William Joel Sandoval Casas

Evaluation of Randomized Movement Using Software

ENGR 0712: Advanced Engineering Applications for Freshman

February 23nd, 2024

Dr. Anna Balazs

Contents

List of Figures & Equations

Introduction

A tragic series of events during a diamond heist results in a police manhunt. In the search for the thieves, detectives establish a perimeter, or a circular outline encompassing the expected location of the criminals, to facilitate the apprehension of the dangerous individuals. With the help of the circular perimeter, police capture the convicts by searching the circular area piece by piece.

In the simulated scenario, the officers utilized the idea of random movement to predict the location of the convicts. Patterns derived from a randomized movement, however, are not solely applicable to law enforcement; rather, drunken walk, or random movement, has applications across a diverse range of topics. Focusing on technology, for instance, one can see how random walks are used to "approximate certain aggregate queries about web pages" [1]. On the other hand, looking specifically at migration patterns, researchers discovered that the long-distance dispersal of tiger sharks through the Hawaiian Islands can be simulated using random walk models [2]. Before analyzing patterns and datasets derived from random walks, it is important to fully comprehend the workings of a random walk. In essence, random walk is the idea that an object or a being moves in a random direction over a period of time; this process repeats itself for various iterations causing the starting entity to move a distance proportional to the number of iterations [3]. Noting the displacement generated by each movement of an entity in relation to starting point, or origin, during the random walk process is rather intriguing.

To replicate the drunken walk movement, a computer program using the development language C was created. This program was designed to test various factors of the drunken walk, including the displacement of an entity over a specified period of time, and the disruption of that entity's walk when forced to follow a self-avoiding path with obstacles of varying sizes present.

To simplify the measure of displacement, every iteration of random walk moves the entity by one unit of space and every motion lasts one second. Additionally, a self-avoiding path entails a movement where an entity cannot revisit a location it had been to before. The process by which random walk is measured and tested is described in thorough detail in the "Experimental Procedure" section of the "Methods".

Identifying a relationship between the randomized movements over an extended period of time may help generate patterns applicable to various aspects of life. In the three abovementioned examples, understanding the unpredictable movement of an entity benefits society. In the case of the jewel thieves, apprehending the dangerous suspects improves the safety of the nearby population. Similarly, tracking the movement of an aquatic apex predator can protect Hawaiian beachgoers and ensure their maximum protection while at sea. Monitoring and predicting the location of great tiger sharks can also prove vital to researchers who are analyzing changes in the region's aquatic ecosystem. Estimating the size of the web, on the other hand, can provide insight into the intelligence of the populace, by recognizing the amount of data available to the masses. Undoubtedly, patterns derived from drunken walks can contribute to the solution of a plethora of problems, such as those surrounding scientific or societal factors, applicable to a breadth of subjects.

Hypothesis

When testing parameters of random walk, predictions must be crafted to provide an anticipated understanding of experimental data. First off, as the number of steps increases in a two-dimensional plane, the displacement between an entity's current location and the origin is expected to increase. When referring to a two-dimensional region, movement can occur

exclusively in the positive and negative directions of a basic x and y axis, that is, only in four directions: left, right, forward, and backward. The probability of a particle's movement based on the number of steps can be modeled by Equation 1, where *n* represents the number of steps or iterations of a random walk.

$$probability(\%) = \frac{100}{(4^n)} \quad (1)$$

The relationship between the probability of an entity following a certain path and the number of steps the entity takes can be better seen in Figure 1.

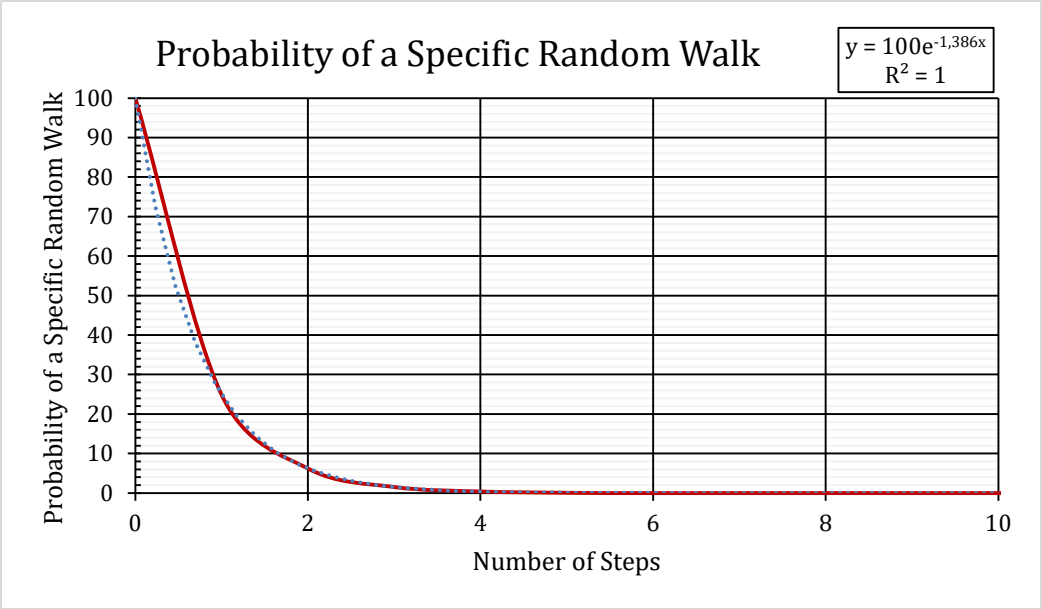

Figure 1: Probability of a Specific Random Walk

The graph above depicts the probability of choosing one specific path in a two-dimensional random walk. Displayed is also a dashed line of best fit, the line's equation, and the coefficient of determination. The probability of a specific random walk can help predict the displacement of an entity.

Figure 1 demonstrates an exponentially decreasing curve, which approximates a probability of 0 as the number of steps approaches infinity. To test the approximation, the

expression in Equation 1 is analyzed using limits and Equation 2 is crafted, demonstrating clearly how the curve, in fact, approximates to 0 as the steps reach infinity.

$$\lim_{n\to\infty} \frac{100}{(4^n)} \to \lim_{n\to\infty} \frac{100}{(4^\infty)} \to \lim_{n\to\infty} \frac{100}{\infty} \approx 0 \quad (2)$$

Figure 1 and Equations 1-2 portray that, as the number of steps in a random walk approaches infinity, the probability of predicting the random walk is 0%. Therefore, the possibility of a singular, specific path resulting in an entity returning to its starting position and having a displacement of 0 would be statistically impossible when tested with a large iteration of steps.

Focusing on a self-avoiding random walk model, that is, a representation of a drunken walk, where an entity is unable to enter a location it has previously occupied, it is anticipated that a being, on average, maintains an increasing displacement that slows down as the step count increases. Due to the very nature of the self-avoiding principle, the possibility exists that an entity traps itself and is unable to move further as the step count, indicative of seconds passed, increases. To calculate the probability of a self-avoiding entity trapping itself, Equation 3, inspired by Equation 1, can be calculated for $n = 7$, where n is representative of the steps required for an entity to trap itself.

$$probability(\%) = \frac{100}{(3^n)} \quad (3)$$

When computed, one concludes that there is a minimum probability of 0.0457% where an entity traps itself. Considering this information, trapped entities would reduce the overall displacement of a dataset, and as each entity approaches a step count of infinity, they are statistically bound to trap themselves. Consequently, a value is predicted to exist, where a maximum displacement of an entity after n number of steps occurs.

Applying the idea that an entity can trap itself to the presence of obstacles of various sizes, the probability of an entity trapping itself increases, due to obstacles resulting in fewer available pathways. For instance, when an entity is placed in a situation where it is beside a wall, the probability of becoming trapped increases drastically, from 0.0457% to 0.9259%, as demonstrated in Equation 4. A sample depiction of this process can be found in Figure 2.

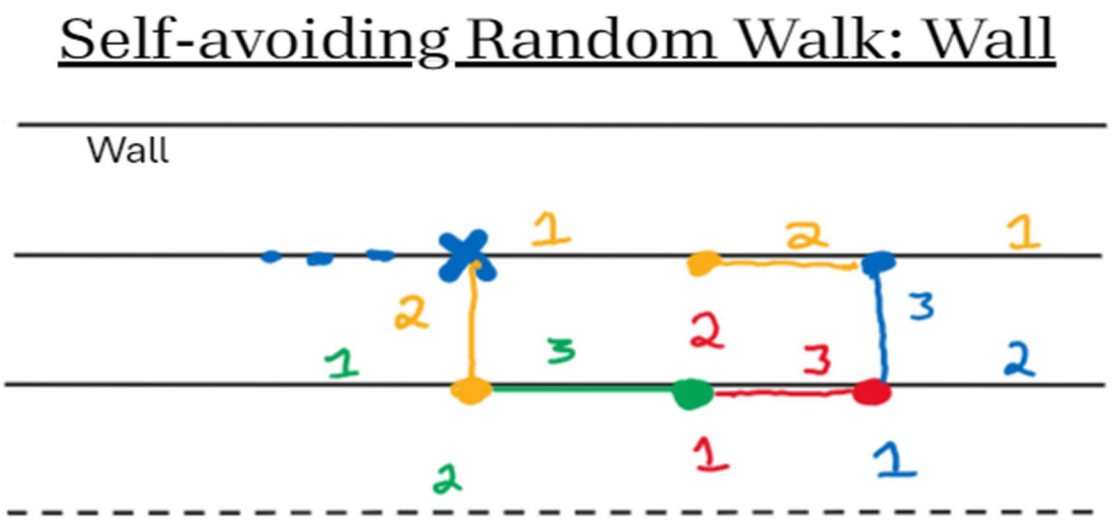

Figure 2: Self-avoiding Random Walk: Wall

The illustration above demonstrates a sample random walk beside a wall, where the entity becomes trapped. By taking the possible pathways of each movement required to become trapped, indicated by the values beside each junction, the entity's probability of becoming trapped may be computed.

$$probability(\%) = \frac{100}{(2*3*3*3*2)} \tag{4}$$

The case where self-avoiding motion induces trapping beside a wall relates to the presence of obstacles. The presence of obstacles, subsequently, results in a greater probability of an entity becoming trapped, therefore the maximum displacement an entity experiences in a field of uniform obstacles with a periodic placement is expected to follow the trend discussed for a

self-avoiding walk with no obstacles. In other words, a steady growing displacement is expected, until the step count is reached, where all tested entities are trapped, resulting in a line parallel to the x-axis. Opposite of the obstacle-free self-avoiding walk, however, as the obstacle size increases, the maximum displacement is also anticipated to decrease.

A thorough hypothesis allows for the creation of various predictions testable using guidelines outlined in the "Methods" portion.

Anti-Hypothesis

While testing the model of random walk from statements in the "Hypothesis" section, one must also consider, or rather prepare, for the unexpected. Therefore, the possibility exists that as a sizeable number of steps, representative of an approachment to infinity, occur in a randomized walk, the displacement of the entity equals 0. In other words, the entity returns to its origin after an infinite number of steps.

Additionally, the probability that the displacement of a self-avoiding model increases as the obstacle sizes in the 2-D plane increase must also be considered.

Methods

Experimental Procedure

The process of simulating a random walk in a computational environment begins with the creation of a script in the C programming language, capable of looping through sizeable iterations of code. In addition, this program must be capable of exporting data to a ".txt" file for further analysis in Microsoft Excel. For its versatility and ease in printing large datasets to an external file, the programming language C was chosen.

Specifications for this specific drunken walk experiment are as follows; the Visual Studio Code (VS Code) Integrated Development Environment (IDE) was utilized to develop code. The handcrafted code scripts were uploaded to a code storing software, Gitlab, to ensure the integrity of the code. Furthermore, a Windows Subsystem for Linux (WSL) in the Linux distribution, Ubuntu, was utilized, because "[Linux] optimally uses all the hardware resources available" which promotes faster code run-times [4]. A Lenovo IdeaPad Flex 5 laptop with a 64-bit operating system, 16GB of RAM, and an AMD Ryzen 7 7730U with Radeon Graphics-2.00 GHz processor was used to compile and run the C program.

Due to the complexity and various components of a random walk, two individual C programs were crafted; the first program gathered data in a non-self-avoiding random walk, and the second program gathered data in a self-avoiding random walk with obstacles.

The structure of the first program causes the sequential code to initialize a random number generator and a file pointer before entering a while loop, which repeats the entirety of the program a prearranged number of times, ensuring the storage of displacement data in each new iteration. Once in the while loop, a secondary, nested while loop, looping for the predetermined number of steps, is entered which calls the function *stepMaker* and *distanceP*. The function *stepMaker* randomizes the direction of the entity by increasing or decreasing a counter on the zero and one index value of the *xy* array, where the index 0 is the x-axis and the index 1 is the y-axis. The modified array is passed to the function *distanceP* which calculates the distance from the origin at that specific step count using the Pythagorean Theorem and stores that very distance in a designated index of the *weit* array. With each iteration of the entire program, the distance at a specific step count is added to the step count-specific index of the *weit* array. After all the iterations of the program have been executed, the while loop is terminated and

the sum of values in each index of the *weit* array are averaged in a for-loop based on the total iterations the program was run. In the same for-loop, the data is printed to a file using the previously declared file pointer. The file is promptly imported to Microsoft Excel using the "Get Data" feature and scatter plots with smooth lines, present in the later sections, is created.

Self-Avoiding Random Walk with Obstacles

Inspired by the approach taken to model a non-self-avoiding random walk, a secondary program, capable of noting every former location of an entity, is developed to ensure the functionality of the self-avoiding random walk model. This rendition of random walk features two major differences, still, it is almost identical to the abovementioned algorithm; this secondary program contains a random number generator, a file pointer, two while-loops promoting the functionality of operating the program for many iterations, and the slightly modified functions *stepMaker* and *distanceP*.

The array *xy*, previously of size two and storing exclusively the current x- and y-coordinates of the entity, is now a matrix of the size 1001x1001. Possessing a matrix format allows for the implementation of a self-avoiding algorithm, where an entity moves in any of four directions as long as none of the possible locations were previously occupied. An integer value of 0 on the matrix represents an open, unvisited space, and an integer value greater than 0 represents either an obstacle or a previously occupied space.

Besides the addition of a 2-D plane, a configuration of the random walk model with obstacles of varying sizes was implemented. Upon the declaration of the *xy* matrix, initially filled with zeros, a for-loop, spanning the size of the matrix, was entered, where obstacles of a constant size were placed at equidistant locations along the grid. Between the core or "nucleus" of each

obstacle, there were 5 blocks of separation. The size of all the obstacles in a test round was manually altered depending on the desired self-avoiding drunken walk data. A list of important metrics for the computer-simulated self-avoiding random walk models, regardless of their obstacle size, is illustrated below in Figure 3.

```
#define gridSize (1001)          //Constant size of grid
#define drunkenLadStart (501)    //Marks the origin, or point where entity begins
#define spacesBetween (5)        //Distance between obstacles (ex) 5 would mean 090000090
#define runData (1000000)        //Number of iterations where data is collected
#define steps (15000)            //Maximum number of steps taken by the entity
```

Figure 3: Conditions for Self-avoiding Random Walk

The image above depicts the global variables utilized in the C language representing key values in the self-avoiding random walks. Each of the five main variable quantities remained fixed through every representation of random walk. Additionally, next to each variable is a description of its intended action in the program.

Overall, the displacement of an entity reaching fifteen-thousand steps, tested for a million iterations, is calculated in a non-self-avoiding walk and a self-avoiding walk with zero, 1x1, 3x3, and 5x5 obstacles.

Results

Representations of Random Walk

The first model computed was the non-self-avoiding drunken walk, seen in Figure 4. During the first 1000 steps, a trend similar to a logarithmic curve is present, however, unlike a logarithm curve, the rate of change of the graph switched to an almost linear, constant relationship after 1000 steps. As the program reaches 15000 steps, the entity averages a distance

of a little over 175 units or exactly 175.144478 units. Using the tools provided by Microsoft Excel, a line of best fit was generated, represented in Figure 4 as an orange and green line.

$$y = 0.0104x + 22.318 \quad (5)$$

The exact formula for the line of best fit can be seen in Equation 5. Additionally, the dataset sustains a coefficient of determination or r^2 value of 0.9951.

Figure 4: Non-self-avoiding Drunken Walk

The graph above depicts the relationship between the displacement of an entity over an increasing step count. There were one million entities tested with the random walk algorithm over 15000 steps, their distance results were averaged to create the visual.

The secondary round of test trials was executed using the second C script, fine-tuned for the analysis of a variable self-avoiding walk model, where the obstacle size manually varies per testing iteration. Important to note is the data for each scenario was tested for 15000 steps, however, after 800 steps, the data for all sets remains constant. The exact point where the dataset reverts to a rate of change of 0 varies per testing iteration.

Before any obstacles are meticulously inserted into the matrix, the self-avoiding walk must first be tested in an unimpeded plane, that is, an obstacle-free field. In Figure 5.1, such a scenario is seen, where a self-avoiding random walk is executed for a million iterations, reaching a maximum distance of 11.9117079491 units, nearly reaching the 12-unit mark. Additionally, beginning at 737 steps, a slope of 0 dominates the data. In contrast, it took the entity 17-18 steps to reach its halfway point, of roughly 5.955853975 units.

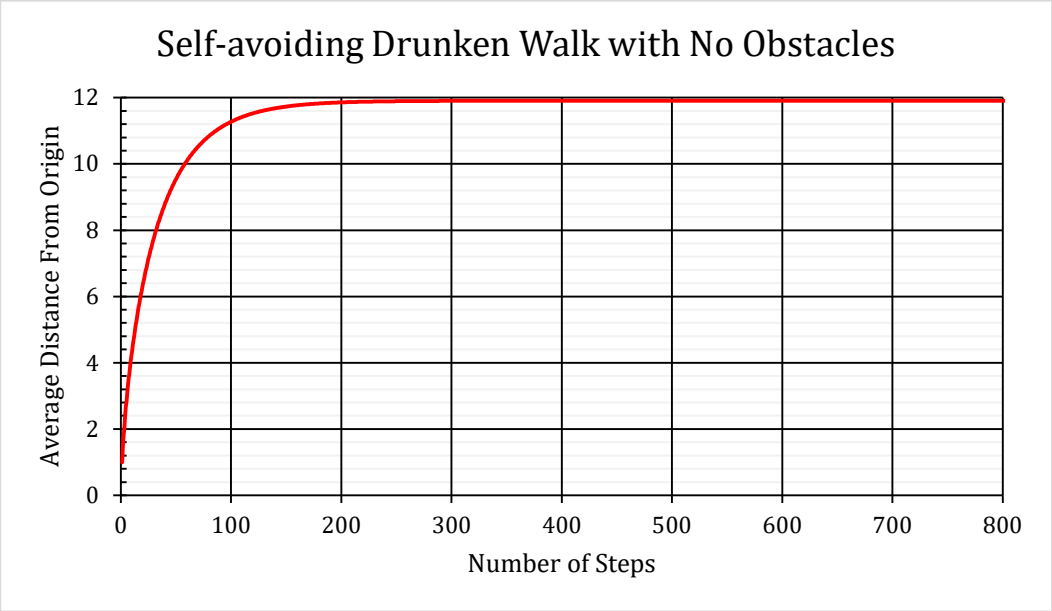

Figure 5.1: Self-avoiding Drunken Walk with No Obstacles

Illustrated above is a curve demonstrating the relationship between the displacement of an entity and the number of steps in any of the four maximum directions that an entity took while following a self-avoiding random walk model.

The addition of obstacles produces rather similar, yet intriguing, displacement results present in Figures 5.2-5.4. Looking specifically at Figure 5.2, however, 1-unit by 1-unit obstacles are added periodically in the two-dimensional field. The appearance of the graph itself mirrors that of a logarithmic graph, except the graph reaches and maintains a slope of zero after 589

steps. At that very point, the graph reaches a maximum y-value of 11.5243996465 units, representative of the entity's average displacement. Comparing the no-obstacle self-avoiding movement, with the 1-unit by 1-unit obstacles self-avoiding walk, an average difference in maximum displacement of 0,3873083026 units is computed. Also important to note is that at it takes between 16-17 steps for half of the maximum displacement to be reached.

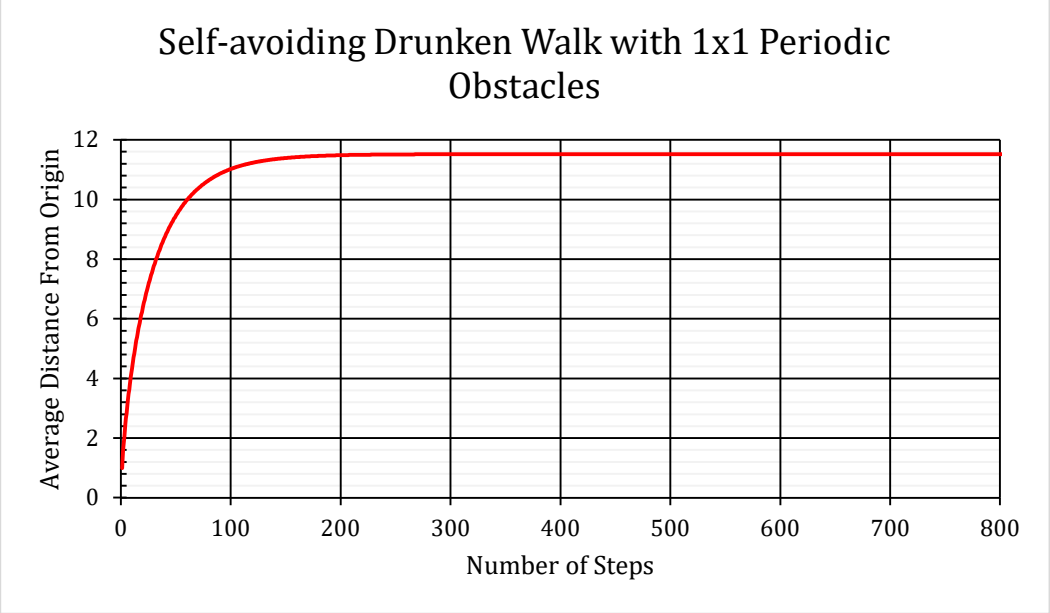

Figure 5.2: Self-avoiding Drunken Walk with 1x1 Periodic Obstacles

Present in the data visualization above is the relationship between the displacement of an entity after initiating a random walk from an origin. For this particular representation, a self-avoiding model is analyzed, where a particle-like object is unable to visit former residing locations. Spaced evenly throughout a 2D plane are 1-unit by 1-unit obstacles.

The observation of Figure 5.3 provides additional insight into the displacement of an entity after engaging in a self-avoiding random walk, particularly amidst 3-unit by 3-unit obstacles. Starting at 398 steps, the dataset extends to a maximum displacement of 9.3941363081 units. The maximum displaced is sustained for the remainder of the test; in other words, the rate

of change becomes 0. As the entity passes 11-12 steps, it exceeds half of its maximum displacement.

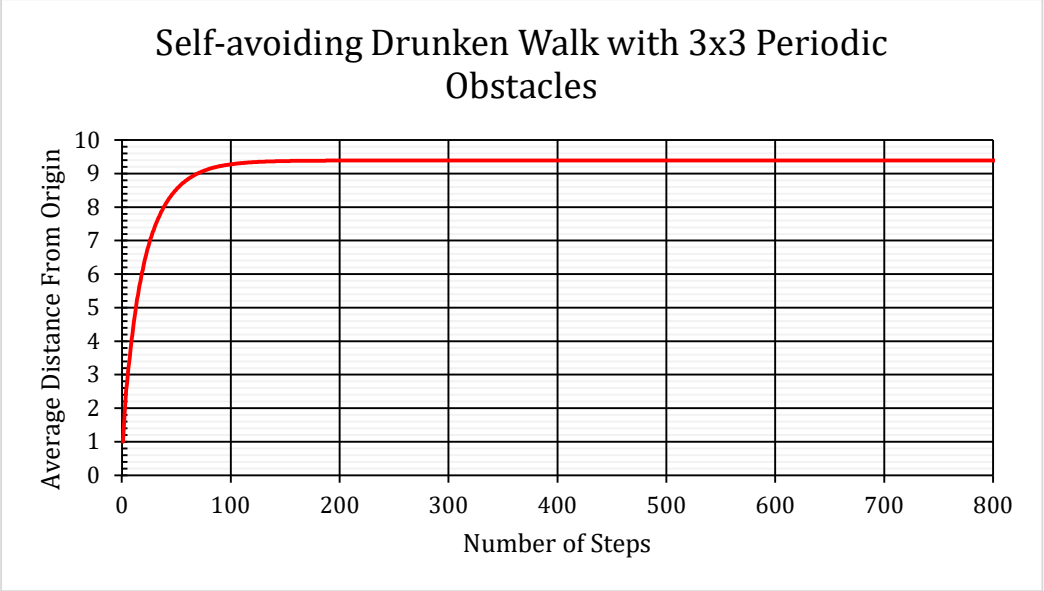

Figure 5.3: Self-avoiding Drunken Walk with 3x3 Periodic Obstacles

The dataset above features a representation of the displacement an entity experiences undergoing a self-avoiding randomized movement in a plane with periodic 3-unit by 3-unit obstacles. There is a distance of 5 units between the 1-unit by 1-unit core of each obstacle.

The final iteration of the self-avoiding random walk series introduces 5-unit by 5-unit obstacles into the two-dimensional field. Like before, these obstacles are spaced out from each other 5 blocks apart from the center of the obstacle. In this computational trial, a maximum displacement of 24.6908967528 units was attained at 732 steps. Furthermore, 18-19 steps are required for half of the maximum displacement data to be reached. Interesting to note is the fact this dataset reached a maximum distance a little over 2 times larger than the self-avoiding walk with no obstacles.

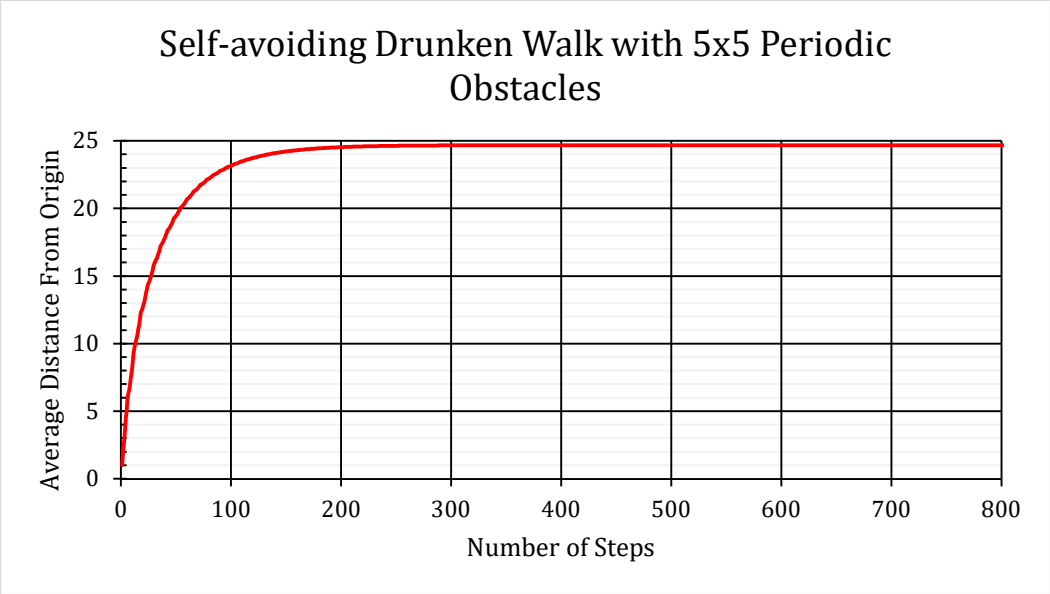

Figure 5.4: Self-avoiding Drunken Walk with 5x5 Periodic Obstacles

The graph depicted above illustrates the phenomena of self-avoiding randomized movement—a form of random walk where an entity is unable to reoccupy spaces in a matrix it has previously occupied. For this specific iteration of self-avoiding random walk, an entity is placed in the middle of a two-dimensional matrix with reoccurring 5-unit by 5-unit obstacles. The line demonstrates the displacement of the entity as the number of steps taken by the entity increases.

To recall, various parameters, including the addition of obstacles and the entity's status as non-self-avoiding or self-avoiding were altered for the execution of this experiment. However, various variables remained constant for all the five trials performed. Each trial was executed for a singular entity walking 15000 steps and each iteration of 15000 steps was fulfilled a million times, the data of which was averaged out for optimal results. Additionally, the number of steps is a representation of time, where one step is roughly equivalent to one second.

Discussions

Assessment of Random Walk

In the non-self-avoiding random walk model, displayed in Figure 4, a relationship between an entity's displacement after a positively increasing number of steps from a predefined origin is observed. For this specific test, the entity is allowed to trace its steps back and move in any of 4 directions, directly in the positive or negative x or y axis. Additionally, the step count for this dataset can be roughly translated to time, where a step is roughly equivalent to a second. While not specifically defined with a SI unit of distance, the length of each step measured is proportional to time, where a constant velocity in an entity is present.

Focusing on the abovementioned model, Equation 5 was derived to indicate a line of best fit for the dataset, with a coefficient of determination of 0.9951, only 0.0049 away from 1. The coefficient of determination, or r-square value, provides insight to the accuracy of an equation, in this case Equation 5, as it relates to the data graphed, seen in Figure 4; therefore, it is implied that Equation 5 possesses a strong relationship to the data for a non-self-avoiding random walk [5]. In other words, Equation 5 could be applied to external situations, where a basic drunken walk model could be functional; y would be representative of a certain distance, or radius, from origin, and x would denote the number of steps, or time in a specific unit, an entity undertook.

Recalling the situation involving law-enforcement from the "Introduction", having the patterns from the random walk model, represented in Equation 5, may help detectives narrow down the area enclosing a malignant individual.

Understanding patterns, relationships, and attributes in a randomized movement scenario can provide a better idea of the movement of an entity's movement over a set amount of time.

Assessment of Obstacles in Random Walk

As opposed to the almost proportionally increasing linear relationship between displacement and step count in a non-self-avoiding random walk, the self-avoiding drunken walk dataset follows an entirely different relationship. At the beginning of the self-avoiding datasets tested, illustrated in Figures 5.1-5.4, the graphs sustain a sharp upward trend. As the number of steps increases, the rate of change of the graphs decreases until an instantaneous rate of change of 0 units is reached. A better representation of the abovementioned trend can be seen in Figure 6, a compilation of Figures 5.1-5.4.

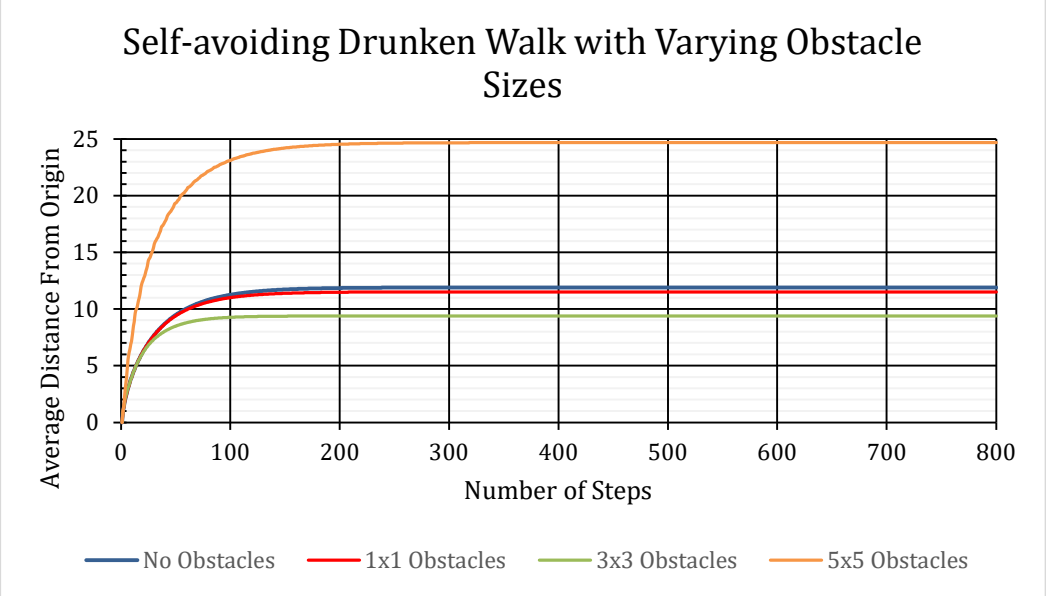

Figure 6: Self-avoiding Drunken Walk with Varying Obstacle Sizes

The graph pictured above demonstrates the displacement of an entity undergoing a self-avoiding random walk with four varying obstacle size conditions. The obstacle sizes tested include: 1-unit by 1-unit, 3-unit by 3-unit, and 5-unit by 5-unit. An additional test with no obstacles was performed. Furthermore, the legend displayed differentiates between each specific obstacle sample tested.

As the obstacle size is increased from 0 to 1 to 3 units of length, the maximum displacement reached by a particle decreases. However, as the obstacle size is increased to its final possible configuration, measuring 5 units in length, the maximum displacement is over two times greater than any of the other previous iterations. Since the distance between the centers of the particles is 5, the obstacles would, in turn, create periodic pathways of 1 unit of width and 6 units of length. In other words, there exist junctions at constant distances where an entity has the option to turn into one of three other pathways. Since the particle is essentially forced to travel at displacement factors of 6, it progresses more distance than other obstacle sizes, therefore possessing a greater maximum distance in comparison to the other datasets.

Another interesting characteristic of the datasets for self-avoiding random walk models is the plateauing-looking line in the graphs; in other words, the point when the dataset holds and maintains a rate of change of zero. Since a self-avoiding entity can trap itself, the steps taken after the entity traps itself and is unable to go anywhere, are essentially void. Therefore, the steps greater than a certain value, for instance x, recorded the same and final value that x possessed, so all steps greater than x would essentially have the same displacement for that iteration of the trial.

As mentioned in the “Introduction”, recognizing patterns derived from random walk models may provide valuable insight into various other fields and topics of study. Typically, instances where an entity’s location needs to be gathered, such as in the scenario where law enforcement work to catch a criminal or researchers attempt to track great tiger sharks in Hawaii, are possible periods where a random walk diagram could be utilized. In a similar fashion, an entity’s location and displacement can be observed using the self-avoiding random walk model. Utilizing the example with the criminal, the self-avoiding random walks, across the board, reveal

that during the first 11-19 steps, the individual walks 50% of their expected displacement. Similarly, after 398-737 steps, no movement is expected from a criminal. In a case where police officers patrol an area a criminal had just left, the criminal would displace over half their anticipated displacement within the first 20 steps. Then, after 737 steps, the criminal does not move in any of the self-avoiding models; therefore, at that point, the officers have succeeded in catching the criminal and keeping the town safe. Overall, random walks provide a vital understanding of a wealth of prominent subjects surrounding various fields of STEM.

To reiterate key findings, as the step count increases in a non-self-avoiding, obstacle-free two-dimensional randomized walk, the displacement, or instantaneous distance, from the origin also increases. Moreover, as the size of the periodic obstacles in a self-avoiding, two-dimensional random walk increases, the maximum displacement of an entity, opposed to popular belief, decreases. One exception exists, however, for the cases where the obstacles are 5-units by 5-units in area, the displacement from the origin increases in comparison to the trials.

Limitations of the Study

For any experimental procedure, an extent of human error is anticipated. Particularly in automated systems, where inaccurate operation of program due to human neglect could lead to skewed results [6]. The evaluation of random walks utilized software components, and while machines operate perfectly based on given parameters—the given parameters could be inaccurate [6]. It is people who program machines and input parameters, which the machine simply obeys. Therefore, any error in a computation occurs by fault of the individual who inputted the data and created the program, rather than the fault of the machine.

In a similar fashion, any observations made by humans could have some degree of error. Over-observing a detail in a dataset or under-observing it could provide inaccurate results.

Additionally, the generation of random values in the evaluation of randomized movement provides random integers. However, these random value generators output some rather predictable values, “on a completely deterministic machine you can’t generate anything you could really call a random sequence of numbers because the machine is following the same algorithm to generate them” [7]. Since an algorithm is employed to return a random value, that very algorithm can be reverse engineered to accurately predict upcoming random values [7]. The use of truly unpredictable parameters, such as changes in temperature or atmospheric pressure, would provide a genuine randomness for random value generators [7].

Recommendations

For future iterations of random walk, it is advised that more intricate variations of a randomized movement are tested. For instance, testing a non-self-avoiding model with periodic obstacle sizes would be interesting. Similarly, changing the radius between obstacles or placing them in irregular locations would provide interesting results, applicable to a wider range of topics.

Likewise, in an effort to diminish data corruption as much as possible, the datasets may be graphed directly using the computer program used to gather the data. While it is possible to create visual depictions of data using the programming language C, it is a rather cumbersome, challenging process. That is, however, not the case in all computer languages; MATLAB and Python provide various user-friendly features for the creation of graphs with various

customizations. Primarily, it is a researcher's personal decision whether they create graphs within a program or export the data for visual representation in a different software, like Microsoft Excel. It also depends on their knowledge of specific programming languages. That being said, data exported to a .txt file faces the minor risk of unintended manipulation, which may skew test results.

Lastly, a rate of change of 0 was observed in Figures 5.1-5.4 as the datasets reached their maximum displacement. This plateau-like relationship was mentioned in the "Assessment of Obstacles in Random Walk" section of the "Discussion". In essence, after the randomly walking entity was trapped, the remaining steps in that iteration stored the displacement the entity reached at the point it became trapped, explaining the maximum displacement in the data. For future test iterations, it would be interesting to analyze the effects of a self-avoiding walk where the entity does not trap itself. Since this approach removes some level of randomness from the experiment, it was not implemented into the experimental procedure, nonetheless, patterns and visual models derived from the abovementioned procedure would provide yet more patterns in random walk to analyze.

Additional iterations and patterns developed from random walk models would provide additional data to analyze, helping improve the world through specialized models and relationships.

References

[1] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol & D. Weitz. "Approximating Aggregate Queries about Web Pages via Random Walks." University of California at Berkeley. Accessed 12.02.2024.
https://www.vldb.org/conf/2000/P535.pdf

[2] Y. Papastamatiou, C. Meyer, F. Carvalho, J. Dale, M. Hutchinson, & K. Holland. "Telemetry and Random-walk Models Reveal Complex Patterns of Partial Migration in a Large Marine Predator." Ecological Society of America. 11.01.13. Accessed 17.02.2024.
https://esajournals.onlinelibrary.wiley.com/doi/10.1890/12-2014.1

[3] C. Rycroft & M. Bazant. "Introduction to Random Walks and Diffusion." Massachusetts Institute of Technology. 02.01.2005. Accessed 12.02.2024.
https://math.mit.edu/classes/18.366/lec05/lec01.pdf

[4] "7 Reasons Why Programmers Should Use Linux." Geeks for Geeks. 23.07.2021. Accessed 19.02.2024.
https://www.geeksforgeeks.org/7-reasons-why-programmers-should-use-linux/

[5] "Coefficient of Determination and Correlation Examples." Penn State: Eberly College of Science. Accessed 22.02.2024.
https://online.stat.psu.edu/stat462/node/97/

[6] "Designing to Reduce Human Error" Massachusetts Institute of Technology. 2013. Accessed 22.02.2024.
https://dspace.mit.edu/bitstream/handle/1721.1/106497/16-63j-fall-2012/contents/lecture-notes/MIT16_63JF12_Class14Human.pdf

[7] J. Rubin. “Can a Computer Generate a Truly Random Number?” Massachusetts Institute of Technology. 01.11.2011. Accessed 22.02.2024. https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

Additional References

W. Sandoval Casas. “Drunken-Walk.” 2024. Accessed 12.02.2024. https://github.com/sandoval-williamj/ENGR-0712/tree/main/DrunkenWalk

D. Johnston. “An Introduction to Random Walks.” University of Chicago. 08.05.2011. Accessed 12.02.2024. https://math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Johnston.pdf

“8 Reasons Why Linux Is Good For Coding.” The Bored Dev. 31.03.2023. Accessed 19.02.2024. https://theboreddev.com/8-reasons-why-linux-is-good-for-coding/

Acknowledgments

The creation of this assignment was facilitated through interactive brainstorming and cooperation with teammates. On that note, I would like to thank my drunken walk team, consisting of Jacklyn Wyszynski, Arun Krishnamurthy, and Lauren McMahon. Together, we were able to specialize and analyze how varying obstacle sizes affected random walks. More specifically, I would like to recognize my group partner, Jacklyn, for her valuable insights during our collaborative sessions developing our respective computer programs. Lastly, I am grateful

for the guidance and academic challenges provided by my professor, Dr. Balazs, and my teaching assistant, Dominick Filonowich.