $\mathbb{R}^2$ is not far fetched. Although allowing the robot to choose an angle at each timestep instead of a cardinal direction may have been more realistic, our model chose to abstract away this choice for both clarity and efficiency.

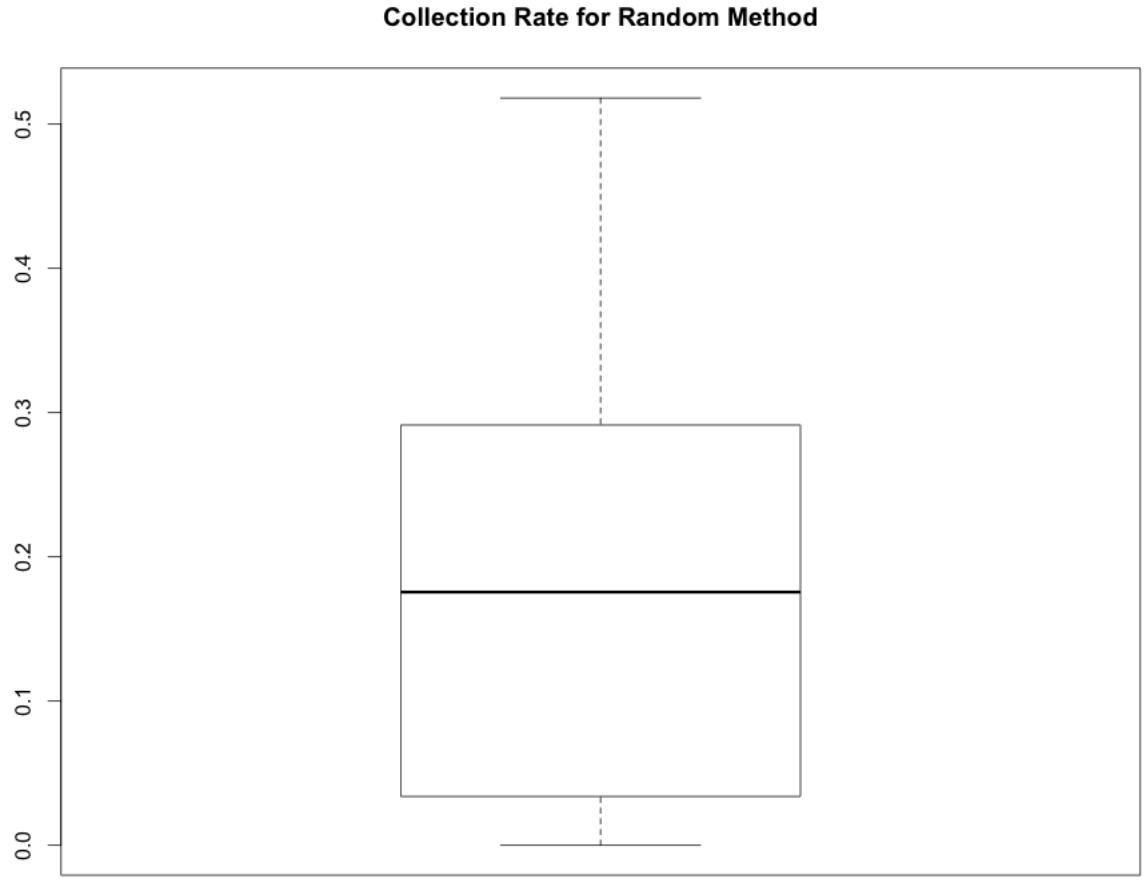
We also assumed that the robot would pick up a constant (but rounded) proportion of the balls underneath it. While in reality this proportion would most likely be a random variable, introducing this variance into our model would not help us tease out which algorithm was best since it would manifest itself as a higher variance in proportion of balls collected. Since this additional variance is neither desired nor helpful, we chose to make this function deterministic.

5 Algorithms and Their Effectiveness

After developing and implementing the strategies, each strategy outputted data. Those were balls hit out (the total number of balls hit out during a simulated day) and balls collected (The corresponding number of balls collected by that strategy). After running 500 simulated days for each strategy, the data was analyzed. Simple computations in R computed the mean, median, variance, and standard deviation relating to the success of each strategy. The success of each strategy, we determined, would be based primarily on two factors. One, after getting both strings of data, balls collected and balls hit out, they were divided to compute the percentage of balls collected for each simulated day. The strategy was then evaluated first by what percentage of balls the strategy collected on average, and secondly by how low the variance of each average collection was.

5.1 Base Strategy: Random Walk

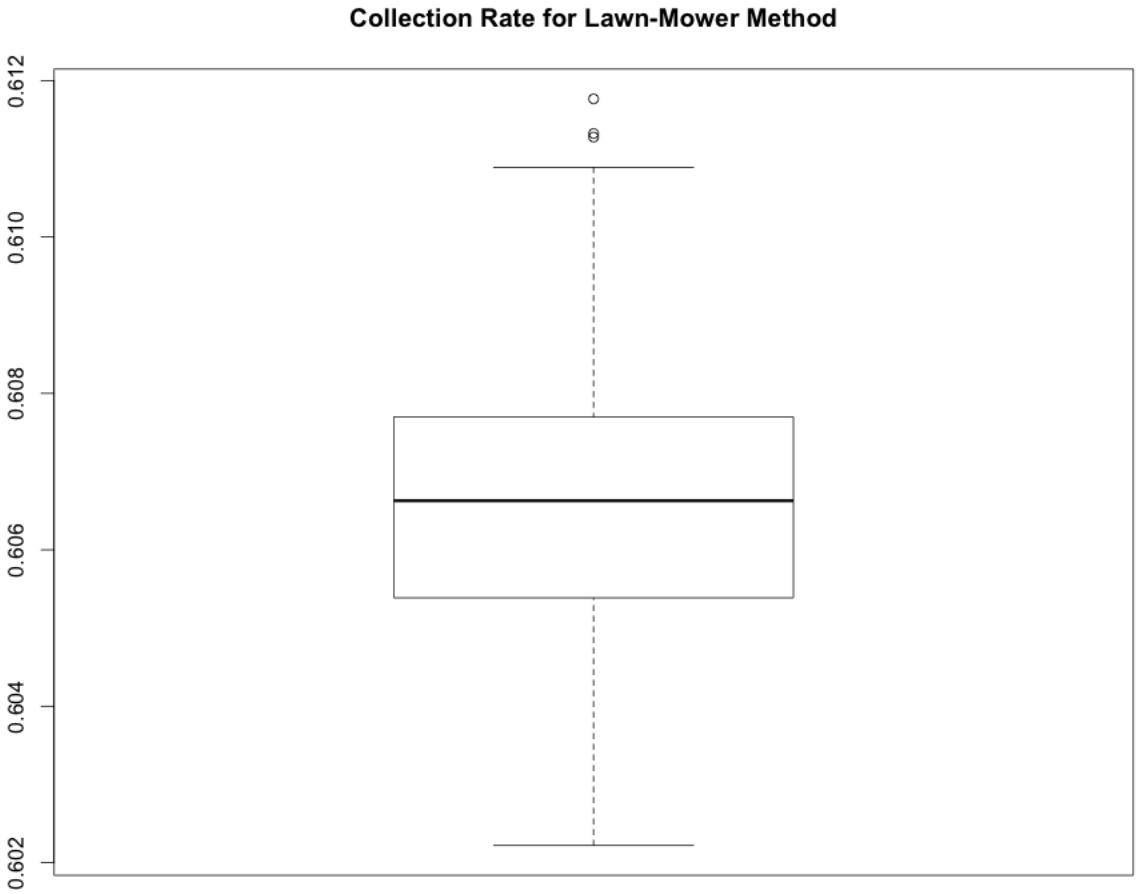
The control strategy for the robot is a random walk, and our goal is to find a strategy that is vastly superior to the random walk. The random walk works as follows. The robot is in a cell in the matrix denoted A_{ij}, the robot has four choices, move up, down, left, or right, each with probability .25. The robot traverses the driving range in this manner, picking up all the balls it can, and the totals are recorded. The random walk turns out to not be very effective and it can be wildly inconsistent. An inconsistent algorithm is not desirable since it would mean the driving range operator would never be sure whether they would have to spend hours or minutes collecting residual golf balls at the end of the day. The collection rate for the Random Method had the widest spread, as the values fell anywhere between 0 and 52.1%



5.2 The Lawnmower

The Lawn-mower strategy had an average rate of collection (over 500 simulated days) of 60.607%, meaning that given a random simulated day, that was the typical sample average. The variance was very low with the Lawn-mower strategy, coming in at 4.533103e-06. So in terms of consistency, this strategy was very reliable, showing basically no change on a day-to-day basis. It would make sense that this strategy is very consistent because the path that the roomba takes every day is exactly the same. And since the balls are distributed basically the same way given a long enough period of time, the strategy should show very little change on a case-by-case basis. The lawnmower algorithm traverses an entire column of the matrix before moving to the right one column

and then traversing that column in the opposite direction. The robot follows this pattern column after column until it finishes the last column, at which point it returns to (1,1) and repeats. Code is not presented here, but if this description is unclear, the graphical simulation can be run with this algorithm to clarify. The sequence of positions of a Lawnmower robot could be as follows: (1,1),(2,1),...,(m,1),(m,2),(m-1,2),...,(2,2),(1,2)...(m,n),.... at which point the roomba returns to (1,1) (which takes $m + n - 1$ steps) and repeats the pattern. Note that the sequences for ranges with an even number of columns and an odd number of columns are slightly different.



Note how small the scale is of the X-Axis, ranging only from .602 to .612.

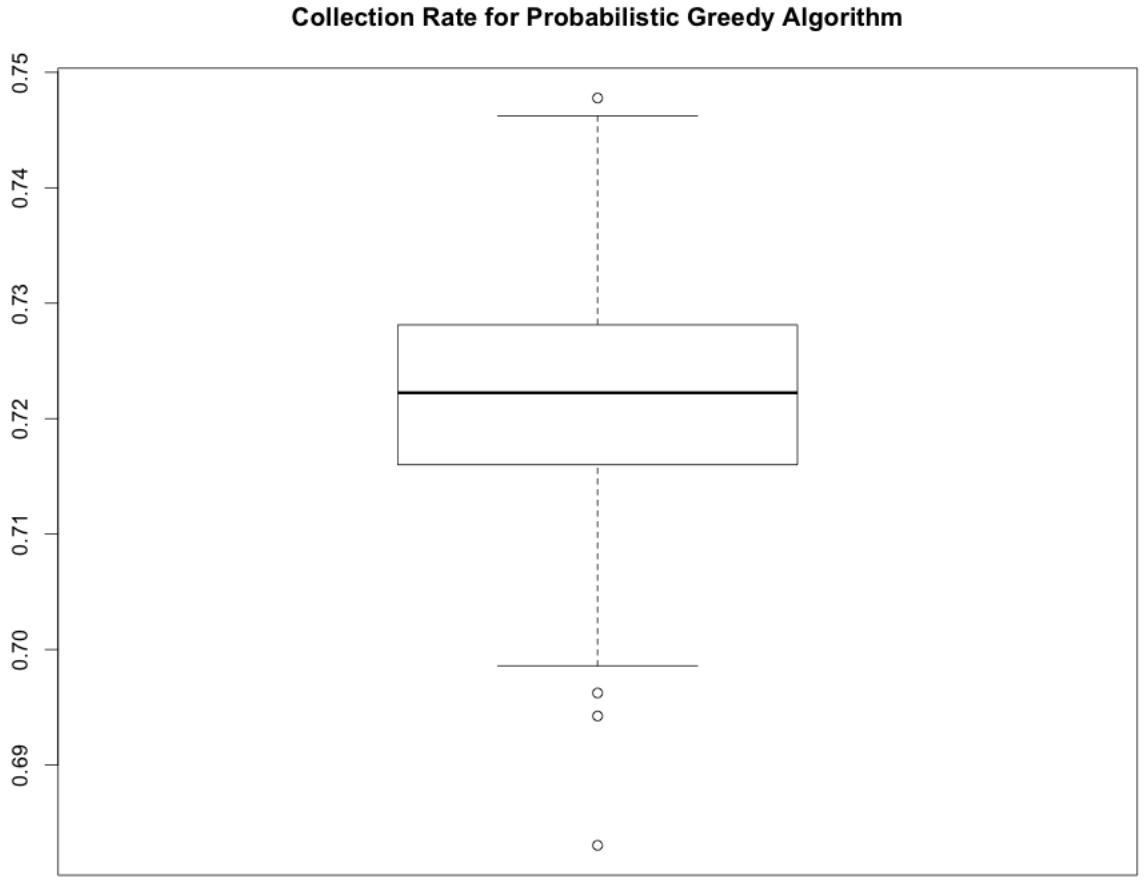
Compared to our control strategy, this was much more effective and consistent. The best case over the 500 samples was 61.2%, which was higher than the random walks best-case scenario, and since the spread was highly centralized at

60.7% it is a vastly superior strategy.

5.3 Probabilistic Greedy Algorithm

As described in the Model section, this robot is capable of solving for the expected value of each cell in the Range at any timestep, which allowed us to develop algorithms that used this information to choose a path. One such strategy, which we call the probabilistic greedy algorithm solves for the expected value in each of the surrounding squares, and moves into the square with the highest expected value. Essentially, this robot calculates the maximum expected value of valid adjacent cells and moves to a cell with that maximum (if there are multiple identical values, then it chooses a random move to maximize the expected value). One noteworthy aspect of this algorithm is that it is deterministic: with a well-defined algorithm for breaking ties (i.e. not choosing one at random, as our implementation did), two simulations run with the same parameters should have identical sequences for their robots' moves. For more information on how this algorithm calculates the expected value, such as how it corrects for balls collected in previous timesteps, see the subsection of the model that describes these calculations.

After running this method for 500 days, it had a collection average of 72.4% with a variance of 1.8%. This was vastly superior to the Lawn Mower method.

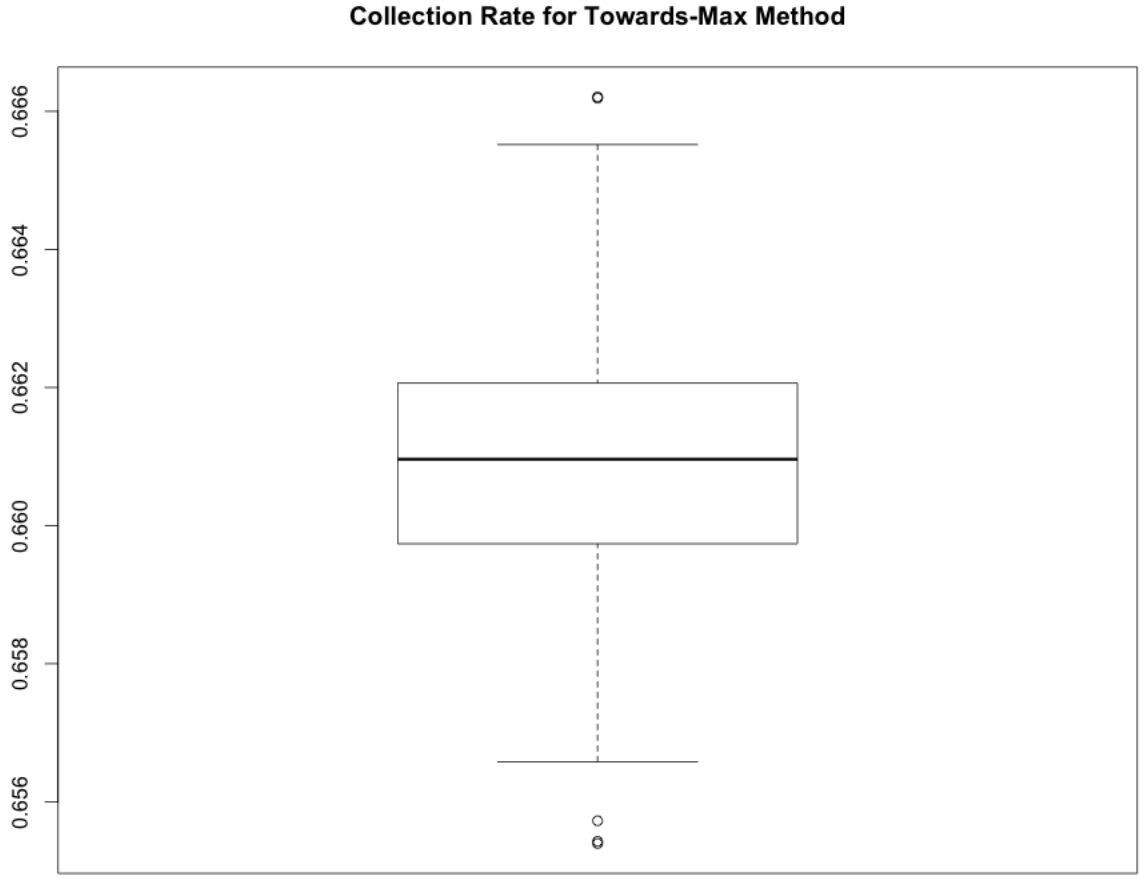


5.4 Towards Maximum Expected Value

Another algorithm that uses the same expected value matrix as the probabilistic greedy algorithm was the so-called Towards Max EV algorithm. This algorithm searched the matrix E for the largest expected value of any cell in the matrix, then moves towards that square, minimizing distance, until it has reached it, at which point it then finds the new maximum, and repeats. Note that even once the cell that maximizes the expected value is found, there are still many choices of the route to take to get to that cell that minimize distance. Our implementation simply first moved to the row that the maximum cell was in, then the column.

The Towards Max method produced results that were slightly superior to

the lawn mower, but not by a wide margin. The average collection rate after 500 days was 66.1%

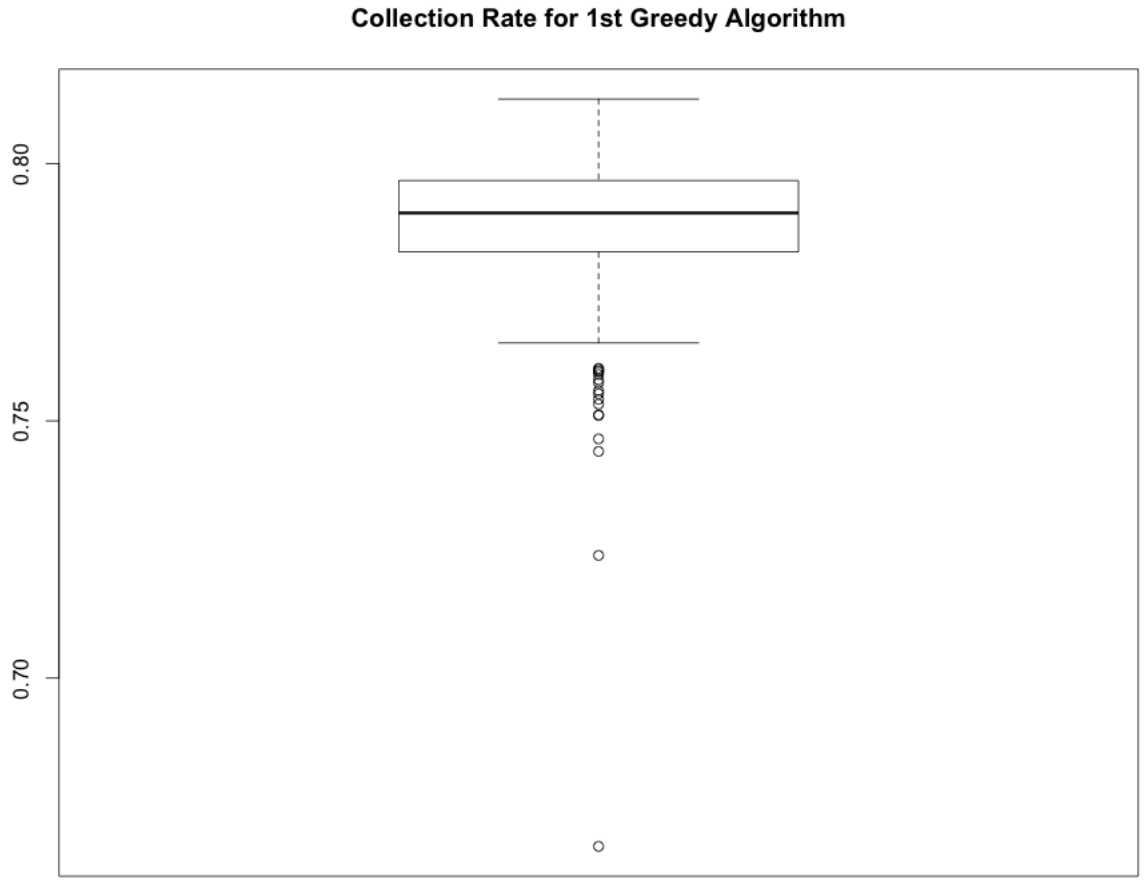


5.5 Greedy Algorithm

While not necessarily reasonable, we also simulated an algorithm that moves to the adjacent square with the most balls in it. This algorithm may not be of much practical use, because the robot must know exactly how many balls are in each square for it to work. None the less, we included this algorithm in simulations as a reference point. In the above simulation, this algorithm collected 78.8% of the 28800173 balls hit out. That this algorithm is more effective than any other tested suggests that a human-operated ball-collector would still collect more balls than any of our algorithms if the operator simply went in the direction of

most balls.

After running the Greedy Algorithm for 500 days, it yielded the highest average, at 78.8% with a very low variance of .0016



6 Conclusion

In our attempt to find the best algorithm for golf ball collection, we found that algorithms that make decisions based on the expected value of cells work particularly well for simulations with relatively few timeteps, however as the number of timesteps grows bigger, algorithms such as the lawnmower algorithm appear to work almost as well. Therefore, the algorithm best suited for a particular driving range is dependent upon their schedule, and how much man power they

devote to picking up residual balls. Our analysis suggests that, when compared to seemingly commonsense pickup strategies such as the lawnmower algorithm, our improved algorithms increased ball pick up rates for typical driving ranges, translating to fewer labor hours spent manually collecting balls and cost savings for driving ranges. This new class of algorithms which consider the expected value holds potential for even further improvements, as even with the expected value calculated, the choice of how to maximize paths with respect to expected value is farm from trivial. We also believe that these algorithms have the potential to increase efficiency in a wide array of applications.