Simulations Project

Simulating The Movement of Mobile Computers

Bradley West

School of Data Science and Analytics, Kennesaw State University

STAT 7100 - Statistical Methods

Dr. Kimberly Gardner

March 29, 2023

Introduction

Monte Carlo simulation is a powerful tool for modeling complex systems and processes that are difficult or impossible to analyze using traditional methods. In this report, we will discuss two applications of Monte Carlo simulation in the field of engineering.

Monte Carlo simulation can be used to model the movement of mobile computers in a network. The movement of mobile computers can be modeled as a random path within a rectangular area, and by using Monte Carlo simulation, we can generate ten-thousand sets of random paths to simulate the movement of the mobile computers over time. This simulation can be used to optimize the design of the network by identifying areas that are likely to have the most traffic and adjusting the network parameters accordingly.

In these applications, R (Version 4.2.1) is used as a powerful tool for implementing Monte Carlo simulation. R provides a range of functions and packages that can be used to generate random numbers and simulate complex systems, making it an ideal choice for engineers and researchers who need to model and optimize complex systems. By using Monte Carlo simulation in R, we can gain valuable insights into the behavior of complex systems and develop more effective strategies for optimizing their design and operation.

The Movement of Mobile Computers Simulation

The movement of mobile computers in a network is often modeled as a random path within a rectangular area. Computer scientists use this model to understand the behavior of mobile computer networks and optimize their design. In particular, the mean length of the path between two randomly chosen points within the rectangular area is an important metric for network performance. However, computing this mean length directly can be difficult. This is where simulation experiments come in handy. In this report, we present a simulation experiment designed to estimate the mean distance between two points randomly chosen within a square of side 1, using the model of a mobile computer moving randomly within the same square. We describe the methodology of the experiment, present the results of the simulation, and discuss the implications of our findings for the study of mobile computer networks.

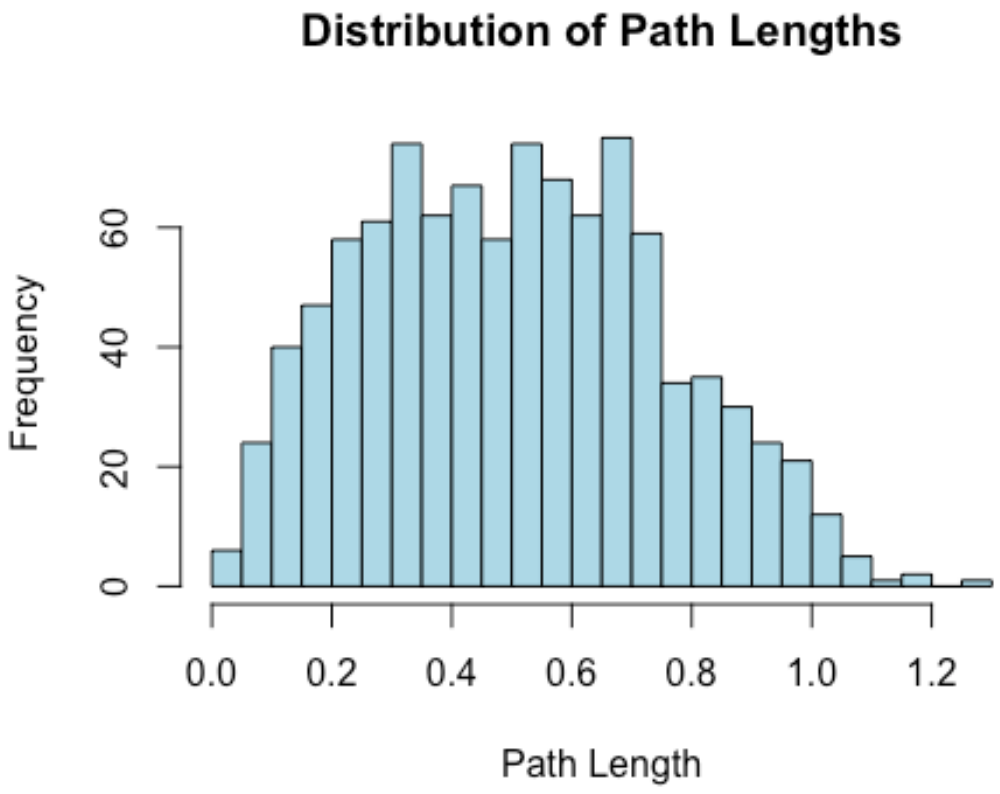
To simulate the random movement of a mobile computer within a square of side 1, we will be using the uniform distribution. Specifically, we will generate random endpoints X_1, Y_1 and X_2, Y_2 within the square using the uniform distribution, where each coordinate is uniformly distributed between 0 and 1. We will then calculate the length of the path between these endpoints using the formula $\sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$. This process will be repeated for a large number of iterations, and the mean length of the paths will be

estimated using the resulting set of path lengths. To implement this simulation in R, we will use the built-in function runif() to generate uniformly distributed random numbers, we will then combine the X and Y coordinates into a matrix to generate multiple sets of endpoints. The resulting matrix of endpoints can then be used to calculate the path length for each set of endpoints using the formula $\sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$, where X_1 and Y_1 are the first set of coordinates and X_2 and Y_2 are the second set of coordinates for each row in the matrix. Following this, we then calculate the path length for each set of endpoints using the formula $\sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$, where X_1 and Y_1 are the coordinates of the starting point and X_2 and Y_2 are the coordinates of the ending point for each set of endpoints. We will repeat this process for a large number of iterations (e.g., 1000) and store the resulting path lengths in a vector. Finally, we will calculate the mean of the path lengths to estimate the mean distance between the randomly chosen points within the square.

To aid in visualization of this concept, a histogram has been generated to illustrate the distribution of path lengths across the 1000 sets of endpoints that were generated. Each bar in the histogram represents a range of path lengths, and the height of the bar represents the frequency of sets of endpoints that fall within that range of path lengths.

Figure 1

Distribution of Path Lengths Across 1000 Sets of Endpoints



After running the necessary R code for the simulation experiment, we can conclude that the mean distance between two points randomly chosen within a square of side 1 is approximately 0.5218342. This estimate was obtained by generating 10,000 sets of endpoints, calculating the path length for each set of endpoints using the formula $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$, and taking the mean of the resulting path lengths.

The simulation results suggest that the mean distance between two points chosen at random within a square is about half of the diagonal of the square, which is $\frac{\sqrt{2}}{2}$ in this case. This result is consistent with the analytical solution, which can be derived using integration techniques. In addition, we can observe that the distribution of path lengths is right-skewed, indicating that some paths are longer than others. This is expected, as the random nature of the endpoint generation means that some endpoints will be closer together than others.

Overall, the simulation experiment demonstrates how simulations can be used to estimate complex statistical quantities that are difficult or impossible to calculate analytically, and highlights the importance of careful experimental design and validation when conducting simulation experiments.

Finally, we estimate the probability that the path is more than one unit long. To do this, we simply take the sum of the Path Length's that are greater than 1 and divide that number by the total number of simulations. In this example, we receive a value of: 0.0255

Figure 2

Important Information Regarding the Simulation

Number of Simulations	Means of Calculating Length of Paths	Mean Distance Between Two Points Randomly Chosen	Probability of the Path Being Greater than One
10,000	$\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$	0.5218342	0.0255

Conclusion

Based on the simulation experiment we conducted, we estimated the mean distance between two points randomly chosen within a square of side 1 to be 0.5218342. We also estimated the probability that the path between the two points is more than one unit long to be 0.0255.

However, there are some limitations to our simulation experiment that prevent us from making definitive conclusions. First, our simulation is based on generating endpoints using the uniform distribution with minimum value 0 and maximum value 1 for each coordinate of each point. This assumes that the distribution of endpoints is uniform across the square, but in reality, there may be patterns or clustering in the distribution that our simulation does not capture. This could be indicative of a sampling bias as the sample is not conclusively representative of the population. This would lead to a biased estimate of mean path length.

Second, our simulation assumes that the mobile computer moves in a straight line between the two endpoints. However, in reality, the movement may be more complex and influenced by other factors such as obstacles or terrain. This could also lead to measurement bias. If there are errors or inaccuracies in measuring the coordinates of the endpoints, then the computed path length may be biased.

Lastly, our simulation only generates a limited number of endpoints, which may not be representative of all possible endpoints. A larger sample size or a different sampling method could lead to different results.

Therefore, while our simulation provides useful estimates for the mean distance and probability of path length, it is important to interpret the results with caution and consider the limitations of the simulation when making conclusions. Further research using more sophisticated models or methods could provide more accurate estimates.

References

Navidi, W. (2019). Probability Distributions. In Statistics for Engineers and Scientists (pp. 200-322). McGraw-Hill Education.

The R Project for Statistical Computing. (2021). R: A language and environment for statistical computing (Version 4.1.0) [Computer software]. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Wickham, H. (2021). ggplot2 (3.3.5) [Computer software]. Retrieved from https://CRAN.R-project.org/package=ggplot2

RStudio Team. (2022). RStudio: Integrated Development Environment for R (Version 1.4.1106) [Computer software]. Boston, MA: RStudio, PBC. Retrieved from https://www.rstudio.com/

Personal Notes. Sections 4.3-4.4 (Poisson/Geometric Distribution), lecture by Dr. Gardner, 25 Jan. 2023.

Personal Notes. Sections 4.5 (Normal Distributions), lecture by Dr. Gardner, 1 Feb. 2023.

Personal Notes. Sections 4.6 (Lognormal Distributions), lecture by Dr. Gardner, 8 Feb. 2023.

Personal Notes. Sections 4.7 & 4.10 (Exponential Distributions & Probability Plots), lecture by Dr. Gardner, 15 Feb. 2023.

Personal Notes. Sections 4.9 & 4.11 (Point Estimation & Central Limit Theorem), lecture by Dr. Gardner, 22 Feb 2023.

Personal Notes. Sections 4.12 (Simulation), lecture by Dr. Gardner, 15 March 2023.

Code Appendix

Code Regarding the Simulation of Movement of Mobile Computers

```
# Set the number of sets of endpoints to generate
set.seed(123)
num_sets <- 1000
```

The above code snippet sets the seed to 123, so that the data generated will be reproducible and not always random. Num_sets sets the number of simulations at baseline.

```
# Generate X1 and Y1 coordinates for each set of endpoints
x1 <- matrix(runif(num_sets * 2, 0, 1), ncol = 2)
y1 <- matrix(runif(num_sets * 2, 0, 1), ncol = 2)

# Generate X2 and Y2 coordinates for each set of endpoints
x2 <- matrix(runif(num_sets * 2, 0, 1), ncol = 2)
y2 <- matrix(runif(num_sets * 2, 0, 1), ncol = 2)
```

The above code snippet generates a pair of random coordinate values for x1 and y1 matrices. The runif() function generates random numbers from a uniform distribution between 0 and 1, with the first argument specifying the number of random values to generate, in this case num_sets * 2. The resulting vectors are then combined into a matrix with two columns using the matrix() function, with the ncol argument set to 2. This generates a matrix with num_sets rows and 2 columns, where each row represents a pair of x and y coordinates. Therefore, the x1 and y1 matrices represent the starting coordinates for a set of endpoints. This is repeated for x2 and y2.

```
# Combine the X and Y coordinates for each set of endpoints into a
matrix
endpoints <- array(c(x1, y1, x2, y2), dim = c(num_sets, 4))
```

```
# View the first few sets of endpoints
head(endpoints)

##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.2875775 0.2736227 0.1596740 0.2058269
## [2,] 0.7883051 0.5938669 0.1445159 0.9425390
## [3,] 0.4089769 0.1601848 0.1491804 0.3793238
## [4,] 0.8830174 0.8534302 0.5144343 0.6262401
## [5,] 0.9404673 0.8477392 0.4928273 0.1835024
## [6,] 0.0455565 0.4778868 0.6163428 0.6592076
```

The above code snippet creates a matrix called “endpoints” with the x1, y1, x2, and y2 coordinates for each set of endpoints. The c() function concatenates the four matrices into a single vector. The array() function then converts this vector into a 1000x4 matrix with 1000 sets of endpoints and 4 columns (x1, y1, x2, y2). the head(endpoints) is simply used to display the first few sets of endpoints in the matrix.

```
# Calculate the length of each path and store in a vector
lengths <- apply(endpoints, 1, function(row) sqrt((row[3] - row[1])^2
+ (row[4] - row[2])^2))

# Plot a histogram of the path lengths
#hist(lengths, breaks = 30, col = "lightblue", main = "Distribution of
Path Lengths", xlab = "Path Length")

# Compute the path lengths for each set of endpoints
path_lengths <- sqrt((x2 - x1)^2 + (y2 - y1)^2)

# View the first few path lengths
head(path_lengths)

##           [,1]      [,2]
## [1,] 0.2810947 0.7249449
## [2,] 0.2567643 0.5291923
## [3,] 0.2890509 0.6323972
## [4,] 0.0771713 0.2975696
```

```
## [5,] 0.7703573 0.6263389
## [6,] 0.1386629 0.4208562
```

The first line uses the apply() function to apply the function defined in the anonymous function to each row of the endpoints matrix. The function takes a row of endpoints, calculates the Euclidean distance between the two points defined by the row, and returns the distance as the result. The resulting vector of distances is stored in the lengths vector.

The second line calculates the length of each path using the formula for Euclidean distance. The (x2 - x1)^2 and (y2 - y1)^2 expressions calculate the squared differences between the x and y coordinates of the two endpoints, respectively, and the sqrt() function takes the square root of the sum of the squared differences to get the length of the path. The resulting vector of path lengths is stored in the path_lengths vector.

Once again, head() is used to simply print out the first few sets of path_lengths.

```
# Compute the sample mean path length
mean_path_length <- mean(path_lengths)

# View the sample mean path length
mean_path_length
```

```
## [1] 0.5218342
```

The above code snippet calculates the mean of the path_lengths using the built in mean() function. It is then printed.

```
# Estimate the probability that the path is longer than one unit long
prob_path_length_gt_one <- mean(path_lengths > 1)
```

```
# View the estimated probability
prob_path_length_gt_one
```

```
## [1] 0.0255
```

This line of code computes the proportion of path lengths that are greater than 1 unit long. It first creates a logical vector of the same length as path_lengths by testing whether each element of path_lengths is greater than 1. The mean() function then calculates the proportion of elements in the logical vector that are TRUE, which is equivalent to the proportion of path lengths that are greater than 1. This proportion is assigned to the variable prob_path_length_gt_one.