Random Walks : The Properties, Applications and Methods of Analysis

Alexander Hill - Bachelor's Thesis

rijksuniversiteit groningen

1st Supervisor : Marco Grzegorczyk
2nd Supervisor : Wim Krijnen

University of Groningen
Mathematics
July 2020

Abstract

Random walks come in an array of interesting classes, each with unique properties, applications and methods of analysis. This paper will provide an analytical and numerical analysis of the different classes of random walks, and study the relationships that connect them. This paper will introduce and discuss the key concepts of simple random walks, Levy flights, reinforced random walks, self-avoiding walks, and Brownian motion. Following this, new research results will be presented. First, an array of numerical evidence will be introduced to support the Levy Flight Foraging Hypothesis. Further to this, upper and lower bounds for the connective constant of the Union Jack lattice will be implemented numerically. Additionally, a new method of analysis will be developed to study the Narrow Escape Problem. Lastly, an extension to the reinforced random walks will be constructed to link reinforced random walks to self-avoiding walks.

Contents

List of Figures

1 Introduction

The random walk is one of the most fundamental models in probability theory, demonstrating profound mathematical properties. It has a broad range of applications in various scientific fields such as physics, chemistry and biology. The history of random walks began in 1905, where Karl Pearson attempted to model the random migration of mosquitoes in *A Mathematical Theory of Random Migration* [1]. In *'Nature'*, Pearson described the specific results he was looking to demonstrate. The following week, Lord Rayleigh pointed out the connection between Pearson's problem of random mosquito migration, and the mathematics behind sound vibrations [2, 3]. The underlying model both mathematicians were studying was the random walk.

Formally, a random walk is any process that follows a procession of random steps in a mathematical space such as a lattice or graph. The direction and magnitude of these steps can follow varying probabilistic distributions, leading to interesting properties to study. Random walks can be found within diverse areas of mathematics and science, from animal foraging and cell movement, to ferromagnetism and long-chain polymers [4–7]. As a result of this, they are of great interest to academics, with results in the field having countless initially unforeseen applications.

The primary goal of this project is to provide a clear analysis of the statistical properties and theorems of different classes of random walks, through both an analytical and a numerical lens. Additionally, the applications of random walks shall be explored and the various methods of analysis will be discussed. To achieve these goals, this paper will first introduce the different types of random walk, alongside their properties and some key proofs. The paper will then move on to a collection of new results in the field of random walks.

There are many different classes of random walk, with each class harboring an extensive array of interesting properties and applications. Covering this material in the necessary depth would be beyond the scope of this paper. For this reason the Theory & Methods section will discuss a non-exhaustive selection of random walks in order to provide a relevant, diverse and coherent analysis. Furthermore, the properties, applications and methods of each class will be chosen in order to emphasise the most interesting and significant aspects of that class. The literature for each class will be given throughout this paper and in the bibliography, thus a deeper look into each class is possible for the curious reader. The primary contribution to the Theory & Methods section will be the careful selection of material across a range of literature, and constructing a system of notation to bring it all together in a coherent and clear manner.

This paper will first discuss the random walks that take steps independently to the path behind them, thus demonstrating the Markov Property. These walks will be referred to as *Markovian.* Following this, the *Non-Markovian* walks will be defined and explored. Below is a table of well studied random walks, categorised by discrete/continuous and Markovian/Non-Markovian. The classes that will be discussed in this paper are denoted by a *****.

	Discrete	**Continuous**
Markovian	*Simple Random Walk *Levy Flight Maximal Entropy RW	*Brownian Motion Schramm–Loewner Evolution
Non-Markovian	*Self-Avoiding Walk *Reinforced Random Walk Loop-Erased Random Walk	

We begin with the most elementary class of random walk, the simple random walk. This walk serves as the basis upon which more complex walks can be later defined. The simple random walk maintains a fixed step-size and acts independently of its previous steps, making it Markovian. Therefore, the simple random walk can be modelled as a Markov chain, allowing for a stronger analysis than walks that cannot be modelled as so. In *Random walks and electric networks* [8], the simple random walk and its properties are discussed in both finite and infinite spaces, and will be the primary guide to this section of the paper. Properties about the simple random walk will be explored, mainly focusing on the expected movement of the random walk as the number of steps n increases. The rate of diffusion of the simple random walk in one dimension will be demonstrated analytically, along with a few other key theorems. Additionally, a particular focus will be given to the application of two dimensional simple random walks in modelling the diffusion of temperature on a solid surface. To achieve this, the Monte-Carlo method and the Method of Relaxation will be introduced and then later compared in the Results & Simulations section. For additional sources that covered the simple random walk and aided this section see [9–14].

The step-size of a random walk can follow various distributions as opposed to being fixed; with a heavy-tailed distribution we observe a random walk called the *Levy flight*. The heavy tails of the distribution allow for large jumps to occur, making the Levy flight suitable for modelling the movement of foraging animals [4, 15–17]. The different types of Levy flight will be defined and key properties will be discussed. The paper *Introduction to the Theory of Lévy Flights* [18] will outline the theory for this class of random walks.

Following this, the Non-Markovian random walks will be discussed. The *reinforced random walk* will be defined and it's properties will be discussed. The reinforced random walk is widely applied to probability theory, it correctly models many systems such as *Polya's urn*, which will be discussed in this section [19]. The reinforced random walk is defined such that the probability of traversing edges of a graph is dependent on the number of times it has been traversed in the past. This can be done in many different ways, leading to interesting recurrence properties. The primary source for this section will be the paper *Reinforced random walk* [19].

The *self-avoiding walk* (SAW) will be the last of the Non-Markovian random walks to be discussed. SAWs have the condition that a site, once visited, can never be revisited. From a mathematical perspective, there is a lack of rigorous analysis about the self-avoiding walk, many of its properties are conjectured and then supported by numerical simulations, a notoriously unsatisfactory result for mathematicians [20]. SAWs are of considerable interest as they are applicable to the research of many lucrative domains, such as long chain polymers in Chemistry or the structure of DNA in Biology [6, 21]. Additionally, the enumeration of SAWs of different lengths is a classic problem in combinatorics, due to its surprising complexity [22, 23]. The book *The Self-Avoiding Walk* [24] provides clear definitions of the self-avoiding walk and its properties, and will thus be the primary guide to this section of the paper. The *critical exponents* define the asymptotic behavior of SAWs, and will be introduced in this section. The most sought-after and intriguing of the critical exponents is the *connective constant*, denoted μ. The connective constant can be thought of as the average number of possible edges the walker can traverse at each step of a long walk. The connective constant is notoriously difficult to find analytically, thus numerical methods have been developed to approximate it. The methods of finding upper bounds to the connective constant will be defined, such as Alm's method demonstrated in [25]. Then, the methods of determining lower bounds will be defined including Keston's method of irreducible bridges [26–28].

The final random walk to be discussed will be Brownian motion; the continuous-time scaling limit of the random walk [29]. The definition of Brownian motion will be given along with a construction of Brownian motion from a simple random walk[30–32].

Following the theory and methods of the different classes, some new research will be presented in the Results & Simulations section. This section intends to advance forward from the known theory to bring new independent research to the field of random walks. Firstly, the statistical properties of the simple random walk will be validated through simulation. Properties include the expected distance of the walker from the origin after n steps and the asymptotic distribution of the random walker's final position. Following this, a numerical comparison between the Monte-Carlo Method and the Method of Relaxation will be conducted via direct simulation.

After this, the *Lévy Flight Foraging Hypothesis* will be discussed, and numerical simulations will be conducted to test the hypothesis [4, 15–17]. The Levy Flight Foraging Hypothesis states that animals have evolved to utilise the Levy flight as a means of searching for food efficiently, as opposed to a simple random walk. By simulating a hungry animal searching blindly for food, the simple random walk and the Levy flight will be compared as a searching strategy. Additionally, an evolutionary algorithm will be applied that aims to maximize the probability of survival of the foraging animal. This will demonstrate the inherent advantage of the Levy flight as the animal will evolve from a simple random walk into a Levy flight as generations pass. This will give strong numerical evidence to support the hypothesis.

The methods of approximating the connective constant will be applied to the Union Jack lattice. SAWs are directly linked to the Ising model of ferromagnetism [5, 33, 34]. As a result, the critical exponents of the self-avoiding walk have a quantitative relationship with the parameters of the corresponding

Ising model [34], thus approximations of the former, lead to contributions in the latter. The Union Jack lattice has indeed been studied in the field of ferromagnetism [35–38], but no computation of the connective constant has been calculated. This thesis will finally provide an approximation for the connective constant on this particular lattice, onto which further research can be made.

The *Narrow Escape Problem* will then be defined and discussed. The Narrow Escape Problem describes a Brownian motion confined to a container, with a small gap where it may escape. The problem is in finding the estimated time until it gets out of the container. A new method of modelling the problem will be introduced and analysed using the theory outlined in this paper.

To conclude the Results & Simulations section, this paper will construct an extension to the class of reinforced random walks. This generalisation offers a starting point for a more rigorous analysis of Non-Markovian walks such as SAWs. In essence, the generalisation stems from the idea that reinforced random walks can 'see' the number of times each edge has been traversed in its near vicinity, and makes choices accordingly to its predetermined function. This paper will extend the 'reach' of this reinforced walk, such that the probability distribution is determined by edges that are further away than the regular case. A reinforced random walk with (reach) $\rho = 2$ can 'see' not only the edges connected to its own vertex, but the edges connected to the vertices one step away from itself. This paper will show that a particular case of the reinforced random walk with $\rho = 2$ can be used to construct a new definition of the self-avoiding walk, potentially allowing for a stronger analysis of its properties. This section will focus on developing this new extension of reinforced random walk and formulating a rigorous definition for future research to be done.

Lastly, this paper will discuss the implementation of the methods and simulations outlined throughout the paper. This section will describe the strategy of encoding different methods and the specific technical details required to produce the given results. The raw code will be also added for clarity and transparency.

2 Theory & Methods

This section will bring together the relevant methods and properties across a range of literature to give a clear and coherent overview of each class of random walk. In order to do this, a large amount of notation had to be constructed and edited in order to display all of the ideas in one consistent format. Almost all of the theory in this section has re-written proofs and ideas in order to make the necessary improvements/clarifications and 'glue it all together'. Before moving onto the different classes, we must discuss the different domains that random walks can take steps upon.

Random walks can take steps on a variety of different domains. A random walk certainly exhibits vastly different behavior on a one-dimensional line as it would on a torus. The most simple of these domains are simply connected spaces such as $\mathbb{R}$ and $\mathbb{R}^2$. The notion of a graph is particularly useful to the study of random walks;

Definition 2.1. A graph is a pair $G = (\mathrm{V}, \mathrm{E})$ where V is a set of vertices and E is a symmetric subset of V $\times$ V, containing the set of edges. E is symmetric iff $(x, y) \in \mathrm{E} \iff$ (y, x) $\in$ E.

Examples of graphs are $\mathbb{Z}$ and $\mathbb{Z}^2$ coupled with edges connecting each neighbouring integer/co-ordinate. This paper will deal primarily with random walks on graphs and simply connected spaces.

2.1 Markovian Random Walks

2.1.1 One Dimensional Simple Random Walks

The *simple random walk*, or *drunkards walk*, is a fundamental system in probability theory, with applications such as gambling and electric networks. The simple random walk is particularly important because it forms the basis upon which more complicated walks can be later defined. This section will demonstrate some key theorems and properties about the simple random walk, which will then be validated through numerical simulations in the Results & Simulations section. Key information that will be covered is the expected position of the walk, the distribution that governs this, and how this distribution evolves in the limit. The following concepts draw upon the works of many authors [8–14].

To construct the simple random walk in one dimension, let $X_1, X_2, \ldots, X_n$ denote independent and identically distributed random variables *s.t.*

$$X_i = \begin{cases} +1 & \text{with probability } p \\ -1 & \text{with probability } q \end{cases}$$

With $p \in (0, 1)$ and $q = 1 - p$. Let S_n denote the sum

$$S_n = \sum_{i=1}^{n} X_i \quad .$$

The random variables X_i represent individual steps, and S_n represents the position of the walker after exactly n steps.

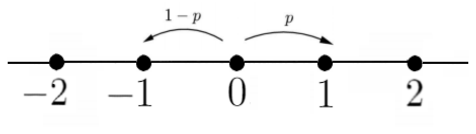


Figure 1: Simple random walk on $\mathbb{Z}$

For simplicity, let the step-size of the random walker be 1, and let the origin be $S_0 = 0$.

The first question one might ask is *what is the expected location of the simple random walk after n steps?*

Theorem 2.1. $\mathbb{E}[S_n] = n \cdot (p - q)$

Proof. An elementary result from probability theory states that if $X_1, X_2, \ldots, X_n$ are random variables then

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Therefore,

$$\mathbb{E}[S_n] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = n \cdot \mathbb{E}[X_1]$$

Where the last equality is a result of $X_1, X_2, \ldots, X_n$ being identically distributed. This gives

$$\mathbb{E}[S_n] = n \cdot [(1) \cdot p + (-1) \cdot q] = n \cdot (p - q)$$

□

Remark. If $p = q = 1/2$, then we have that $\mathbb{E}[S_n] = 0$, i.e. the expected position of the walker will be at the origin for any number of steps n. This kind of simple random walk is called a *symmetric random walk.*

Let us focus on the symmetric case. The probability distribution of the expected position is centered at 0, but that does not imply that the walker is always (or often) at 0, quite the contrary. As the number of steps increase, the wider the probability distribution will be. For further insight into this, take the mean-squared displacement,

$$S_n^2 = \left(\sum_{i=1}^{n} X_i\right)^2 = \sum_{i=1}^{n} X_i \sum_{j=1}^{n} X_j = \sum_{i=1}^{n} X_i^2 + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} X_i X_j \tag{1}$$

Considering the final sum in Eq. (1), for $i \neq j$, we have

$$X_i X_j = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}$$

Which is a analogous to just another symmetric random walk! Thus applying Theorem (2.1), we have

$$\mathbb{E}\left[\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} X_i X_j\right] = 0 \quad . \tag{2}$$

Additionally, $X_i^2 = 1$ for $i = 1, 2, \ldots, n$. Therefore, with Eq.(2) and the linearity of expectation,

$$\mathbb{E}[S_n^2] = \mathbb{E}\left[\sum_{i=1}^{n} X_i^2 + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} X_i X_j\right] = n + \mathbb{E}\left[\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} X_i X_j\right] = n$$

Implying that the expected distance from the origin is increasing with n. In general, the 'speed' at which a random walk travels away from the origin is given by the parameter ν in the following equation

$$\mathbb{E}[S_n^2] \propto n^{2\nu} \tag{3}$$

Thus for the simple symmetric random walk we have demonstrated that $\nu = 1/2$. In Eq. (3), ν is called a critical exponent. A thorough explanation of the critical exponents will be given in a later section dedicated to self-avoiding walks. The above derivation for the mean squared distance from the origin serves as preliminary insight for the following theorem, which will be proved following [39–41].

Theorem 2.2. $\mathbb{E}\left[|S_n|\right] \sim \sqrt{\frac{2n}{\pi}}$

Proof. First, we require the probability of arriving at a particular final position S_n after n steps, $P(S_n)$. Given n Bernoulli trials with probability p of success and probability $1 - p$ of failure, the probability of k successes is

$$\binom{n}{k} p^k (1-p)^{n-k} \quad . \tag{4}$$

To arrive at S_n, the walker must take exactly $\frac{n+S_n}{2}$ steps to the right and $\frac{n-S_n}{2}$ steps to the left. Then utilising Eq. (4) and setting $p = \frac{1}{2}$ gives

$$P(S_n) = \frac{1}{2^n} \binom{n}{\frac{S_n+n}{2}} \quad . \tag{5}$$

Which is best visualised by the following table, which closely resembles Pascal's triangle [13].

$n \setminus \mathrm{S}_n$	-4	-3	-2	-1	0	1	2	3	4
0					1				
1				$\frac{1}{2}$	0	$\frac{1}{2}$			
2			$\frac{1}{4}$	0	$\frac{2}{4}$	0	$\frac{1}{4}$		
3		$\frac{1}{8}$	0	$\frac{3}{8}$	0	$\frac{3}{8}$	0	$\frac{1}{8}$	
4	$\frac{1}{16}$	0	$\frac{4}{6}$	0	$\frac{6}{16}$	0	$\frac{4}{16}$	0	$\frac{1}{16}$

Table 1: Probability table for different final positions (S_n) of a symmetric simple random walk over a range of number of steps n

Note that if n is even, arriving at an odd final position S_n is impossible, and vice versa. By definition,

$$\mathbb{E}[|S_n|] = \sum_{S_n=-n,-(n-2),\ldots}^{n} |S_n| \cdot P(S_n) \tag{6}$$

Subbing in Eq. (5) and expanding,

$$= \frac{1}{2^n} \sum_{S_n=-n,-(n-2),\ldots}^{n} \frac{|S_n| n!}{\left(\frac{n+S_n}{2}\right)! \left(\frac{n-S_n}{2}\right)!} \tag{7}$$

For clarity, it is important to note that n is fixed, and the sum iterates over all the possible final positions S_n, from $-n$ to n. To evaluate this sum, we split it into two cases, when n is even, and when n is odd. First, let $n = 2p$ where $p \in \mathbb{N}$.

$$\begin{aligned}
\mathbb{E}[|S_n|] &= \frac{n!}{2^n}\left[\sum_{S_n=-2p,-2p-2,\ldots}^{-2} \frac{|S_n|}{\left(\frac{2p+S_n}{2}\right)! \left(\frac{2p-S_n}{2}\right)!} + 0 + \sum_{S_n=2,4,\ldots}^{2p} \frac{|S_n|}{\left(\frac{2p+S_n}{2}\right)! \left(\frac{2p-S_n}{2}\right)!}\right] \\
&= \frac{n!}{2^n}\left[\sum_{S_n=-p,-(p-1),\ldots}^{-1} \frac{|2S_n|}{\left(\frac{2p+2S_n}{2}\right)! \left(\frac{2p-2S_n}{2}\right)!} + \sum_{S_n=1,2,\ldots}^{p} \frac{|2S_n|}{\left(\frac{2p+2S_n}{2}\right)! \left(\frac{2p-2S_n}{2}\right)!}\right] \\
&= \frac{n!}{2^n}\left[2\sum_{S_n=1}^{p} \frac{2S_n}{(p+S_n)!(p-S_n)!}\right] \\
&= \frac{n!}{2^{n-2}} \sum_{S_n=1}^{p} \frac{S_n}{(p+S_n)!(p-S_n)!}
\end{aligned}$$

The sum can be solved analytically to give

$$\sum_{S_n=1}^{p} \frac{S_n}{(p+S_n)!(p-S_n)!} = \frac{1}{2\Gamma(p)\Gamma(1+p)} \tag{8}$$

Where $\Gamma(x)$ is the standard Gamma function defined by

$$\Gamma(n) = (n-1)! \ \ . \tag{9}$$

An important property of the Gamma function is that if n is a non-negative integer, then [42]

$$\Gamma\left(\frac{1}{2}+n\right) = \frac{(2n)!}{4^n n!}\sqrt{\pi} = \frac{(2n-1)!!}{2^n}\sqrt{\pi} \tag{10}$$

Using this property and subbing $p = n/2$ into Eq. (8) and simplifying,

$$\mathbb{E}[|S_n|] = \frac{2}{\sqrt{\pi}} \frac{\Gamma\left(\frac{1}{2}+\frac{1}{2}n\right)}{\Gamma\left(\frac{1}{2}n\right)} = \frac{(n-1)!!}{(n-2)!!} \ \ . \tag{11}$$

Now let's consider the case where n is odd. Let $n = 2p-1$,

$$\begin{aligned}
\mathbb{E}[|S_n|] &= \frac{n!}{2^n}\left[\sum_{S_n=-(2p-1),-(2p-3),\dots}^{-1} \frac{|S_n|}{\left(\frac{2p-1+S_n}{2}\right)!\left(\frac{2p-1-S_n}{2}\right)!} + \sum_{S_n=1,3,\dots}^{2p-1} \frac{|S_n|}{\left(\frac{2p-1+S_n}{2}\right)!\left(\frac{2p-1-S_n}{2}\right)!}\right] \\
&= \frac{n!}{2^n}\left[\sum_{S_n=1,3,\dots}^{2p-1} \frac{2S_n}{\left(\frac{2p-1+S_n}{2}\right)!\left(\frac{2p-1-S_n}{2}\right)!}\right] \\
&= \frac{n!}{2^{n-1}}\left[\sum_{S_n=2,4,\dots}^{2p} \frac{S_n-1}{\left(\frac{2p-2+S_n}{2}\right)!\left(\frac{2p-S_n}{2}\right)!}\right] \\
&= \frac{n!}{2^{n-1}}\left[\sum_{S_n=1}^{p} \frac{2S_n-1}{(p+S_n-1)!(p-S_n)!}\right]
\end{aligned}$$

The sum can be solved analytically to give

$$\sum_{S_n=1}^{p} \frac{2S_n-1}{(p+S_n-1)!(p-S_n)!} = \frac{1}{\Gamma(p)^2} \tag{12}$$

As before, using the property in Eq. (10), subbing in $p = (n+1)/2$ and simplifying gives

$$\mathbb{E}[|S_n|] = \frac{n!}{2^{n-1}\left[\Gamma\left(\frac{1}{2}+\frac{1}{2}n\right)\right]^2} = \frac{2}{\sqrt{\pi}} \frac{\Gamma\left(\frac{1}{2}n+1\right)}{\Gamma\left(\frac{1}{2}n+\frac{1}{2}\right)} = \frac{n!!}{(n-1)!!} \ \ . \tag{13}$$

The next trick is realising that both the odd and even case can be expressed by the same equation of p,

$$\mathbb{E}[|S_p|] = \frac{2}{\sqrt{\pi}} \frac{\Gamma\left(p+\frac{1}{2}\right)}{\Gamma(p)} = \frac{(2p-1)!!}{(2p-2)!!} \tag{14}$$

Now we can use the asymptotic expansion of the Gamma function [43], applied to this ratio to get

$$\frac{\Gamma\left(p+\frac{1}{2}\right)}{\Gamma(p)} = \sqrt{p}\left(1 - \frac{1}{8p} + \frac{1}{128p^2} + \dots\right) \tag{15}$$

Subbing this into Eq. (14) and replacing p for n, we get

$$\mathbb{E}[|S_n|] = \sqrt{\frac{2n}{\pi}}\left(1 \mp \frac{1}{4n} + \frac{1}{32n^2} \pm \frac{5}{128n^3} - \frac{21}{2048n^4} \mp \dots\right) \tag{16}$$

where the top signs represents the case where n is even and the bottom signs represent when n is odd. Therefore, for large n we obtain the desired result

$$\mathbb{E}\left[|S_n|\right] \sim \sqrt{\frac{2n}{\pi}} \quad . \tag{17}$$

□

This now gives a good idea of how far to expect the random walk to be from the origin, which is extremely useful to know in practical settings. For example, in Finance, it is clearly useful to investors to know the expected change of a valuable stock price over a given time period.

To gain a deeper understanding of the distribution of S_n, consider its PDF when n is very large. Recall the Central Limit Theorem (CLT):

Theorem 2.3. (Central Limit Theorem). Given an i.i.d. sequence $(X_n, n \geq 1)$ with $\mathbb{E}\left[X_i\right] = \mu$, $\operatorname{var}\left[X_i\right] = \sigma^2$. For every constant a

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{\sum_{1\leq i\leq n} X_i - \mu n}{\sigma\sqrt{n}} \leq a\right) = \int_{-\infty}^{a} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$$

Thus a sequence of i.i.d. random variables will converge in distribution to the normal distribution as $n \to \infty$. Therefore, as the steps X_i are i.i.d., the Central Limit Theorem can be directly applied to prove that the PDF of S_n in a long walk tends to a particular Gaussian-like distribution

$$P(S_n = x) \approx \frac{2}{\sqrt{2n\pi}} e^{\frac{-x^2}{2n}} \quad . \tag{18}$$

For large n Eq. (18) can be used to approximate $P(S_n)$, which will be demonstrated numerically in the Results & Simulations section. The step-by-step derivation of Eq. (18) can be found in [44].

2.1.2 Two Dimensional Simple Random Walks

The simple random walk in two dimensions will be considered in this section predominantly on finite graphs. The simple random walk on $\mathbb{Z}^2$ will be briefly considered, before moving on to finite subsets of $\mathbb{Z}^2$. The application to modelling diffusion via Dirichlet problems will be established, along with two methods of finding solutions algorithmically. This will serve as preliminary theory for a numerical comparison of these methods in the Results & Simulations section.

The simple random walk on $\mathbb{Z}^2$ has four choices at every vertex, the $\pm\, x$ direction, and the $\pm\, y$ direction. This process is analogous to two simple random walks on $\mathbb{Z}$ where each step you flip a coin to determine which of the two walks takes the next step. Thus the y co-ordinate follows the same behavior as the one dimensional case with k steps, and the x co-ordinate follows the same behaviour of a one-dimensional random walk of $(n-k)$ steps, where the number of steps in the y direction, k, follows a binomial distribution with $p = 0.5$. Therefore, characteristics on $\mathbb{Z}^2$ are generally similar to the one-dimensional case, and thus will not be discussed at length. For example, the expectation of the distance from the origin can be calculated directly from Theorem (2.2) along with the expectation for the number of successes of a symmetric binomial distribution (which is $\frac{n}{2}$). This gives $\mathbb{E}\left[|S_n|\right] \sim 2\sqrt{\frac{n}{\pi}}$.

The simple random walk in two dimensions can also be constructed on $\mathbb{R}^2$ such that the direction of the steps are sampled from the uniform distribution on $[0, 2\pi)$. This type of random walk is called the Pearson-Rayleigh random walk and certainly deserves a mention in this paper at the least. For an introduction into the Pearson-Rayleigh random walk and its properties, see [45].

The simple random walk on a finite subset of $\mathbb{Z}^2$ is particularly useful in modelling temperature gradients on finite domains [8]. To begin, take a finite subset of $\mathbb{Z}^2$ and set boundary points as 1 or 0, as in the Figure (2) below.

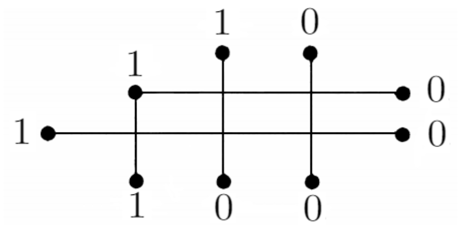


Figure 2: Finite 2D subset of $\mathbb{Z}^2$ with binary boundary points.

Then take a simple random walker starting in the interior, taking steps until it hits one of the absorbing boundary points. The expected value of the random walk's boundary point once it has finished is equivalent to the probability of it finishing at a 1, denoting a win. As the boundary points are known, solving for this probability is a Dirichlet problem, analogous to the diffusion of temperature [8]. The sought-after probabilities $p(x, y)$ can be shown to be a unique harmonic solution to the Dirichlet problem, this proof uses the maximum principle and can be found in [8]. To calculate $p(x, y)$, there are two numerical methods, the Monte-Carlo Method and the Method of Relaxation [8].

2.1.2.1 The Monte-Carlo Method

The Monte-Carlo method comprises of simulating a large number of simple random walks, beginning at the different interior points, and estimating $p(x, y)$ as the ratio of successful random walks to the total number of random walks. This method is guaranteed to converge to the true solution due to the Law of Large Numbers. This method requires a large number of simulations and is thus expected to be the less efficient method.

2.1.2.2 The Method of Relaxation

The next method of approximating $p(x, y)$ is the Method of Relaxation. This method takes advantage of the fact that our solution is harmonic, and thus has the averaging property:

$$p(x, y) = \frac{p(x+1, y) + p(x-1, y) + p(x, y+1) + p(x, y-1)}{4} \quad . \tag{19}$$

The algorithm takes a point in the interior, next to the boundary, and replaces its value for the mean of the four points adjacent to it. The next step is to chose a neighbouring point and do the same process, until all of the interior points have been 'averaged' exactly once. After the first iteration, the solution will not be necessarily harmonic, but after many iterations, it approximately demonstrates the averaging property and is consequently a good approximation for $p(x, y)$. The Method of Relaxation and the Monte-Carlo method will be numerically tested and compared with regards to efficiency and reliability. For more details on these methods, see [8].

2.1.3 Levy Flight

The Levy flight is a class of random walk that exhibits a heavy-tailed distribution for the step-size, thus allowing for large spontaneous jumps to occur on a connected domain such as $\mathbb{R}^2$ and $\mathbb{R}^3$. Figure (3) demonstrates typical paths of two Levy flights on $\mathbb{R}^2$.

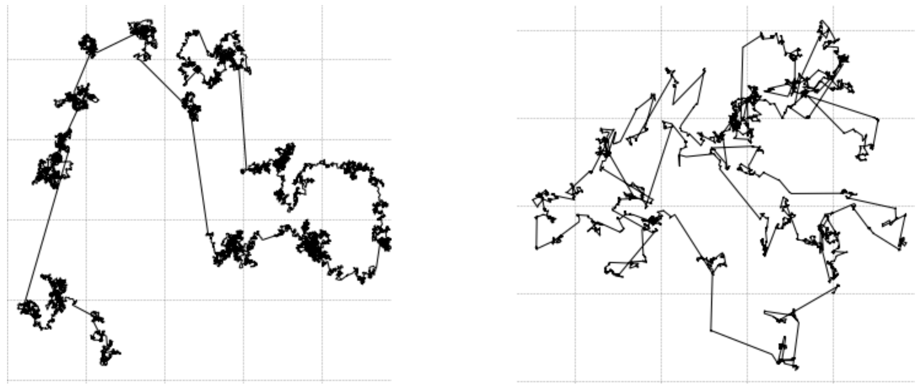


Figure 3: Levy flight paths on $\mathbb{R}^2$ with Cauchy distribution (left) and Levy-Smirnov distribution (right)

Levy flights are applicable to many interesting fields, for example, the spread of disease in a society can be accurately modelled, with infection via air travel representing the characteristic large jumps. A key application of the Levy flight is modelling the movement of foraging animals. The *Levy Flight Foraging Hypothesis* states that natural selection has caused Levy flight behaviour in animals due to its efficiency at search optimisation. This hypothesis will be tested numerically in the Results & Simulations section. The theory in this section will primarily draw upon the works of Dubkov [46] and Chechkin [18]. A Levy stable PDF is defined by its characteristic function i.e its Fourier transform

$$\begin{aligned} p_{\alpha,\beta}(k;\mu,\sigma) = \mathcal{F}\left\{p_{\alpha,\beta}(x;\mu,\sigma)\right\} &= \int_{-\infty}^{\infty} e^{ikx} p_{\alpha,\beta}(x;\mu,\sigma)dx \\ &= \exp\left[i\mu k - \sigma^{\alpha}|k|^{\alpha}\left(1 - i\beta\frac{k}{|k|}\omega(k,\alpha)\right)\right] \end{aligned}$$

where

$$\omega(k,\alpha) = \begin{cases} \tan(\frac{\pi\alpha}{2}) \quad , & \text{if} \quad \alpha \neq 1, \quad 0 < \alpha < 2 \\ -\frac{2}{\pi}\ln|k|, & \text{if} \quad \alpha = 1 \end{cases}$$

The PDF is dependent on four parameters, $\alpha, \beta, \mu, \sigma$. $\alpha \in [0,2]$ is the Levy index, a measure of how heavy-tailed the distribution is. $\beta \in [-1,1]$ is the skewness parameter, $\mu \in \mathbb{R}$ is the shift parameter, and $\sigma > 0$ is the scale parameter. The Levy index α and the skewness parameter β are the primary parameters for the PDF, the former determines the asymptotic decay of the distribution, whilst the latter determines the asymmetry of the distribution. Both μ and σ can be removed with the correct shift and scale transformations [18]

$$p_{\alpha,\beta}(x;\mu,\sigma) = p_{\alpha,\beta}\left(\frac{x-\mu}{\sigma};0,1\right)$$

For the remainder of this section, let the PDF be denoted by $p_{\alpha,\beta}(x)$. There are only three cases where the PDF can be written in terms of elementary functions, and thus these cases will be treated with additional importance in this section. First, we have the Gaussian distribution where $\alpha = 2$ and β arbitrary

$$p_2(x) = \frac{1}{\sqrt{4\pi}} \exp\left(-\frac{x^2}{4}\right) \tag{20}$$

Secondly we have the Cauchy distribution, where $\alpha = 1$ and $\beta = 0$

$$p_{1,0}(x) = \frac{1}{\pi\left(1 + x^2\right)} \tag{21}$$

and lastly the Lévy-Smirnov distribution, with $\alpha = \frac{1}{2}$ and $\beta = 1$

$$p_{1/2,1}(x) = \begin{cases} \frac{1}{\sqrt{2\pi}} x^{-3/2} \exp\left(-\frac{1}{2x}\right), & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{22}$$

Note. The normalisation constants are subject to change when implementing the distributions numerically, due to constraints on the domain.

The Levy stable distributions are of particular interest as they demonstrate clear asymptotic behavior. For $\beta = 0$, the Levy stable distributions are symmetric for all α. Taking an arbitrary symmetric Levy stable distribution, as x becomes large we have [47]

$$p_{\alpha,0}(x) \approx \frac{C_\alpha}{|x|^{1+\alpha}} \tag{23}$$

where

$$C_\alpha = \frac{1}{\pi} \sin(\frac{\pi\alpha}{2})\Gamma(1+\alpha). \tag{24}$$

Further to this, when $0 \leq \alpha < 2$, the variance diverges as $\mathbb{E}[x^2] = \infty$, and with $\alpha = 2$ (the Gaussian distribution), the variance is finite [46]. When $\beta \neq 0$, the distribution is named *asymmetric*, and with $\beta = \pm 1$ the distribution is called *extremal*. With the Levy-Smirnov distribution being the only exception, the asymmetric and extremal cases will be left untreated. For further reading of these cases, see [18].

Another interesting heavy-tailed distribution also used for Levy flight simulation is the Pareto distribution, defined as

$$p_{\alpha,x_0}(x) = \begin{cases} \frac{\alpha x_0^\alpha}{x^{\alpha+1}} & \text{if } x \geq x_0 \\ 0 & \text{if } x < x_0 \end{cases} \tag{25}$$

Here $\alpha \in [0, 2]$ is similar to before, defining the rate at which the distribution tends to 0, and x_0 is simply the minimum possible step-size.

Remark. The non-zero component of the Pareto distribution with a scale and shift transformation closely resembles the asymptotic behavior described in Eq. (23) and thus providing insight into its appropriateness as a Levy flight distribution.

Through an extensive analysis of literature, the Levy flight is consistently defined by either a Levy stable distribution(e.g. Gaussian/Cauchy/Levy-Smirnov) or the Pareto distribution [18, 45, 46]. Therefore, these distributions will form the basis of the analysis in the Results & Simulations section. Below is a table with a quick summary of these four representative distributions.

	Distribution	**Definition**
Levy-Stable	Gaussian	$\frac{1}{\sqrt{4\pi}} \exp\left(\frac{-x^2}{4}\right)$
	Cauchy	$\frac{1}{\pi(1+x^2)}$
	Levy-Smirnov	$\begin{cases} \frac{1}{\sqrt{2\pi}} x^{-3/2} \exp\left(-\frac{1}{2x}\right), & x \geq 0 \\ 0, & x < 0 \end{cases}$
Discontinuous	Pareto	$\begin{cases} \frac{\alpha x_0^\alpha}{x^{\alpha+1}}, & x \geq x_0 \\ 0, & x < x_0 \end{cases}$

Table 2: Four Levy flight distributions and their respective definitions

Later in this paper, the corresponding Levy flights to these distributions will be simulated and analysed with regards to optimal search efficiency, and thus providing evidence for the Levy Flight Foraging Hypothesis.

2.2 Non-Markovian Random Walks

2.2.1 Reinforced Random Walks

The first Non-Markovian random walk to be discussed is the reinforced random walk (RRW). This type of walk is widely applied to probability theory, it accurately models many systems such as *Polya's urn*, which will be discussed in this section [19]. The reinforced random walk is defined such that the probability of traversing edges is dependent on the number of times it has been traversed in the past. For example, a particular RRW may have a slight bias towards edges it has already traversed, considering it a 'safer' option. This section will discuss the main types of RRW, along with key applications and properties. For a deeper look into reinforced random walks, see [19].

Let the path of a reinforced random walk on graph G up to a time t be denoted by a sequence of vertices $x_0, x_1, \ldots, x_t$ where each x_i is a neighbour of x_{i+1}, denoted $x_i \sim x_{i+1}$. Define the number of traversals of an edge e up to time t as $N(e,t)$, specifically

$$N(e,t) = \#\{s : 1 \leq s \leq t, (x_{s-1}, x_s) = e\} \tag{26}$$

Let $f : \{0, 1, 2, \ldots\} \to (0, \infty)$ be a function denoting the reinforcement function of the walk, then the transition probabilities are given by

$$\mathbb{P}(x_{t+1} = x | x_0, \ldots, x_t) = \frac{f(N((x, x_t), t))}{\sum_{y \sim x_t} f(N((y, x_t), t))} \tag{27}$$

Where the denominator is simply a normalisation constant. The reinforcement function f determines the behaviour of the RRW, the key types of RRW are as follows:

(1) Once-reinforced random walks are defined by

$$f_a(n) = \begin{cases} 1 & n = 0 \\ 1 + a & n \geq 1 \end{cases} \tag{28}$$

Thus, edges which have already been traversed are given a $(1+a)$ times greater weight when compared to untraversed edges, but the exact number of traversals makes no difference.

(2) Linearly reinforced random walks are defined by $f(n) = 1 + n$. Therefore, whenever the walker traverses an edge, its weight increases by 1.

(3) Strongly reinforced random walks are the class of RRW where f grows faster than linear. For example, $f(n) = 1 + n^2$.

These walks all have in common that subsequent traversals can only increase the likelihood of future traversals. With the superlinear case (3), the walker is likely to eventually get stuck on a single edge, forever boosting its weight and making the relative weights of other neighbouring edges tend to 0. To better understand this, let's prove the claim

Theorem 2.4. Let G be an arbitrary graph, x_0 a vertex of G and e an edge connected to x_0. Take a strongly reinforced random walk on G starting from x_0 with reinforcement function $1 + n^2$. Then with positive probability the walker gets stuck traversing e indefinitely i.e. for all $t, x_{2t} = x_0$ and x_{2t+1} is the other vertex connected to e.

Proof. Let y be the other vertex connected by e. Let E_s denote the event that the walker has been stuck on e for all $t \leq s$. Now, let's calculate the probability of E_{s+1} given E_s. If E_s is true, then $N(e,s) = s$, and $N(\tilde{e}, s) = 0$ for any $\tilde{e} \neq e$. Therefore,

$$\mathbb{P}(E_{s+1} | E_s) = \frac{f(s)}{f(s) + (\deg x_s - 1) f(0)} = \frac{1 + s^2}{1 + s^2 + (\deg x_s - 1)} \geq 1 - \frac{C}{s^2} \tag{29}$$

for some constant C given by the degrees of x_0 and y. For E_s to be true, E_t must also be true for all $t \leq s$, thus using Eq. (29)

$$\mathbb{P}\left(E_s\right) = \prod_{t=1}^{s} \mathbb{P}\left(E_t | E_{t-1}\right) \geq \prod_{t=1}^{s} \left(1 - \left(\frac{C_t}{t^2}\right)\right) \geq c > 0$$

□

The most significant application of reinforced random walks is in probability theory, and to gain an insight into why this is, this next part of the section will discuss its application to Polya's Urn, which is a particularly nice example. Polya's Urn is an urn that contains white balls and black balls, and one at a time, you take a ball out, inspect the colour, and then put two balls back in of that same colour. As this process takes place, one colour will inevitably become dominant, and this slight edge will keep growing indefinitely. Now take a linearly reinforced random walk with reinforcement function $f(n) = 2 + n$ and define G as 3 vertices in a line.

Figure 4: Simple 3-vertex graph analogous to Polya's Urn

Then this is analogous to Polya's Urn. When the RRW begins at the centre of G, it is equally likely to go left or right, representing the equal probability of choosing a black or white ball. Once the walker chooses left or right, its next step is guaranteed to be going back to the centre. Thus the weight on this edge has increases exactly by 2, however, the number of balls has only increased by 1. Therefore the weights of the edges in G correspond exactly to 2 times the number of black and white balls. Thus a simple division by 2 gives you the number of black and white balls in the urn at any time t. Furthermore, a linearly reinforced random walk on $\mathbb{Z}$ is analogous to a sequence of i.i.d Polya Urns, and when G is a tree, you can couple the RRW to a sequence of multicoloured Polya Urns [19]!

In the Results & Simulations section, this paper will introduce a generalisation to the set of reinforced random walk, allowing the reinforcement function f to take further edges as inputs, and thus allowing for a stronger analysis.

2.2.2 Self-Avoiding Walks

The *self-avoiding walk* (SAW) is a random walk on a graph or lattice with the strict condition that a site, once visited, can never be revisited. Much research has been conducted on SAWs on particular lattices that appear in physics and chemistry, such as the modelling of long-chain polymers on the honeycomb lattice. The most prominent question, is how many n step SAWs are there? The answer to this simple question has no exact formula, and it is generally accepted that the difficulty of this problem is so immense, that it cannot be solved by current theory and methods [24].

The asymptotic characteristics of SAWs are slightly less difficult to analyse, with well established coefficients to quantify their behavior, depending on the lattice. These coefficients are called the *critical exponents*. The critical exponents quantify the behavior in the following equations:

$$c_n \sim A\mu^n n^{\gamma-1} \tag{30}$$

and

$$\mathbb{E}[S_n^2] \sim Bn^{2\nu} \tag{31}$$

Where A, B are positive constants, and μ, γ, ν are critical exponents. The constant ν determines the rate at which the random walk moves away from the origin, and is strictly dependent on the lattice. It is conjectured that $\nu = 3/4$ in two dimensions. Recall that for the simple random walk, $\nu = 1/2$, this implies that a SAW travels quicker than a simple random walk, which is intuitive because a SAW cannot double back on itself. γ, is believed to be *universal* i.e. it is independent of the lattice. γ not only characterises the long-term behavior of c_n, but it is a measure of the probability that two n step SAWs starting at the same place are to intersect[24].

The most sought-after and intriguing of these coefficients is the *connective constant*, denoted μ. The connective constant can be thought of as the average number of possible edges the walker can traverse at each step of a long walk, and has a direct relationship to the number of n step self-avoiding walks c_n. For example, when the self-avoiding walk on the one dimensional line $\mathbb{Z}$ takes a step in one direction, it is now fixed to that direction indefinitely, and thus only ever has one choice at any time, indicating that $\mu = 1$. The connective constant is unknown in almost all non-trivial lattices, with the only exception being the honeycomb lattice, where $\mu = \sqrt{2+\sqrt{2}}$ [48]. This section will primarily discuss the connective constant, proving its existence, constructing approximations, and formulating the methods for upper and lower bounds.

Let c_n be the number of n step self-avoiding walks, then the connective constant is defined as follows,

Definition 2.2.

$$\mu = \lim_{n\to\infty} c_n^{1/n} \tag{32}$$

To prove that this limit exists, first notice that any SAW can be broken down into two smaller SAWs - if the parent walk is self-avoiding, then its sub-walks are certainly too. However, the converse is not always true, you cannot pair together any two SAWs and guarantee that the resulting walk is also a SAW. This leads to the following property

$$c_{n+m} \le c_n \cdot c_m \quad \text{for } n, m \in \mathbb{N} \tag{33}$$

Taking the logarithm of both sides demonstrates that the sequence $\{\log c_n\}$ is *subadditive*:

$$\log c_{n+m} \le \log c_n + \log c_m \tag{34}$$

The existence of the limit (32) is dependant upon this property, which will become clear in the proof of the following lemma:

Lemma 2.5 (Fekete's Lemma)**.** Let $\{a_n\}_{n\ge1}$ be a sequence of real numbers which is *subadditive*, i.e., $a_{n+m} \le a_n + a_m$. Then the limit $\lim_{n\to\infty} n^{-1}a_n$ exists and is given by

$$\lim_{n\to\infty} \frac{a_n}{n} = \inf_{n\ge1} \frac{a_n}{n}$$

Proof. Let L be defined as

$$L = \inf_{n\ge1} \frac{a_n}{n}$$

Let $\epsilon > 0$, then by the definition of infimum $\exists\, n$ s.t.$|\frac{a_n}{n} - L| < \epsilon$. This directly implies $a_n < n(L+\epsilon)$. Let $b = \max_{1\le i<n} a_i$. Take $m \ge n$, let $m = qn + r$ with $0 \le r < n$. Using the subadditivity property,

$$a_m = a_{nq+r} = a_{n+n+\cdots+n+r} \le \underbrace{a_n + a_n + \cdots + a_n}_{q \text{ times}} + a_r \le qa_n + b$$

Thus

$$\begin{aligned}\frac{a_m}{m} &\le \frac{qa_n}{m} + \frac{b}{m}\\ &< \frac{qn(L+\epsilon)}{m} + \frac{b}{m}\\ &\to L+\epsilon \text{ as } m\to\infty\end{aligned}$$

since $qn/m \to 1$ as $m \to \infty$. As ϵ was taken arbitrarily, we have

$$\lim_{n\to\infty} \frac{a_n}{n} = L = \inf_{n\ge1} \frac{a_n}{n} \ .$$

□

Therefore, applying Lemma (2.5), the connective constant exists as the limit of the sequence $\{c_n^{1/n}\}$. The approximations first calculated for μ were by using computers to generate terms in this sequence. Due to the exponential nature of c_n, the enumeration of SAWs is certainly a challenge, and approximations of μ converge very slowly.

The solution is to find upper and lower bounds for μ, confining the interval to smaller and smaller lengths. This is much more effective than brute force enumeration.

2.2.2.1 Lower Bounds for μ

There are several ways of constructing lower bounds for the connective constant μ, both analytical and numerical. The most straightforward method, is by finding subsets of SAWs. For example, take the walks that can only move in one direction on the square lattice $\mathbb{Z}^2$, with d_n representing the number of such n step walks. Then we have $d_n = 4 \leq c_n$ for all $n > 1$, which gives $\mu = \lim_{n\to\infty} c_n^{1/n} \geq \lim_{n\to\infty} d_n^{1/n} = \lim_{n\to\infty} 4^{1/n} = 1$. This set of unidirectional walks doesn't come close to the full set of SAWs, and thus only a weak lower bound; $\mu \geq 1$ is found, but this can be improved significantly.

A better subset is proposed in [49], where the SAW can no longer move in the $-x$ direction, but only in the $\pm y$ direction and the $+x$ direction. Such a walk is guaranteed to be self-avoiding as long as it doesn't make immediate reversals. Given the n step walks of this type, let the total number of walks ending in a $+x$ step denote a_n, and the total number of walks ending in a $\pm y$ denote b_n. Note that if the last step the walker took was a $+x$ step, then its next step can be either $+x$, $+y$, or $-y$. Whereas if the last step was a $\pm y$, then it's next step must either be a $+x$ or a $\mp y$. This leads to the following equations;

$$\begin{aligned} a_{n+1} &= a_n + b_n, \\ b_{n+1} &= 2a_n + b_n \end{aligned} \tag{35}$$

To analyse the asymptotic behavior of this system, assume $a_n = A\lambda^n$ and $b_n = B\lambda^n$. The lower bound for μ is given by the largest root of the characteristic polynomial of

$$\begin{vmatrix} 1-\lambda & 1 \\ 2 & 1-\lambda \end{vmatrix} = 0$$

Which simplifies to $\lambda^2 - 2\lambda - 1 = 0$. Thus we obtain $\mu \geq 1+\sqrt{2} \approx 2.414$...Which is already much closer to the current approximation of $\mu \approx 2.638$... This technique of finding subsets is a good starting point, but more effective methods have been developed.

A more advanced method for constructing lower bounds is *Keston's Method of irreducible bridges.* This method was originally developed in Keston's book [50], however, the following theory will follow closer to Alm and Parviainen's paper on the same topic [26].

To apply this method, the notion of a *bridge* must be defined.

Definition 2.3. Given a fixed embedding of the lattice in $\mathbb{Z}^2$, let the coordinates for a vertex v be denoted by $(v(x), v(y))$. A bridge of length n is a self-avoiding walk such that

$$v_0(x) < v_i(x) \leq v_n(x), \text{ for } i = 1, \ldots, n-1$$

A bridge is a type of SAW with the constraint that the 'furthest to the left' point of the walk must be **only** at the origin, and the 'furthest to the right' points must be at least at the last step of the walk. The convention is that the first step for any bridge is in the $+x$ direction. Let b_n be the number of n step bridges. Keston showed that $\lim_{n\to\infty} b_n^{1/n} = \mu$. Bridges are a subset of SAWs and thus $b_n \leq c_n$ for all n. Thus $b_n^{1/n} \leq \mu$. However, Keston realised a better lower bound can be obtained by using *irreducible bridges.* An irreducible bridge is a bridge that cannot be decomposed into two smaller bridges. Let a_n denote the number of n step irreducible bridges, and denote $B(x)$ and $A(x)$ as the generating functions for the corresponding sequences. Furthermore, Keston demonstrated the relation

$$B(x) = \frac{1}{1 - A(x)} \tag{36}$$

With the result, astonishingly, that $A(x_c) = 1 \iff x_c = \mu^{-1}$. Using this relation, a lower bound can be constructed. If $0 \leq \tilde{a}_n \leq a_n$ and x_c is the solution to

$$\sum_{n=1}^{\infty} \tilde{a}_n x^n = 1, \tag{37}$$

then $x_c^{-1} \leq \mu$. In practice, if a_n is known for $n = 1, \ldots, N$, then set $\tilde{a_n} = a_n$ for $n \leq N$, and $\tilde{a_n} = 0$ for $n > N$, which clearly satisfies the condition $0 \leq \tilde{a}_n \leq a_n$. Computing a_n directly is cumbersome, but thankfully, using Eq. (36) and following Alm and Parviainen [26], the number of irreducible bridges a_n, can be found easily by the number of bridges b_n. First, let the number of bridges of length n and span l be given by $b_{n,l}$, and the number of irreducible bridges of length n and span l be given by $a_{n,l}$, with generating functions $B_l(x)$ and $A_l(x)$. The span of a bridge is how far it has traversed along the x axis, thus a bridge with span L, has the property: $v_0(x) = 0$ and $v_n(x) = L$. Clearly $\sum_{l=1}^{\infty} a_{n,l} = a_n$, and thus $\sum_{l=1}^{L} a_{n,l} \leq a_n$ for some maximum span L. Therefore, for a maximum length N and maximum span L, the reciprocal of the solution to

$$\sum_{n=1}^{N} \sum_{l=1}^{L} a_{n,l} x^n = 1 \tag{38}$$

is a lower bound to μ. A bridge is either irreducible, or the concatenation of a bridge with an irreducible bridge, this gives us that

$$B_l(x) = A_l(x) + \sum_{k=1}^{l-1} A_{l-k}(x) B_k(x) \tag{39}$$

which leads to,

$$A_l(x) = B_l(x) - \sum_{k=1}^{l-1} A_{l-k}(x) B_k(x) \tag{40}$$

Giving the generating functions $A_l(x)$ recursively for $l = 1, \ldots, L$. Keston's method is particularly effective as it can be entirely encoded, and left to a computer to find accurate lower bounds. For the implementation of Keston's method, see Section (A.3). Following this, this paper will discuss the methods of finding upper bounds.

2.2.2.2 Upper Bounds for μ

Similarly to lower bounds, an upper bound can be found by finding supersets to the set of SAWs. One such example is the set of simple random walks on $\mathbb{Z}^2$. With every new step, a simple random walk always has 4 options, this gives the upper bound of $\mu \leq 4$. Following the theory outlined in [25], *Alm's Method* will be formulated.

Two vertices of a lattice are considered to be of the same vertex class if one can be mapped on the other by a translation of the lattice whilst preserving its structure. For example, the Euclidean lattice only has one vertex class, and the honeycomb lattice has two.

This method is applicable to any *finitely generated lattice.* Such lattices have the following conditions

(i) There is a finite number, K_0, of vertex classes.

(ii) The number of outgoing edges from any vertex is finite.

(iii) Each vertex can be connected to any other by a walk

Let c_n^k denote the number of n step self avoiding walks starting at a vertex of class k, for $k = 1, \ldots, K_0$. Let C_n denote the total number of walks,

$$C_n = \sum_{k=1}^{K_0} c_n^k \tag{41}$$

Furthermore, let $\gamma_i(n)$ denote the individual SAWs of length n, where $i = 1, \ldots, C_n$. Following this, let $g_{ij}(m,n)$ with $m < n$ denote the number of n step SAWs that begin with $\gamma_i(m)$ and end with a translation of $\gamma_j(m)$. Further, define the matrix

$$G(m,n) = (g_{ij}(m,n))_{C_m \times C_m} \tag{42}$$

and let $\lambda_1(G(m,n))$ denote the largest eigenvalue of $G(m,n)$. Then finally, we have

Theorem 2.6.

$$\mu \leq \lambda_1(G(m,n))^{\frac{1}{n-m}} \tag{43}$$

Proof. Take two self-avoiding walks of length n, one starting with $\gamma_i(m)$ and ending with $\gamma_k(m)$, and the other starting with $\gamma_k(m)$ and ending with $\gamma_j(m)$. These self-avoiding walks can be joined, with an overlap of m steps (because of $\gamma_k(m)$), to form a new walk of length $2n - m = m + 2(n-m)$. This new walk may not necessarily self-avoiding. However, as all the self-avoiding walks of length $2n-m$ can be obtained in this way, we get

$$g_{ij}(m, m+2(n-m)) \leq \sum_k g_{ik}(m,n)g_{kj}(m,n) = (G^2(m,n))_{ij}$$

The same argument can be generalised to give

$$g_{ij}(m, m+r(n-m)) \leq (G^r(m,n))_{ij}$$

Using the norm $\|G\| := \sum_i \sum_j g_{ij}$,

$$\begin{aligned} \mu &= \lim_{r\to\infty} \|G(m, m+r(n-m)\|^{\frac{1}{m+r(n-m)}} \\ &\leq \lim_{r\to\infty} \|G^r(m,n)\|^{\frac{1}{m+r(n-m)}} \\ &= \lambda_1(G(m,n))^{\frac{1}{n-m}} \end{aligned}$$

as from [25] we have

$$\|G^r(m,n)\| = C(\lambda_1(1+o(1))^r$$

$\square$

The optimal choice for a given n is $m = n-1$, but the size of the matrix $G(m,n)$ is $C_m \times C_m$, thus G quickly contains more information than available in memory. Thus, it is better to choose the largest m possible for the given memory storage, and increasing n. For the implementation of this method, see Section (A.3).

2.3 Brownian Motion

Brownian motion is the continuous time analogue of the simple random walk. In this section we will discuss the basic definition of Brownian motion, as well as a construction of Brownian motion from a simple random walk [51].

Definition 2.4. The stochasic process $B = \{B(t), t \geq 0\}$ is a Brownian motion if

1. $B(0) = 0$,
2. $B(t)$ has independent increments, i.e. for all times $0 \leq t_1 \leq t_2$ the increments $B(t_2) - B(t_1)$ and $B(t_1) - B(t_0)$ are independent random variables,
3. For all $t \geq 0$ and $h > 0$, the increments $B(t+h) - B(t)$ are normally distributed with expectation zero and variance h,
4. With probability 1, the function $t \mapsto B(t)$ is continuous.

Brownian motion is the limit of the simple random walk where the step-size is getting smaller, and the 'speed' of the walk is increasing proportionally. To demonstrate this, we will first show that

$$B_n(t) := \frac{\sum_{1\leq i\leq \lfloor nt\rfloor} X_i}{\sqrt{n}} = \frac{S_{\lfloor nt\rfloor}}{\sqrt{n}} \tag{44}$$

satisfies all of the conditions (1-4) for Brownian motion when $n \to \infty$. The following lines of reasoning will develop upon Resnick's book [51].

1. $B_n(0) = 0$ by definition.

2. Let $t_1 < t_2$ be fixed and consider

   $$B_n(t_1) = \frac{\Sigma_{1\leq i\leq \lfloor nt_1\rfloor} X_i}{\sqrt{n}}$$

   and

   $$B_n(t_2) - B_n(t_1) = \frac{\sum_{\lfloor nt_1\rfloor < i\leq \lfloor nt_2\rfloor} X_i}{\sqrt{n}} \quad .$$

   The two sums contain mutually exclusive subsets of the sequence $X_1, X_2, \ldots.$ and since the sequence is i.i.d., $B_n(t_1)$ and $B_n(t_2) - B_n(t_1)$ are independent.

3. By CLT, for every $0 < h \leq t$, the distribution of

   $$\frac{\sum_{\lfloor nt\rfloor < i\leq \lfloor n(t+h)\rfloor} X_i}{\sigma\sqrt{\lfloor n(t+h)\rfloor - \lfloor nt\rfloor}}$$

   converges to the standard normal distribution as $n \to \infty$. The difference between $\lfloor nt\rfloor - \lfloor n(t+h)\rfloor$ is at most 1 and becomes negligible as $n \to \infty$, thus we write

   $$\frac{\sum_{nt<i\leq n(t+h)} X_i}{\sigma\sqrt{nh}} = \frac{B_n(t+h) - B_n(t)}{\sigma\sqrt{h}}$$

   which implies $B_n(t+h) - B_n(t)$ converges in distribution to $N\left(0, \sigma^2 h\right)$

4. Following the above reasoning, with a very small h the difference $|B_n(t+h) - B_n(t)|$ can become arbitrarily small as $n \to \infty$ with probability 1. Thus $B_n(t)$ becomes a continuous function with probability 1 as $n \to \infty$.

Therefore, $B := \lim_{n\to\infty} B_n(t)$ is a Brownian motion. For a proof that this limit exists, see [51]. This construction of Brownian motion from the simple random walk will be used directly for the Narrow Escape Problem in the Results & Simulations section.

3 Results & Simulations

This section will introduce a series of new results to the field of random walks. Each subsection is intended as an advancement forward from the Theory & Methods section, introducing new independent research. Each of the new results will be followed by a discussion on the implications to the research, and where necessary, future research projects to continue the work in this paper are also proposed.

3.1 Simple Random Walk

3.1.1 One Dimensional Simple Random Walk

From Theorem (2.2) we have $\mathbb{E}\left[|S_n|\right] \sim \sqrt{\frac{2n}{\pi}}$. In Figure (5) below we have the paths of 1000 simple random walks on $\mathbb{Z}$ up to $n = 1000$.

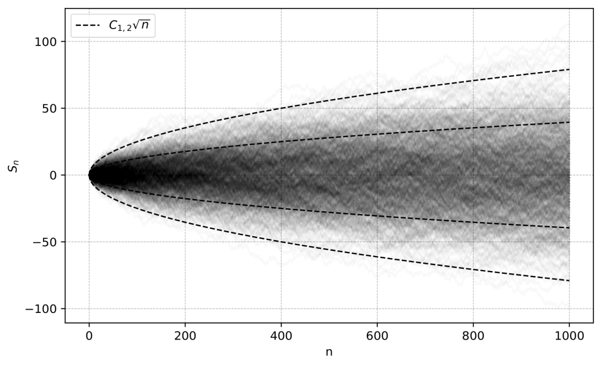


Figure 5: The positions of 1000 simulated simple symmetric random walks (S_n) vs number of steps (n), against $y = C_1\sqrt{n}$ and $y = C_2\sqrt{n}$ for $C_1 = 2.5$, $C_2 = 1.25$ to demonstrate Theorem (2.2)

This supports Theorem (2.2) as S_n is clearly travelling away from 0 at a rate proportional to $\sqrt{n}$. Additionally, S_n can be seen to be symmetrical about $x = 0$, which supports the theory for when $p = q = \frac{1}{2}$. Figure (6) demonstrates the histogram of S_n at $n = 1000$ against its Gaussian approximation:

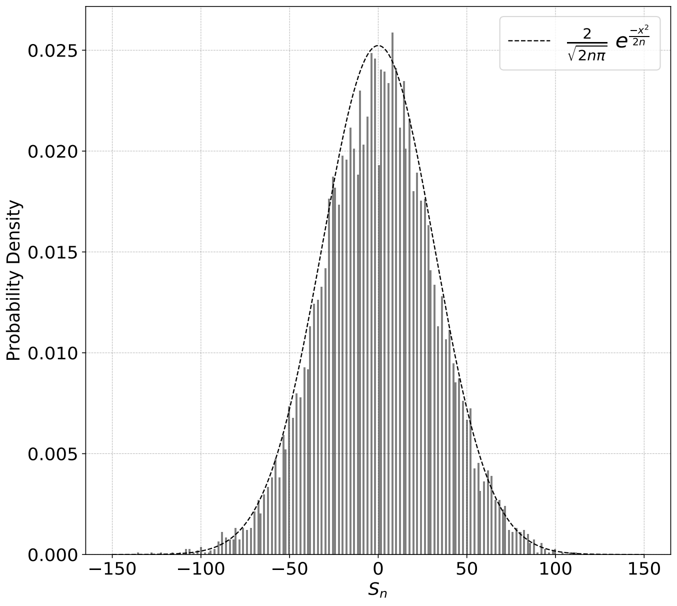


Figure 6: Histogram of the final position (S_n) of 10,000 simple symmetric random walks after 1000 steps, against a Gaussian approximation

The data lines up very closely to the Gaussian-like distribution proposed in Section (2.1.1). An interesting way to think of Figure (6) is that the curve corresponds to the cross-section of $n = 1000$ in Figure (5).

Therefore we have shown that the theoretical properties of the simple random walk on $\mathbb{Z}$ are well supported by numerical simulation.

3.1.2 Two Dimensional Simple Random Walk

To solve the 2D Dirichlet problem outlined in Section (2.1.2), there are two numerical methods to approximate the solution, the Monte-Carlo method and the Method of Relaxation.

For the Monte-Carlo method we can estimate the number of iterations n required for a given accuracy of the approximation using the CLT. Recall that the Monte-Carlo method solves the problem by simulating simple random walks until they hit a boundary point, with some of the boundary points denoting a success (1), and other boundary points denoting a failure (0). Let p be the probability of success with $q = 1 - p$, and let $\hat{p}$ be an estimator of p given by the number of successes $S(n)$ divided by the total number of iterations n. Then by CLT we have

$$\mathbf{P}\left(-a < \frac{S(n) - np}{\sqrt{npq}} < a\right) \approx \int_{-a}^{a} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \tag{45}$$

Thus for roughly a 95% certainty we set $a = 2$

$$\mathbf{P}\left(-2 < \frac{S(n) - np}{\sqrt{npq}} < 2\right) \approx 0.95 \tag{46}$$

rearraging gives

$$= \mathbf{P}\left(-2 < \frac{\hat{p} - p}{\sqrt{\frac{\bar{p}q}{n}}} < 2\right) \tag{47}$$

$$= \mathbf{P}\left(-2\frac{\sqrt{pq}}{n} < \hat{p} - p < 2\frac{\sqrt{pq}}{n}\right) \tag{48}$$

$$\leq \mathbf{P}\left(-\frac{1}{\sqrt{n}} < \hat{p} - p < \frac{1}{\sqrt{n}}\right) \gtrsim 0.95 \tag{49}$$

as $\sqrt{pq} \leq \frac{1}{2}$.

Therefore, to get an accuracy of $|\hat{p} - p| \leq 0.01$ with a confidence of 95%, $n = 10,000$ is required. To compare this against the Method of Relaxation, the number of iterations n as well as the computing time t will be calculated and compared. Two scenarios will be covered on an $N \times N$ grid; a single boundary point for success, analogous to a single flame applied to a surface, and an entire edge of boundary points for success, analogous to a heat column applied to the edge of a surface. These two scenarios are depicted in Figure (7).

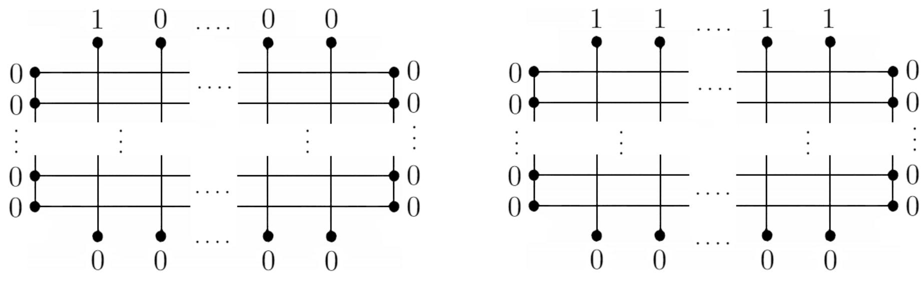


Figure 7: Finite subsets of $\mathbb{Z}^2$ with assigned binary boundary points to model the Dirichlet Problem associated with temperature diffusion - Scenario 1 (Left) & Scenario 2 (Right)

Which yields the following solutions in Figure (8), with the height and color representing the probability of success for the simple random walker.

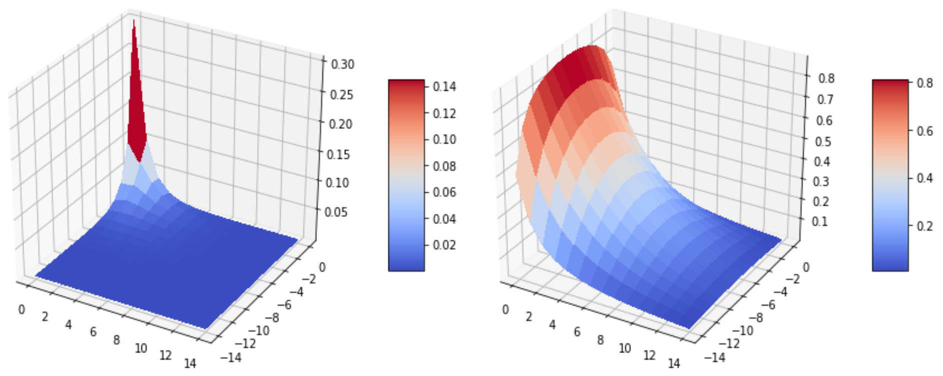


Figure 8: 2D Numerical solution of the Dirichlet Problem defined in Section (2.1.2) associated with temperature diffusion where color/height corresponds to heat - Scenario 1 (Left) & Scenario 2 (Right)

The two methods yielded the following results across the two scenarios:

Method - Scenario	Error	Iterations	Time (s)
Method of Relaxation - Scenario 1	0.000 056 6	100	0.5279
Monte-Carlo Method - Scenario 1	0.007 03	1000	72
Method of Relaxation - Scenario 2	0.000 069 8	200	0.635117
Monte-Carlo Method - Scenario 2	0.002 40	10 000	908

Table 3: Comparison of Method of Relaxation and Monte-Carlo method across two scenarios with regards to Error, Iterations, and Time

The errors were found by calculating the difference between the numerical solutions and the 'exact' solution. The exact solution was generated by running the Method of Relaxation over a huge number of iterations. Firstly, both methods successfully converged upon the correct solution displayed in Figure (8), demonstrating that the methods work as expected. The Method of Relaxation was able to achieve a significantly more accurate solution for both scenarios in a fraction of the time it took the Monte-Carlo Method. This clearly demonstrates that the Method of Relaxation was the more efficient algorithm, as expected. Additionally, the errors of the Monte-Carlo method are consistent with the error estimation derived previously in Eq. (49). This demonstrates that the numerical simulations for the simple random walk in two dimensions also agree with the theory. For details on the numerical implementation of these methods, see Section (A.1).

3.2 Levy Flight Foraging Hypothesis

The Levy Flight Foraging Hypothesis states that animals have evolved to utilise the Levy flight as a means of searching for food efficiently, as opposed to a simple random walk. The classical approach to this topic is to track the movement of foraging animals and comparing that to different models of Levy flight. The goal of this section is to provide a statistical analysis of the Levy Flight Foraging Hypothesis and consequently present some numerical evidence to support the theory. This will be achieved by simulating the foraging paths of animals when using different search strategies, and comparing their success rates. First, a direct comparison will take place between the different types of Levy flight defined in Section (2.1.3) and the simple random walk, and a t-test will demonstrate the superiority of the Levy flight as a searching strategy. Following this, an evolutionary algorithm will be formulated to simulate the process of natural selection, thus showing the evolution from simple random walks, to an optimised Levy flight search pattern.

The simulation consists of an 'animal', beginning at $(0, 0)$ on $\mathbb{R}^2$, taking steps according to a pre-determined distribution(e.g. Levy flight or simple random walk) and blindly searching for patches of 'food' in the nearby area. After a certain number of steps, if the animal has not reached any patches of food, it passes away. If the animal finds multiple patches off food, it will be rewarded for its accomplishment, but only one patch of food is required for survival. To minimize the risk of bias, the size and location of food were randomised for each individual simulation. For the technical details and implementation of this simulation, see Section (A.2).

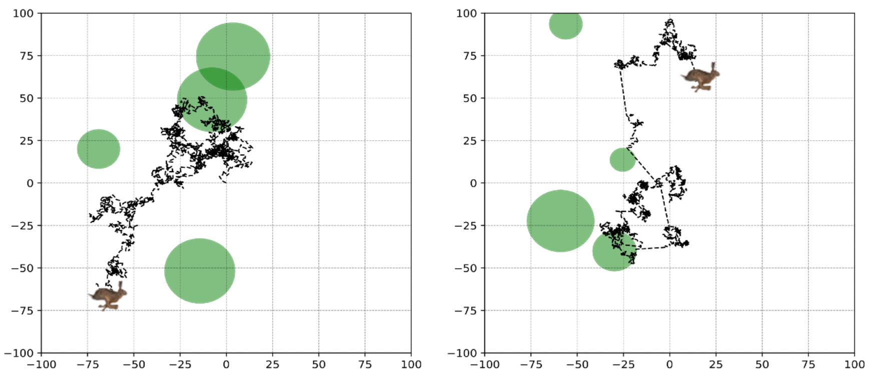


Figure 9: Simulation examples of a foraging animal determined by a simple random walk (left) and a Levy flight (right) searching for patches of food (displayed in green)

The above figure displays an example of a simple random walk and a Levy flight searching for food (green circles). The characteristic jumps of the Levy flight can be seen in the example on the right. Although it may seem intuitive that these jumps will improve the search success for the animal, we must provide a comparative numerical analysis to demonstrate it.

3.2.1 Comparison with Simple Random Walk

For a thorough comparison, the simple random walk will be tested against the four key Levy flight distributions outlined in Section (2.1.3), the three Levy-stable distributions (Gaussian, Cauchy and Levy-Smirnov) and the Pareto distribution. To estimate the survival rates of the different walkers, this paper will use a Monte-Carlo approach. For each distribution, the animal will be simulated over many iterations and the success rates will be recorded. Figure (10) shows the histogram for the simple random walk and two of the Levy flight walks' survival rates[1].

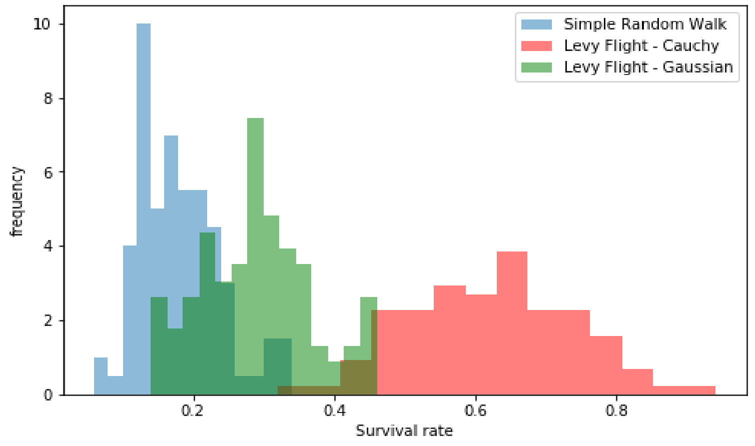


Figure 10: Histograms for survival rates of simulated animals foraging with various probabilistic distributions determining step-size (simple random walk, Cauchy-Levy flight, Gaussian-Levy flight)

The mean survival rates for each walk clearly demonstrate the advantage of following a Levy distribution, the values in order are given in the following table:

[1]Only two Levy flight distributions were included in the histogram as all four distributions diminished readability

Random Walk	Mean Survival Rate
Pareto	0.6226
Cauchy	0.6210
Levy-Smirnov	0.6059
Gaussian	0.2927
Simple	0.1707

Table 4: Simulated survival rates of four Levy flights against the simple random walk with regards to optimal search efficiency

Recall from Section (2.1.3) that the Gaussian distribution is given by the Levy stable distribution with $\alpha = 2$, making it the only distribution with finite variance in the group. This implies that the Gaussian distribution has the thinnest tails, making it the least 'Levy-like', and thus less optimal at searching. This could explain why it under-performed compared to the other, thicker tailed distributions.

For each Levy distribution, a t-test gives the probability of getting this data given that the simple random walk and the Levy flight have the same survival rate to be $< 10^{-16}$, thus beyond a reasonable doubt the Levy flight has the advantage in search efficiency in this system.

3.2.2 Evolutionary Algorithm

To provide additional evidence towards the Levy Flight Foraging Hypothesis, an evolutionary algorithm was employed to demonstrate that a simple random walker over time will evolve into the more advantageous Levy flight. Evolutionary algorithms are a class of optimisation algorithm that takes inspiration from biological evolution. The algorithm introduced in this paper aims to mimic the biological process of natural selection: reproduction, mutation and selection. The algorithm is designed such that each generation, small mutations occur to the variables that govern the animals movement (Levy index α and maximum step-size x_M). Then, a large set of different mutations would be tested against one another, with the most successful (using search efficiency as a measure of success) of the mutations passing on its genes to the next generation. This process is an example of stochastic gradient descent. For more details on this evolutionary algorithm, see Section (A.2). Figure (11) demonstrates the path of the walker through the parameter space as generations of this evolutionary algorithm take place.

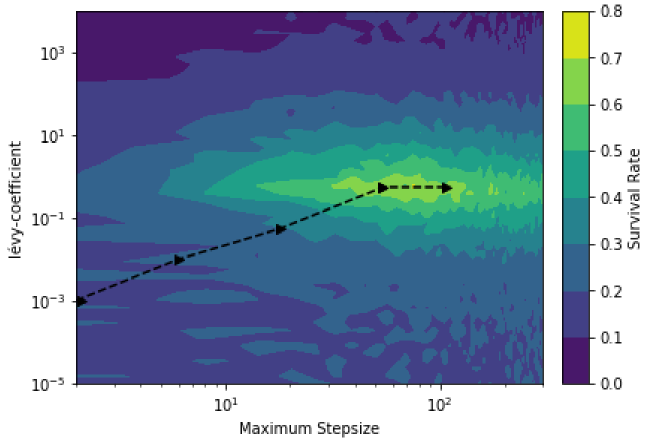


Figure 11: Evolution of foraging animal through parameter space with Levy-coefficient on y-axis and maximum step-size on x-axis, beginning with simple random walk and ending with Levy flight.

The animal begins as a simple random walk, with the maximum step-size equal to 1 and a very low Levy-coefficient (10^{-3}), and evolves into a Levy flight (Levy-coefficient ≈ 1.5, maximum step-size ≈ 75) over the course of 5 generations. Figure (12) depicts the travel paths of the final Levy flight optimized for survival.

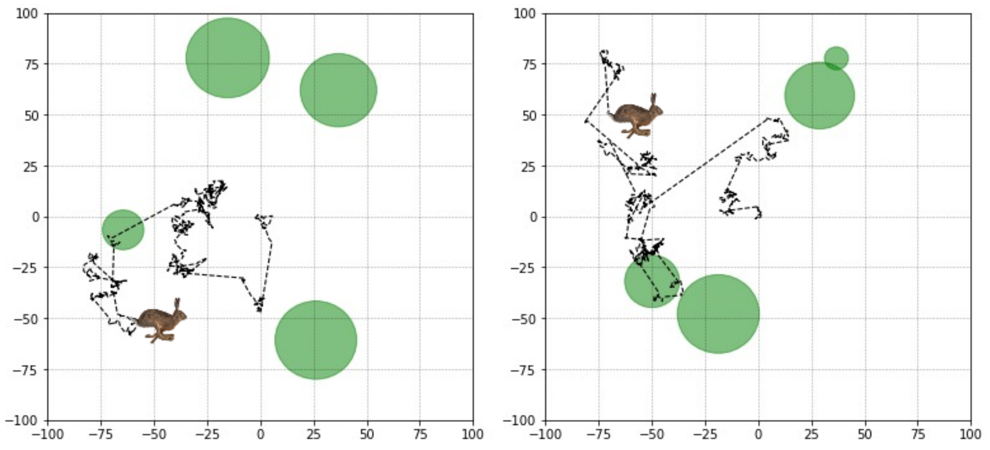


Figure 12: Levy-flight travel paths for the final generation of foraging animals generated by an evolutionary algorithm designed to optimize search efficiency

The travel paths in the above figure clearly demonstrate Levy flight behavior, showing that indeed our optimised animal is governed by a Levy flight, as opposed to the simple random walk it began with. Therefore this demonstration provides further evidence towards the Levy Flight Foraging Hypothesis.

It should be noted that the simulations in this subsection only cover a certain range of possible scenarios for which the animals search for food. Thus to continue the work in this section, a worthwhile place to begin would be to increase the diversity of scenarios.

This evolutionary algorithm, along with the previous cross comparison achieves the goal of providing substantial numerical evidence for the Levy Flight Foraging Hypothesis from a probabilistic and statistical perspective.

3.3 Connective Constant of the Union Jack Lattice

Self-avoiding walks are directly linked to the Ising model of ferromagnetism[5, 33, 34]. As a result, the critical exponents defined in Section (2.2.2) have a quantitative relationship with the parameters of the corresponding Ising model [34], thus approximations of the former, lead to contributions in the latter. The Union Jack lattice has indeed been studied in the field of ferromagnetism [35–38], but no computation of the connective constant has been calculated. Using the methods described in Section (2.2.2), an approximation to the connective constant of the Union Jack lattice will be found.

The Union Jack lattice is similar to the regular $\mathbb{Z}^2$ lattice, with the addition of vertices in the set $\{(x, y) \in \mathbb{R}^2 \mid \exists\, a, b \in \mathbb{Z}\ s.t.\ x = a + 0.5,\ y = b + 0.5\}$ and the corresponding edges attaching to $\mathbb{Z}^2$. Figure (13) below depicts the Union Jack lattice.

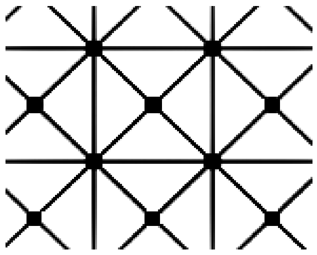


Figure 13: Two Dimensional Union Jack Lattice

3.3.1 Upper Bound

Using Alm's method to find upper bounds, we get the following results in the table below. Due to restraints on computational power[2], the approximations can only reach a certain level of accuracy. With additional resources, the code in Section (A.3) can be used to find arbitrarily accurate approximations.

[2]Intel i5 4th-gen processor, approx. 4 core-hours

$n \setminus m$	1	2	3	4	5
2	5.391 65				
3	5.181 24	5.017 54			
4	5.063 15	4.919 82	4.845 36		
5	4.978 91	4.855 16	4.792 84	4.7546	
6	4.919 18	4.812 42	4.756 28	4.722 88	4.699 77

Table 5: Upper bounds for the connective constant of the Union Jack lattice found by *Alm's method* outlined in Section (2.2.2)

3.3.2 Lower Bound

Keston's Method of Irreducible bridges was employed to find lower bounds for μ. The enumeration of bridges up to 10 steps was possible given the computational resources, the table below gives these lower bounds. For details about the implementation of Keston's Method, see Section (A.3).

n	x_c^{-1}
2	3.0000
3	3.7440
4	3.9717
5	4.0401
6	4.0697
7	4.0987
8	4.1320
9	4.1641
10	4.1913

Table 6: Lower bounds for the connective constant of the Union Jack lattice found by *Keston's Method of Irreducible Bridges* outlined in Section (2.2.2)

3.3.3 Extrapolation

With the upper and lower bound values, an estimate for the connective constant μ can be found. The upper and lower bounds will be fitted to appropriate curves and an estimate for μ will be given by the average of the two bounds. Without further data, the accuracy of the extrapolation is limited, and future research on the Union Jack lattice can be done to improve the following approximation. Figure (14) shows the increasing bounds along with the extrapolated curves and the approximations for μ.

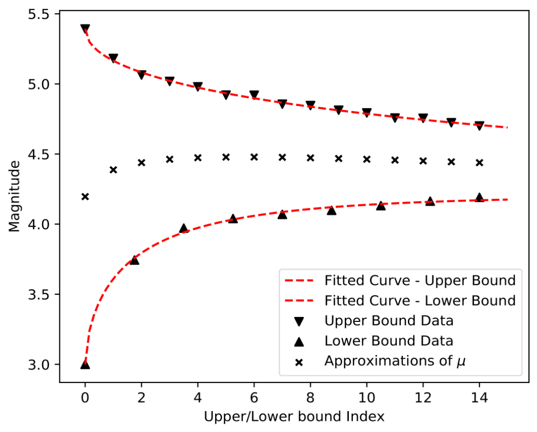


Figure 14: Upper & lower bounds for the connective constant of the Union Jack lattice, generated by Alm's Method (upper) and Keston's Method of Irreducible bridges (lower), alongside estimates of the connective constant and fitted extrapolated curves for the corresponding data

The absolute bounds for μ achieved are $4.1913 \leq \mu \leq 4.69977$, thus giving $\mu \approx 4.4455$. The error for μ is no greater than 0.25424, but by taking the difference between subsequent approximations as an error estimate we get $\mu \approx 4.4455 \pm 0.0062$. However, the reliability of the extrapolation should not be left unquestioned; the size of the interval between the upper and lower bound is still quite large. The goal of producing an approximation to the connective constant of the Union Jack lattice has certainly been met, however the accuracy of the approximation should be improved in future research given greater computing power.

3.4 Narrow Escape Problem

The Narrow Escape Problem concerns a Brownian motion confined to a container, with a small gap where it may escape. The problem is in finding the expected time that the Brownian motion escapes the container. This problem is commonly applied in the containment of harmful gasses, allowing scientists to minimise the probability of accidental escape within a reasonable timescale. The standard approach to analyse the Narrow Escape Problem is by deriving asymptotic equations of the Brownian motion in a container with a decreasing escape point[52–54]. This section will formulate a new method of analysis for the Narrow Escape Problem.

3.4.1 Method

This new method will demonstrate that it is possible to model the continuous system by an analogous discrete system i.e Markov chain. To successfully construct this discrete system, two conditions must be satisfied

(i) The scaling limit of the discrete approximation is a Brownian motion

(ii) The size of the escape point is tending to 0

So if we can construct a model that accounts for both of these limits, the properties of the Narrow Escape Problem can be analysed with the additional tool set acquired from it being a Markov chain.

Consider a Brownian particle confined to a 2 dimensional box, with a narrow exit point somewhere on the edge of the box. This Brownian particle can then be modelled as a simple random walk traversing through a lattice. The Brownian particle bounces off the inside of container, thus the corresponding lattice must have a reflecting boundary at all points, *except* at the narrow exit point, where the vertex is absorbing (representing escape). As the structure of the lattice can be taken arbitrarily, for simplicity, let the lattice be of the form $\{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x, y \leq N\}$ i.e. an $N \times N$ subset of the 2D Euclidean lattice. We will see that by fixing the size of the box and taking $N \to \infty$, the model will satisfy (i)-(ii).

With a fixed box size, as the number of vertices increases, the lattice obtains a finer 'resolution' of the Brownian particles path it is approximating as the vertices on the lattice are getting closer together. This causes the step size of the simple random walk to decrease as N increases. From Section (2.3), we have that as the step-size of the simple random walk tends to 0, in the limit we obtain a Brownian motion. Therefore, as $N \to \infty$, the discrete simple random walk becomes a Brownian motion, demonstrating (i).

Further to this, with an increasing number of vertices, the relative 'size' of the single absorbing vertex also decreases. Therefore by taking the limit as $N \to \infty$, we also obtain (ii). As both conditions are met we can conclude that this discretisation successfully models the Narrow Escape Problem in the limit. It is important to note that without the link between simple random walks and Browinian motion described in Section (2.3), this new method would not be possible.

The transformation between the continuous and discrete cases can be seen in Figure (15).

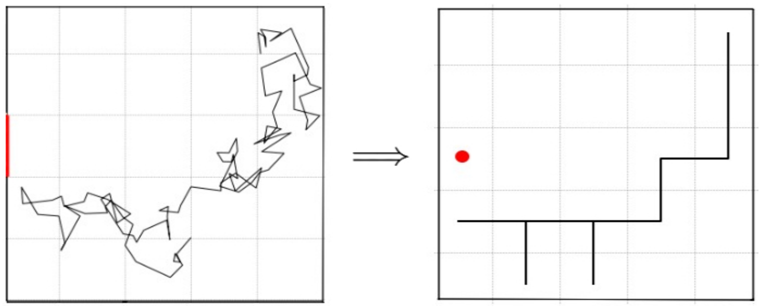


Figure 15: 2D Example of the Discretisation of the Narrow Escape Problem, with a Brownian motion on a continuous domain (left) converting to a corresponding simple random walk on a finite square subset of $\mathbb{Z}^2$ (right)

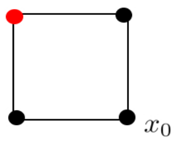
3.4.2 Analysis

To begin the analysis of this model, we fix the starting position of the random walk and the escape point to opposite ends of the lattice. This decision was made to give the particle the 'worst-case scenario', so that any other combination would give a lower expected escape time. It also seems to be the most simple choice for the starting point, as the 'centre' of the lattice is sometimes ill defined ($N = 2$, for example), and choosing a starting point near to the escape point is against the point of the problem.

Let E_t be the event of escape at time t, then the expected number of steps for escape is given by

$$\mathbb{E}[E_t] = \sum_{n=1}^{\infty} 2n \cdot P(E_{2n}) \tag{50}$$

As escape can only happen in an even number of steps due to the positions of the starting/escape point. We begin with the most elementary case; $N = 2$.



For all $t = 2n + 1$, the walker has a 0.5 chance of escape, and a 0.5 chance of returning to x_0. Additionally, the event E_{2n} can only happen if E_t has not already happened for any $t < 2n$. Thus we have

$$P(E_{2n}) = \frac{1}{2} \cdot \prod_{i=1}^{n-1} P(x_{2i} = x_0 \mid x_{2i-2} = x_0) = \frac{1}{2} \cdot \frac{1}{2^{n-1}} = \frac{1}{2^n} \tag{51}$$

subbing into Eq. (50) gives

$$\mathbb{E}[E_t] = \sum_{n=1}^{\infty} \frac{2n}{2^n} = 4 \quad . \tag{52}$$

Therefore for $N = 2$, the expected number of steps until escape is exactly 4. Solving $P(E_{2n})$ as a function of n analytically for $N > 2$ is a non-trivial task, thus constructing a Markov chain allows for computation of $P(E_{2n})$ as far as needed, thus giving approximations for $\mathbb{E}[E_t]$. For example, take the transition matrix from the Markov chain induced by the $N = 2$ system:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix}$$

P is constructed such that the escape point is the top row, and x_0 is the bottom row. Thus $P(E_{\leq 2n}) = (P^{2n})_{4,1}$ i.e. the bottom-left entry of P^{2n}. Thus $P(E_t)$ for $t \leq 2n$ can be retrieved recursively by this method. Then subbing these values of $P(E_t)$ into Eq. (50) gives the desired approximations for $\mathbb{E}[E_t]$. Below are the following approximations for $\mathbb{E}[E_t]$ for $N = 1, \ldots, 16$.

N	$\mathbb{E}[E_t]$
2	4
3	18
4	44.57
5	85.45
6	141.94
7	215.05
8	305.65
9	414.45
10	542.10
11	689.17
12	856.16
13	1043.54
14	1251.73
15	1481.12
16	1732.02

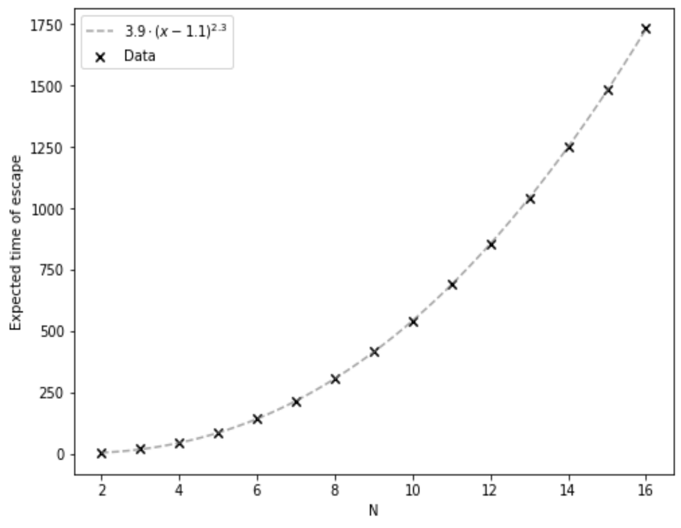


Table 7: Expected escape time for a simple random walk in the Narrow Escape Problem($\mathbb{E}[E_t]$) vs resolution of the box (N)

This accelerated growth is not exponential, but follows a polynomial. This can be seen by the Log-Log plot in Figure (16), where the data tends to a straight line, a defining characteristic of monomials: singular term polynomials with non-negative exponents e.g. $7x^2$.

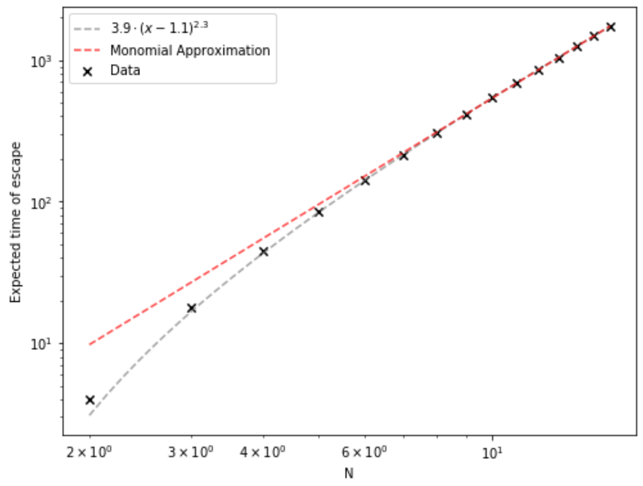


Figure 16: Log-Log plot of Expected escape time for a simple random walk in the Narrow Escape Problem($\mathbb{E}[E_t]$) vs resolution of the box (N)

This leads to the following conjecture:

Conjecture 3.0.1. For large N,

$$\mathbb{E}[E_t] \propto N^{\beta} \tag{53}$$

where $0 < \beta \in \mathbb{R}$.

Numerical estimates give

$$\mathbb{E}[E_t] \approx 4N^{2.45}. \tag{54}$$

Finding the exact value for β would be of great value to the Narrow Escape Problem and thus could be the basis for future research on this topic.

The PDF of E_t is another key topic for analysis. In particular, the trend of the probability distribution as $N \to \infty$ gives interesting insight into the Narrow Escape Problem. Values of $P(E_t)$ were calculated via the Markov chain method described above. Figure (17) gives plots of these PDFs for $N = 6, 8, 10$.

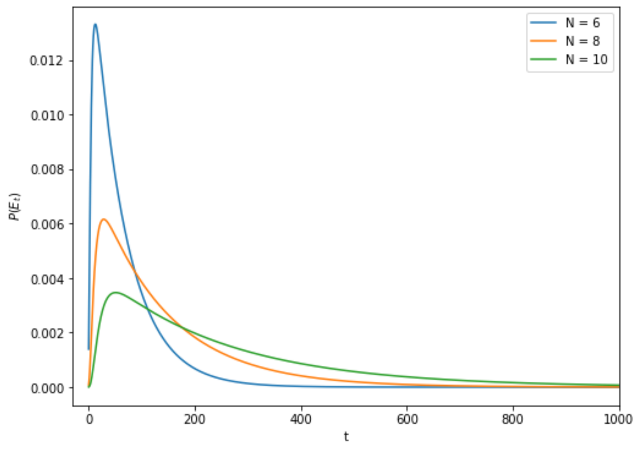


Figure 17: Probability density function for the escape time(E_t) of a simple random walk in a box of resolution $N = 6, 8, 10$

As one might expect, as N increases, the distribution flattens. A first glance suggests the underlying distribution could be a gamma distribution, to examine this, Cullen and Frey graphs come in particularly helpful. A Cullen and Frey graph maps probability distributions onto the plane with kurtosis on the y axis, and skewness on the x axis.

Cullen and Frey graphs[3] are given below for $N = 3$ and $N = 15$

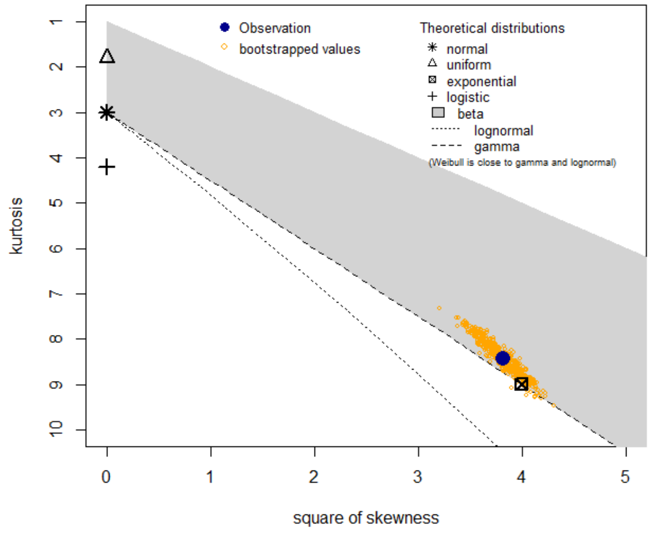


Figure 18: Cullen and Frey graph for the probability density function for the escape time(E_t) of a simple random walk in a box of resolution $N = 3$

[3]Generated via the fitdistrplus package in R.

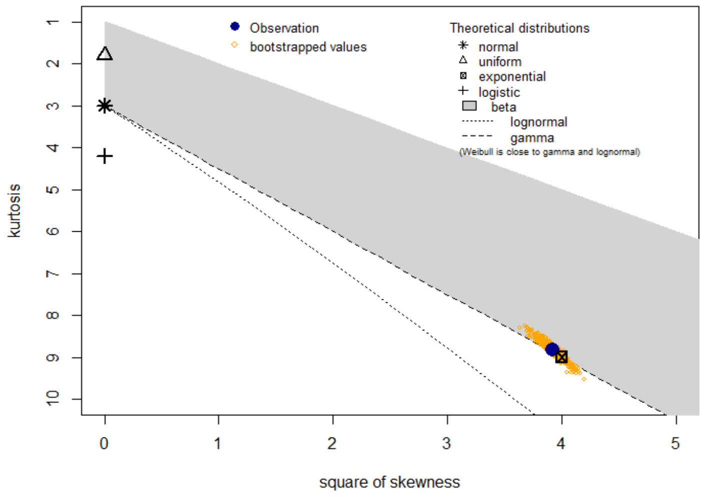


Figure 19: Cullen and Frey graph for the probability density function for the escape time(E_t) of a simple random walk in a box of resolution $N = 15$

The Cullen and Frey graphs indicate that the underlying distributions are similar to a gamma distribution, and are also in the close neighbourhood of the exponential distribution. As N increases, the kurtosis and skewness of the distribution become closer to those of the exponential distribution, 9 and 2 respectively. This can be seen directly in Figure (19) as the distribution is very close to exponential. Figure (20) below demonstrates this trend.

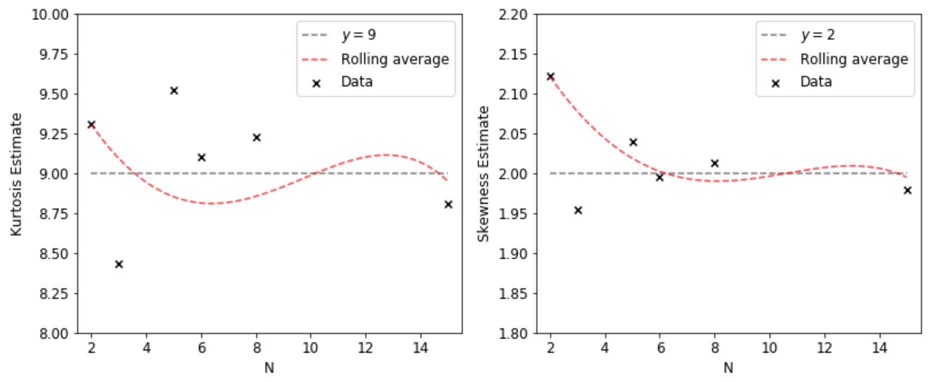


Figure 20: Kurtosis of probability density function of escape time(E_t) vs resolution of box(N) & Skewness of probability density function of escape time(E_t) vs resolution of box(N)

This indicates that the asymptotic distribution of E_t could be an exponential distribution.

As a final topic of analysis, we take a look at the tails of the PDFs. After a certain point, the PDF begins to exponentially decay, this can be seen in the semi-log plot of the PDFs;

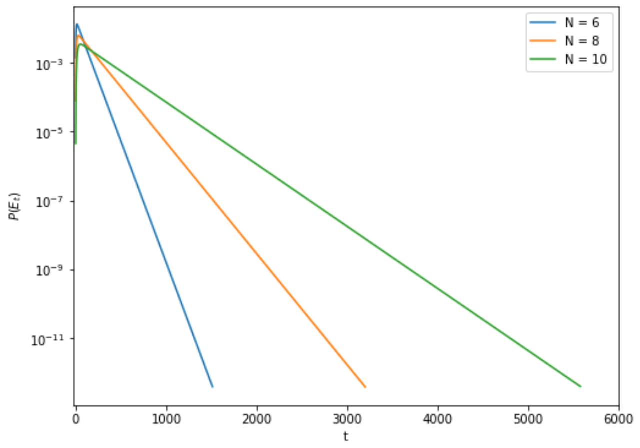


Figure 21: Semi-Log Plot of probability density function of escape time(E_t) against time (t) for resolution of box ($N = 6, 8, 10$)

The parameter of the exponential decay,

$$\lambda_N := \lim_{t\to\infty} \frac{P(E_{t+1})}{P(E_t)} \tag{55}$$

increases with N, which provides insight into the how the system changes as N increases. When $N = 2$, the probability of escape at time $t+1$ is 0.5 times smaller than it is at time t, but with $N = 3$, the probability of escape is approximately 0.873 times smaller at $t+1$ than it is at t. Thus the parameter λ_N serves as a measure of the ease/difficulty at which the walker escapes the box at each subsequent step.

The sequence $\{\lambda_N\}$ is hard to find analytically, but the simple cases $\lambda_{2,3}$ can be found. For $N = 2$ we have from Eq. (51) that $P(E_{t+1}) = 0.5 \cdot P(E_t)$, which immediately gives $\lambda_2 = 0.5$. For $N = 3$, the sequence

$$\left\{ \frac{P(E_{t+1})}{P(E_t)} \right\}$$

can be computed using a Markov chain to give

$$\{1, \frac{8}{9}, \frac{7}{8}, \frac{55}{63}, \frac{48}{55}, \frac{377}{432}, \frac{329}{377}, \ldots\}$$

taking every other term gives

$$\frac{a_{n+2}}{b_{n+2}} = \frac{8a_n - b_n}{9a_n - b_n}$$

thus solving

$$\frac{a}{b} = \frac{8a - b}{9a - b}$$

gives $\lambda_3 = \frac{\phi+1}{3} \approx 0.873$ where ϕ is the golden ratio. Exact values of λ_N for $N > 3$ are unknown, but estimates give $\lambda_N \in (0, 1)$ with $\lambda_N < \lambda_{N+1}$, however this is only backed by numerical simulations. This analysis of λ_N gives further insight into the Narrow Escape Problem; as N increases, the thicker the exponential tail of the PDF gets, which further indicates that the asymptotic nature of E_t is exponential in some form.

In practice, the escape point has a finite length. For example, a gas atom in a solid container has the narrow escape point of the microscopic cavity between the atoms in the container. For a large N, Eq. (54) can be used to estimate the expected time of escape, and an exponential distribution (for some exponential parameter γ) can be used to estimate its PDF. To continue the research on this problem from the perspective of the method described in this paper, accurate estimates of β would provide a strong basis for practical applications. Additionally, a proof of the exponential nature of the asymptotic distribution of E_t would provide a solid backbone to the analysis of this section, as the analysis is primarily numerical. Furthermore, a research into the behavior of λ_N in the limiting case, $\lambda_\infty = 1$, could be of great interest to the future research of the Narrow Escape Problem.

3.5 Extension to Reinforced Random Walks

The reinforced random walk is a random walk that takes steps depending on the number of times the adjacent edges have been traversed in the past. The limitation to this definition is that the reinforcement function can only take the edges directly attached to the current vertex into account. The motivation behind the following extension is to extend the 'reach' of the reinforcement function, allowing for edges slightly further away to direct the path of the walk.

3.5.1 Definition

The reinforced random walk takes random steps on a graph G. If v_1, v_2 are vertices in V connected by some edge $e \in E$ then we write $v_1 \sim v_2$. Similarly, if two edges e_1, e_2 are connected by a vertex v, then write $e_1 \sim e_2$. Additionally, denote the *reach* of the reinforced random walk by ρ. The reach defines the edges in E that act as inputs for the reinforcement function. First, we define $Y_n \subset E$ as the set of edges n steps away from the current position x. Thus we have

$$Y_n := \{y_i \in E \mid \exists\, z \in Y_{n-1} \text{ s.t. } z \sim y_i \text{ and } y_i \notin Y_{<n}\} \tag{56}$$

Thus Y_n is defined inductively as all of the new edges in E that can be reached in one extra step from Y_{n-1}. Following this, the *region of influence* is defined as the union of Y_i for $i = 1, \ldots, \rho$ for a walk with reach ρ. The first three Y_i's in $\mathbb{Z}^2$ are displayed below in figure (22) to provide insight for Eq. (56)

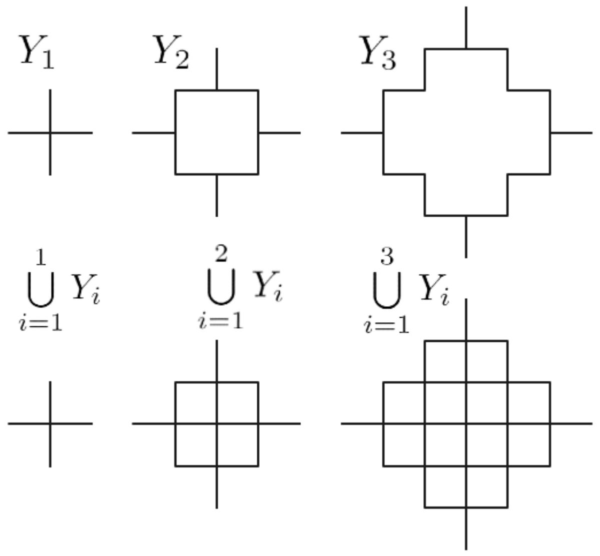


Figure 22: Regions of Influence on $\mathbb{Z}^2$ for $\rho = 1, 2, 3$

The reinforcement function is determined by the number of traversals of edges in $\bigcup_{i=1}^{\rho} Y_i$. Recall that the number of traversals of an edge e up to time t is given by $N(e, t)$, then we have

$$P\left((x_{t+1}, x_t) = y_i | x_0, \ldots, x_t\right) = \frac{f\left(N\left(y_i, t\right), N(Y_2, t), \ldots, N(Y_\rho, t)\right)}{\sum_{y \,\in\, Y_1} f\left(N\left(y, t\right), N(Y_2, t), \ldots, N(Y_\rho, t)\right)} \tag{57}$$

For $\rho = 1$, this new definition reduces to the standard definition of the reinforced random walk introduced in Section (2.2.1). To index Y_n, define $Y_{i,j}$ as the edges in Y_i that stem from the edge $y_j \in Y_{i-1}$, this will allow for more flexibility in the construction of reinforcement functions.

Increasing the reach of the reinforced random walk opens up new possibilities for research. A brief exploration of the applications of reinforced random walks with $\rho = 2$ will be given to provide insight into the importance of this extension.

3.5.2 Examples

The first application of the reinforced random walk with an extended reach will be a demonstration that the self-avoiding walk can be defined as particular case of the reinforced random walk with $\rho = 2$. To construct a self-avoiding walk, we need a reinforcement function that assigns a 0 probability to traveling along an edge which leads to a vertex already visited. If a vertex v has been visited before, then there exists at least one edge e connected to v that has been traversed before. Therefore, to remove the possibility of travelling to a vertex already visited, the reinforcement function needs to assign a 0 probability to an adjacent edge y_i if there exists a $y \in Y_{2,i}$ with $N(y,t) > 0$. Thus

$$f(N(y_i,t), N(Y_{2,i},t)) = \begin{cases} 0 & N(y_i,t) + N(Y_{2,i},t) > 0 \\ 1 & N(y_i,t) + N(Y_{2,i},t) = 0 \end{cases} \tag{58}$$

gives the desired result! In the event where all of the available edges give 0 probability, the self-avoiding walk has trapped itself and the process stops. By replacing the 0 in Eq. (58) with a small number $\epsilon > 0$, we define the *near self-avoiding walk* (NSAW):

$$f(N(y_i,t), N(Y_{2,i},t)) = \begin{cases} \epsilon & N(y_i,t) + N(Y_{2,i},t) > 0 \\ 1 & N(y_i,t) + N(Y_{2,i},t) = 0 \end{cases} \tag{59}$$

The NSAW has similar behavior to the regular SAW, however, at each step, there is a small probability of the NSAW disregarding the self avoiding property and revisiting a vertex. Additionally, in the event that the NSAW is trapped, all of the edges are assigned the same probability ϵ and it selects an edge with equal probability and carries on as usual. Therefore the NSAW can be applied in situations where the self-avoiding behavior is required, but the walk can continue indefinitely, unlike the SAW. The parameter $\epsilon \in [0,1]$ can be used to alter how self-avoiding the walk is: for $\epsilon = 0$ we have the standard SAW, for $\epsilon = 1$ we have the simple random walk, and for $\epsilon \in (0,1)$ we have an NSAW of varying degrees of strength.

Other variations of the SAW can be designed by reinforced random walks with $\rho > 2$. For example, with $\rho = 3$, the reinforcement function can be designed such that the probability of taking an edge is inversely proportional to the number of traversed edges in that general direction (up to 3 steps away). This would mean the walker could avoid regions of its neighbourhood that have been traversed more often, with the addition of the self-avoiding property. Theoretically, this type of walk would have a much lower probability of self-trapping and thus could be applied in situations where entrapment of the walk should be avoided at all costs. With $\rho = \infty$, the reinforcement function can take into account the entire path of walk and thus can be designed such that the self-avoiding walk never traps itself. Reinforced random walks with $\rho > 2$ with this self-avoiding behavior could be the basis for a new class of *super-self-avoiding walks.*

The reinforced random walk with an extended reach can also by designed to approximate an Eulerian path. An Eulerian path is a walk on a graph such that every edge is travelled exactly once. To achieve this goal with a regular reinforced random walk, the reinforcement function could be

$$f(N(y_i,t)) = \frac{1}{N(y_i,t)+1} \tag{60}$$

but this would not guarantee that it wouldn't get trapped in the situation where all possible edges have already been traversed, thus it is only an approximation. This approximation can be improved with $\rho = 2$. The walk could take steps in the direction where there are lesser travelled edges, for example with the reinforcement function

$$f(N(y_i,t), N(Y_{2,i},t)) = \frac{1}{N(y_i,t) + N(Y_{2,i},t) + 1} \tag{61}$$

With an ever further reach, the walker would be able to 'seek out' the untraveled edges in the graph.

To better understand the applications of this new generalized definition for reinforced random walks, a full investigation would have to take place, which is beyond the scope of this paper. Future research projects could study the asymptotic behavior of the NSAW as $\epsilon \to 0$, or the practical applications of the super-self-avoiding walk. The purpose of this section is primarily to develop a new method of analysis for the reinforced random walk, and also the self-avoiding walk. This new definition gives a different perspective into the study of SAWs, which could allow for a more rigorous study of the walk in the future. This new link between the reinforced random walk and the self-avoiding walk can be seen in Figure (23) below:

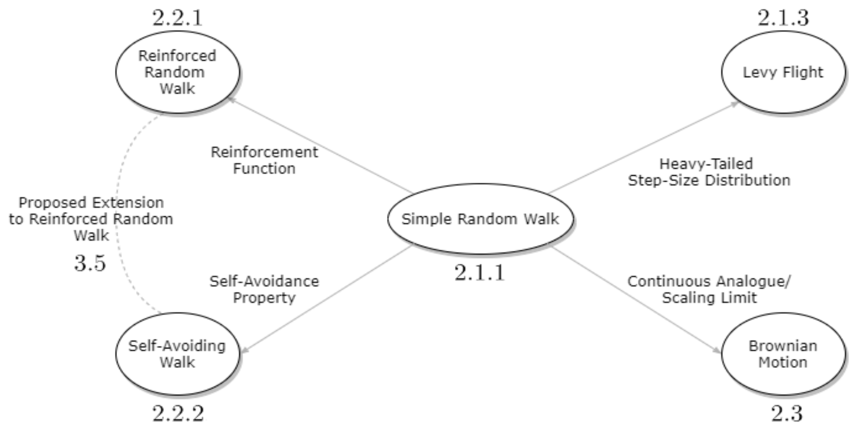


Figure 23: Five major classes of random walks and the relationships between them, including a new link between reinforced random walks and self-avoiding walks introduced in Section (3.5)

4 Conclusion

Random walks come in an array of interesting classes, each with unique properties, applications and methods of analysis. Through an extensive analysis of literature, it is clear that most papers and research projects concern only one class of random walk, for example either reinforced random walks or Levy flights, but never both. This makes sense as the mathematics behind the different classes of random walks are vastly different, as can be seen in this paper. However, this has lead to the study of random walks to be somewhat fragmented between the different classes. The primary research goal of this paper is to provide an analytical and numerical analysis on a diverse set of key classes of random walks, bringing mathematical concepts from different authors together into one paper. Following this, this paper aimed to utilise this broad approach of analysis to bring new research results to the field.

The paper begins by presenting the key theories and concepts of five major classes of random walk in the Theory & Methods section. First and foremost, the goal of the Theory & Methods section was to provide a concrete introduction to the mathematical material necessary to explore the new results introduced later in the paper. In order to give a coherent and relevant overview of these classes, there was a necessary process of collecting and refining a selection of literature, and then constructing a consistent and intuitive system of notation to bring the different concepts together into one format. Therefore, the largest contribution to the Theory & Methods section was the rewriting of proofs and ideas from different authors and piecing it all together with the right notation.

Following the Theory & Methods section, a series of new results are presented in the Results & Simulations section. The section begins with an array of simulations regarding the simple random walk, in dimensions one and two. The theorems regarding the one dimensional simple random walk were verified through numerical simulation, and then the two methods of approximating the 2D Dirichlet Problem were simulated and compared with regards to accuracy and efficiency. Following this, the paper discusses the Levy Flight Foraging Hypothesis. The research goal in this subsection was to provide a new array of numerical evidence to support the hypothesis. A simulation was constructed to compare different search strategies of a foraging animal, and a direct comparison yielded that all four of the chosen Levy flights had a higher survival rate than the simple random walk it was competing against. Additionally, an evolutionary algorithm was designed by considering a stochastic gradient descent inspired by natural selection. This demonstrated that over the course of 5 generations, the 'animal' would evolve away from the simple random walk, and towards the more successful Levy flight, as the hypothesis predicts. Following this, the Union Jack lattice was defined, and numerical methods of finding upper and lower bounds were implemented to approximate its connective constant. The final approximation gives $\mu \approx 4.45$ which is the first known approximation for the Union Jack lattice, despite its relevance in neighbouring fields. The next major result was a new method of analysing the Narrow Escape Problem, which directly applies the link between simple random walks and Brownian motion to formulate an analogous discrete system. This discrete system was then modelled as a Markov chain which allowed for direct approximations of the expected escape time (E_t), amongst other interesting results. Lastly, an extension to the class of reinforced random walks was formulated by allowing the reinforcement function to take further away edges into account. In particular, the research in this paper has shown that there are many interesting results to be found by considering multiple different classes together. Firstly, the extension to the reinforced random walks demonstrated that the self-avoiding walk can be redefined as a particular case of the extended reinforced random walk, which consequently allows for an entirely new perspective on the mathematics governing the self-avoiding walk. The implications of this new definition can certainly form the basis of a new research project on the topic. Secondly, the link between simple random walks and Brownian motion allowed for the construction of a new method of analysis for the Narrow Escape Problem, as presented in this paper. This further supports that there is much to be gained by analysing the field of random walks from a broad perspective.

The research in this paper though innovative, is not as extensive as it could be. First, the animal foraging simulation for the Levy Flight Foraging Hypothesis demonstrates the desired results in only one system. To combat this, the layout of food was randomised each iteration, showing that regardless of where the food is, the Levy flight still has the best chance of discovering its location. To improve the results in the future, more complex simulations could be developed, perhaps including a notion of different terrains and natural landmarks such as rivers and mountains. This would make the simulations more realistic and thus offer more reliable results. Second, the analysis of the new method of studying the Narrow Escape Problem is primarily numerical, and thus future research could focus on the method from a more analytical perspective, giving a stronger understanding of the numerical results and why they occur. Another minor limitation regards the connective constant of the Union Jack lattice. Although the methods of constructing upper and lower bound were successfully implemented, the accuracy could be improved with the use of stronger computational resources (it is not uncommon for connective constant approximations to utilise high-end supercomputers). Lastly, the extension to the reinforced random walks offers a wealth of new potential research on the topic, future research could focus on developing the exact implications of this new extension, and in particular, what can be learnt about the self avoiding walk by considering it from this new perspective.

Through the development of new results, this paper has opened up various new avenues for potential research projects to take place in the future, possibly sparking a new front of research on the topic, bringing different classes of random walk together into one broad and diverse field.

References

[1] Karl Pearson and John Blakeman. *A mathematical theory of random migration*, volume 1. Dulau and Company, 1904.

[2] Lord Rayleigh. Xii. on the resultant of a large number of vibrations of the same pitch and of arbitrary phase. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 10(60):73–78, 1880.

[3] John William Strutt and Baron Rayleigh. *The theory of sound*. Dover, 1945.

[4] Nicolas E Humphries and David W Sims. Optimal foraging strategies: Lévy walks balance searching and patch exploitation under a very broad range of conditions. *Journal of theoretical biology*, 358:179–193, 2014.

[5] BK Chakrabarti and S Bhattacharya. Study of an ising model on a self-avoiding-walk lattice. *Journal of Physics C: Solid State Physics*, 16(29): L1025, 1983.

[6] John Michael Hammersley. Long-chain polymers and self-avoiding random walks. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 29–38, 1963.

[7] Fateme Safaeifard, Seyed Peyman Shariatpanahi, and Bahram Goliaei. A survey on random walk-based stochastic modeling in eukaryotic cell migration with emphasis on its application in cancer. *Multidisciplinary Cancer Investigation*, 2(1):1–12, 2018.

[8] Peter G Doyle and J Laurie Snell. *Random walks and electric networks*, volume 22. American Mathematical Soc., 1984.

[9] David Aldous and James Fill. Reversible markov chains and random walks on graphs, 1995.

[10] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of data science*. Cambridge University Press, 2020.

[11] Geoffrey Grimmett and Dominic Welsh. *Probability: an introduction*. Oxford University Press, 2014.

[12] Russell Lyons and Yuval Peres. *Probability on trees and networks*, volume 42. Cambridge University Press, 2017.

[13] Hannelore Lisei Mihai Iancu. Properties of random walks in dimension one, 2017.

[14] Nolan Outlaw. Markov chains, random walks, and card shuffling, 2016.

[15] Nicolas E Humphries, Henri Weimerskirch, Nuno Queiroz, Emily J Southall, and David W Sims. Foraging success of biological lévy flights recorded in situ. *Proceedings of the National Academy of Sciences*, 109 (19):7169–7174, 2012.

[16] Marina E Wosniack, Marcos C Santos, Ernesto P Raposo, Gandhi M Viswanathan, and Marcos GE da Luz. The evolutionary origins of lévy walk foraging. *PLoS computational biology*, 13(10):e1005774, 2017.

[17] Arild O Gautestad and Atle Mysterud. The lévy flight foraging hypothesis: forgetting about memory may lead to false verification of brownian motion. *Movement Ecology*, 1(1):9, 2013.

[18] Alexei V Chechkin, Ralf Metzler, Joseph Klafter, Vsevolod Yu Gonchar, et al. *Introduction to the theory of Lévy flights*. Wiley Online Library, 2008.

[19] Gady Kozma. Reinforced random walk. *arXiv preprint arXiv:1208.0364*, 2012.

[20] Brian Hayes. Computing science: How to avoid yourself. *American Scientist*, 86(4):314–319, 1998.

[21] Michaël Bon, Davide Marenduzzo, and Peter R Cook. Modeling a self-avoiding chromatin loop: relation to the packing problem, action-at-a-distance, and nuclear context. *Structure*, 14(2):197–204, 2006.

[22] Iwan Jensen. Enumeration of self-avoiding walks on the square lattice. *Journal of Physics A: Mathematical and General*, 37(21):5503, 2004.

[23] EJ Janse van Rensburg. Approximate enumeration of self-avoiding walks. *Algorithmic Probability and Combinatorics*, 520:127–151, 2010.

[24] Neal Madras and Gordon Slade. *The self-avoiding walk*. Springer Science & Business Media, 2013.

[25] Sven Erick Alm. Upper bounds for the connective constant of self-avoiding walks. *Combinatorics, Probability and Computing*, 2(2):115–136, 1993.

[26] Sven Erick Alm and Robert Parviainen. Bounds for the connective constant of the hexagonal lattice. *Journal of Physics A: Mathematical and General*, 37(3):549, 2004.

[27] Yuxuan Chen. *Senior Thesis*.

[28] Iwan Jensen. Improved lower bounds on the connective constants for two-dimensional self-avoiding walks. *Journal of Physics A: Mathematical and General*, 37(48):11521, 2004.

[29] Frank B Knight. On the random walk and brownian motion. *Transactions of the American Mathematical Society*, 103(2):218–228, 1962.

[30] Prof. Yuh-Dauh Lyuu. *Brownian Motion as Limit of Random Walk*. National Taiwan University, 2014.

[31] Prof. David Gamarnik. *Brownian motion: Introduction.*

[32] Peter Mörters and Yuval Peres. *Brownian motion*, volume 30. Cambridge University Press, 2010.

[33] Michael E Fisher and David S Gaunt. Ising model and self-avoiding walks on hypercubical lattices and" high-density" expansions. *Physical Review*, 133(1A):A224, 1964.

[34] Gordon Slade. Self-avoiding walk, spin systems and renormalization. *Proceedings of the Royal Society A*, 475(2221):20180549, 2019.

[35] TC Choy and RJ Baxter. Spontaneous magnetizations of the ising model on the union jack lattice. *Physics Letters A*, 125(8):365–368, 1987.

[36] Vincent Mellor. Numerical simulations of the ising model on the union jack lattice. *arXiv preprint arXiv:1101.5015*, 2011.

[37] Adam Lipowski. Mixed-spin ising model with four-spin interaction. *Physica A: Statistical Mechanics and its Applications*, 248(1-2):207–212, 1998.

[38] FY Wu and KY Lin. Ising model on the union jack lattice as a free fermion model. *Journal of Physics A: Mathematical and General*, 20(16):5737, 1987.

[39] Eric W Weisstein. Random walk–1-dimensional. *https://mathworld. wolfram. com/*, 2002.

[40] Branko Grünbaum. Projection constants. *Transactions of the American Mathematical Society*, 95(3):451–465, 1960.

[41] Hermann König, Carsten Schütt, and Nicole Tomczak-Jaegermann. Projection constants of symmetric spaces and variants of khintchine's inequality. *Journal für die reine und angewandte Mathematik*, 1999(511):1–42, 1999.

[42] *NIST Digital Library of Mathematical Functions Equation 5.5.5.* https://dlmf.nist.gov/5.5iii, Release 1.0.27 of 2020-06-15. URL `https://dlmf.nist.gov/5.5#iii`. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

[43] Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

[44] Kai Nordlund. Basics of monte carlo simulations. *http://beam.helsinki.fi/ knordlun/mc/mc7nc.pdf*, 2006.

[45] David P. Goldenberg. Physical principles in biology. *University of Utah*, 2019.

[46] Alexander A Dubkov, Bernardo Spagnolo, and Vladimir V Uchaikin. Lévy flight superdiffusion: an introduction. *International Journal of Bifurcation and Chaos*, 18(09):2649–2672, 2008.

[47] Harry Pollard et al. The representation of $\hat{e}\{-\hat{x}\backslash$lambda$\}$ *asalaplaceintegral*. *Bulletin of the American Mathematical Society*, $52\,(10) : 908 - -910, 1946$.

[48] Hugo Duminil-Copin and Stanislav Smirnov. The connective constant of the honeycomb lattice equals 2+ 2. *Annals of Mathematics*, pages 1653–1665, 2012.

[49] Michael E Fisher and MF Sykes. Excluded-volume problem and the ising model of ferromagnetism. *Physical Review*, 114(1):45, 1959.

[50] Harry Kesten. On the number of self-avoiding walks. *Journal of Mathematical Physics*, 4(7):960–969, 1963.

[51] Sidney I Resnick. *Adventures in stochastic processes*. Springer Science & Business Media, 1992.

[52] Zeev Schuss, Amit Singer, and David Holcman. The narrow escape problem for diffusion in cellular microdomains. *Proceedings of the National Academy of Sciences*, 104(41):16098–16103, 2007.

[53] Habib Ammari, Josselin Garnier, Hyeonbae Kang, Hyundae Lee, and Knut Sølna. The mean escape time for a narrow escape problem with multiple switching gates. *Multiscale Modeling & Simulation*, 9(2):817–833, 2011.

[54] David Holcman and Zeev Schuss. Stochastic narrow escape in molecular and cellular biology. *Analysis and Applications. Springer, New York*, 2015.

[55] Youcef Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

A Implementation

There are four key sections where code was employed to implement the methods discussed in this paper. For each section, the coding strategy and technical details will be discussed, following closely by the raw code, which is included for clarity and transparency. All of the key concepts and methods are provided below, however, the non-essential and/or non-enlightening code will not be included in this section to maintain a level of relevance. The reader is encouraged to implement the code themselves to gain a deeper understanding of the systems. To implement the methods, the programming language Python was used on an IDE (Integrated Development Environment) called Spyder. It should also be noted that the raw code has been slightly edited for readability, however, it is not perfect and some parts may not be intuitive for the reader. All of the following code is completely original and not taken from existing literature or online databases.

A.1 Simple Random Walk

The simple random walk is the easiest of the random walks to implement, as it requires no additional constraints or distributions to govern the movement. This subsection will describe the method of simulating the simple random walk numerically in dimensions one and two. Following this, the Monte-Carlo method and the Method of Relaxation will be described and the corresponding code will be provided.

To construct the simple random walk, one must first design a function that takes the current location of the walker as input, takes a single step from a set of finite options, then reports back the new location. In the code below, this function will be named 'take_a_step'. The next stage is to construct a for-loop to allow the walker to take a large number of steps, and then store the positions in an array, which will be named 'pos_track'.

Below is the code for the simulation of simple random walks of dimension one and two:

Simple Random Walk Dimension 1 :

```
# 1D - simple random walk

import numpy as np
import random

def take_a_step(position):

    #walker takes a single step

    steps = [[-1],[1]]
    new_step = np.array(random.choice(steps))
    position = position + new_step
```

```python

    return position


x = np.array([0])      #origin
N = 1000               #number of steps


#tracking position
pos = x
pos_track = x
for i in range(0,N):
    pos = take_a_step(pos)
    pos_track = np.vstack((pos_track,pos))
```

Simple Random Walk Dimension 2 :

```python
#2D simple random walk - square lattice
import numpy as np
import random
from matplotlib import pyplot as plt

x = np.array([0,0])
N = 100


def take_a_step(position):

    #walker takes one step

    steps = [[-1,0],[1,0],[0,1],[0,-1]]
    new_step = np.array(random.choice(steps))
    position = position + new_step

    return position


#taking N steps and tracking position

pos = x
pos_track = x
for i in range(0,N):
    pos = take_a_step(pos)
    pos_track = np.vstack((pos_track,pos))

#plotting result
pt = pos_track
plt.figure(figsize = (6,6))
for i in range(1,len(pt[:,0])):
    plt.plot([pt[i-1,0],pt[i,0]],[pt[i-1,1],pt[i,1]],color = 'k')
plt.axis('equal')
plt.grid(color='k', linestyle='--', linewidth=0.5,alpha = 0.4)
plt.show()
```

In Section (2.1.2) the Monte-Carlo method and the Method of Relaxation was introduced to approximate the solution to the 2D Dirichlet Problem. Recall that the Monte-Carlo method comprises of simulating a large number of simple random walks beginning at the different interior points, and estimating $p(x, y)$ as the ratio of successful random walks to the total number of random walks. The Method of Relaxation consists of averaging neighbouring interior points in a a particular way in order to approximate $p(x, y)$, for the full definition see Section (2.1.2.2). These two methods were implemented numerically and the code can be seen below with comments to guide the reader through the lines of code:

Monte Carlo Method:

```
#Simulating and Analysing the Monte-Carlo Method for 2D Dirichlet Problem

import numpy as np
import time
import random
from tqdm import tqdm
from matplotlib import pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D

a = 17                #length of grid
b = 17                #width of grid
N = 10000             #maximum number of steps walks may take before deletion
its  = 100            #number of random walk per starting position of grid

x_0s = np.zeros(((a-2)*(b-2),2))
p_x = np.zeros((a-2,b-2,its))
successes = np.zeros((a-2,b-2))

#building co-ordinate system
for i in range(a-2):
    for j in range(b-2):
        x_0s[(b-2)*i+j,0]  = i
        x_0s[(b-2)*i+j,1]  = j


def take_a_step(position):

    #walker takes a step

    steps = [[-1,0],[1,0],[0,1],[0,-1]]
    new_step = np.array(random.choice(steps))
    position = position + new_step

    return position


#setting up for-loop to find average time to reach accurate solution
dum = 10
tic = time.time()
for p in tqdm(range(dum)):
    for j in tqdm(range(its)):
        for k in range((a-2)*(b-2)):
            success = 0
            pos = np.array([x_0s[k,0],x_0s[k,1]])

            #scenario 1 below - uncomment to use!
    # =============================================================================
    #         for i in range(0,N):
    #             pos = take_a_step(pos)
    #             if pos[0] == -1 or pos[0] == a-2 or pos[1] == b-2 :
    #                 break
    #             if pos[1] == -1 and pos[0]<a-3:
    #                 break
    #             if pos[1] == -1 and pos[0] == a-3:
    #                 successes[int((k-np.mod(k,b-2))/(b-2)),np.mod(k,b-2)] =
    #successes[int((k-np.mod(k,b-2))/(b-2)),np.mod(k,b-2)] + 1
    #                 break
    # =============================================================================

            #scenario 2 below - uncomment to use!
    # =============================================================================
    #         for i in range(0,N):
    #             pos = take_a_step(pos)
    #             if pos[0] == -1 or pos[0] == a-2 or pos[1] == b-2 :
    #                 break
    #             if pos[1] == -1:
    #                 successes[int((k-np.mod(k,b-2))/(b-2)),np.mod(k,b-2)] =
    #successes[int((k-np.mod(k,b-2))/(b-2)),np.mod(k,b-2)] + 1
    #                 break
    #
    #     p_x[:,:,j] = successes/(j+1)
    # =============================================================================

toc = time.time()
print(f'Time Elapsed : {(toc-tic)/dum}')

#Plotting solution

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')

```

```python
X = np.arange(0, b-2, 1)
Y = np.arange(0, a-2, 1)
X, Y = np.meshgrid(X, Y)
Z = p_x[:,:,9000]


surf = ax.plot_surface(X,Y,Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

Method of Relaxation:

```python
#Simulating and Analysing the Method of Relaxation for 2D Dirichlet Problem

import numpy as np
from tqdm import tqdm
from matplotlib import pyplot as plt
from matplotlib import cm
import time
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D

#setting up for-loop to find average time to reach accurate solution
dum = 10
tic = time.time()
for i in tqdm(range(dum)):


    a = 17                                  #length of grid
    b = 17                                  #width of grid
    its = 200                               #number of iterations
    p_x = np.zeros((b,a,its))
    x_coords = np.linspace(0,a-1,a)
    y_coords = np.linspace(0,b-1,b)
    graph = np.zeros((a*b,2))

    #generating 2D graph
    for j in range(0,a):
        for i in range(0,b):
            graph[a*i+j,0] = x_coords[j]
            graph[a*i+j,1] = y_coords[i]

    #Building the set of edges via finding closest neighbours of each vertex
    neighbours = []
    g = graph
    for k in range(a*b):
        neighbours.append(np.array(np.where(abs(g[k,1]-g[:,1])+abs(g[k,0]-g[:,0])<1.1))[0])
        neighbours[k] = np.delete(neighbours[k], np.where(neighbours[k] == k), axis=0)

    vals = np.zeros((a*b,1))
    scenario = 16        #for scenario 1 set equal to 2, for scenario 2 set equal to 16
    vals[:scenario] = 1

    #finding interior of graph i.e. where the walker may walk
    interior = []
    for i in range(a*b):
        if len(neighbours[i]) == 4:
            interior.append(i)

    #approximating p(x,y) over many iterations
    for i in range(its):
        for k in range(len(vals)):
            if k in interior:
                vals[k] = np.mean(vals[neighbours[k]])
                p_x[int((k-np.mod(k,a))/(a)),np.mod(k,a),i] = vals[k]
        if i > 0:
           errorest = (np.abs(p_x[:,:,i]-p_x[:,:,i-1]).sum())/15**2
           p_x1 = p_x[1:-1,1:-1]

    #neatening solution
    p_x = p_x[1:-1,1:-1]

toc = time.time()
print(f'Time Elapsed : {(toc-tic)/dum}')

#plotting solution of 2D Dirichlet Problem via Method of Relaxation

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')

Y = np.arange(0, b-2, 1)
Y = -1* Y
X = np.arange(0, a-2, 1)
X,Y = np.meshgrid(X, Y)
Z = np.transpose(p_x[:,:,99])

surf = ax.plot_surface(X,Y,Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

A.2 Levy Flight Foraging Hypothesis

To provide evidence to the Levy Flight Foraging Hypothesis, a simulation was constructed to test different search strategies. This subsection will first describe the method of constructing a Levy flight numerically, before moving onto the technical details of the animal foraging simulation. Following this, the evolutionary algorithm will be described. All of the necessary code will also be provided after each explanation.

To implement a Levy flight one first has to build a function that returns a sequence of Levy flight steps, which consists of implementing the Levy PDF numerically, and then sampling the step-sizes from this distribution. Additionally, as the step-size and direction are in polar co-ordinates, one needs to write a function that converts polar co-ordinates into Cartesian co-ordinates in order to plot the final travel path. These two functions are named 'take_levy_steps' and 'polar_to_cart'. Lastly, one needs to track the positions of the flight path after each step. The code for this implementation can be seen below:

Levy Flight Implementation:

```python
#Simulating the Levy flight

import numpy as np
from matplotlib import pyplot as plt

x = np.array([0,0]) #origin
N = 2000            #number of steps

def polar_to_cart(r,theta):

    #Converts polar coordinates to cartesian coordinates

    z = r * np.exp( 1j * theta )
    x_coord = np.real(z)
    y_coord = np.imag(z)
    return np.array([x_coord, y_coord])

def take_levy_steps(x_0,N,L):

    #takes a sequence of N levy steps from one of 3 Levy stable pdfs

    t = np.linspace(0.51,50,10*N)

    if L == 1:    #L vy-Smirnov distribution
        s = (1/np.sqrt(2*np.pi))*(t-0.5)**(-3/2)*(np.exp(-1/(2*(t-0.5))))

    if L == 2:    # Cauchy distribution
        s = (1/np.pi)*(1/(0.01+(t-1)**2))

    if L == 3:    # Gaussian distribution
```

```
        s = (1/np.sqrt(2*np.pi))*(np.exp(((t-1)**2)/-2))


    #sampling distribution
    stepsizes = np.random.choice(t,size = N, p = s/s.sum())
    directions = np.random.uniform(low = 0, high = 2*np.pi, size = N)
    steps = polar_to_cart(stepsizes, directions)
    pos_track = x_0
    pos = x_0

    #tracking position
    for i in range(0,N):
        pos = pos + steps[:,i]
        pos_track = np.vstack((pos_track,pos))
    return pos_track


#plotting travel path
pos_track = take_levy_steps(x,N,2)
fig = plt.figure(figsize = (6,6))
plt.plot(pos_track[:,0],pos_track[:,1], color = 'k', linewidth = 1)
plt.axis('equal')
plt.scatter([-1,2],[-0.7,4],color ='white')
plt.grid(color='k', linestyle='--', linewidth=0.5,alpha = 0.4)
plt.show()
```

Now using this code we can start to build the animal foraging simulation. This simulation works by tracking the position of a Levy flight, and recording the number of times it finds 'patches of food'. The patches of food are given as 2 dimensional circles that are randomised each iteration with respect to location and size in order to reduce the risk of bias. In the simulation there are exactly 4 patches of food, where the location of the centre of the circles are drawn from a uniform distribution from $[-100, 100]$, and the size of the circles are drawn from a uniform distribution from $[5, 20]$. The difficulty of finding food can be varied greatly by editing the frequency and size of the circles, which would offer an equally valid simulation.

Another important technical detail is that a cost function (proportional to distance) was implemented to limit the total amount of movement the walker can make. Without this cost function, the random walker will happily take only large steps to maximise the chance of success, which is unrealistic as animals cannot travel at such speeds for long periods.

Randomising the location and size of the food allows the simulation to cover a wider range of situations, however, more could be added to the simulation to increase the reliability of the results. To improve this simulation in a future research project, a first step would be to cover a greater array of different scenarios for the walkers to travel upon.

The following code below demonstrates the implementation of this simulation, and also computes survival rates for different combinations of levy-coefficient and maximum step size (producing the background of Figure (11)):

Levy Flight Foraging Simulation:

```python
#Levy Flight Foraging Hypothesis Simulation
#computing survival rates over a large parameter space

import numpy as np
from matplotlib import pyplot as plt

def take_levy_steps(x_0,N,b,m,max_energy):

    # x_0 : Initial Position
    # N : Number of Steps
    # b : Levy coefficient
    # m : Maximum step-size
    # max_energy : maximum energy of animal


    # defining heavy tailed pdf using levy-coefficient and max step-size
    t = np.linspace(0.51,m,10*N)
    s = (1/np.pi)*(1/((b)+(t-1)**2))

    #sample from pdf
    stepsizes = np.random.choice(t,size = N, p = s/s.sum())
    directions = np.random.uniform(low = 0, high = 2*np.pi, size = N)

    #polar to cartesian conversion
    steps = polar_to_cart(stepsizes, directions)
    pos_track = x_0
    pos = x_0
    cost = np.zeros(N)

    #tracking position
    for i in range(0,N):
        pos = pos + steps[:,i]
        pos_track = np.vstack((pos_track,pos))
        cost[i] = (b)**0.5*(steps[0,i]**2 + steps[1,i]**2)**0.5
        if cost.sum() > max_energy:
            break

    return pos_track

def polar_to_cart(r,theta):

    #converting polar coordinates to cartesian coordinates

    z = r * np.exp( 1j * theta )
    x_coord = np.real(z)
    y_coord = np.imag(z)

    return np.array([x_coord, y_coord])


#%%

b_s = np.logspace(-5,4,50)       #range of levy coefficients to test
m_s = np.linspace(2,300,50)      #range of max step sizes to test
b_vs_m = np.zeros((50,50))

#initialising simulation
x_0 = np.array([0,0])
N = 500
max_energy = 750
its = 100

#finding survival rates of every combination of b vs m
for i in range(0,50):
    for j in range(0,50):
        for n in range(0,its):
            pos_track = take_levy_steps(x_0,N,b_s[i],m_s[j],max_energy)

            f = np.random.uniform(low = -100,high = 100, size = 8)
            food = np.array([[f[0],f[1]],[f[2],f[3]],[f[4],f[5]],[f[6],f[7]]])
            r = np.random.uniform(low = 5,high = 20, size = 4)
            pt = pos_track
            for k in range(4):
                if (((pt-food[k,:])[:,0]**2+(pt-food[k,:])[:,1]**2)**0.5).min()<r[k]:
                    b_vs_m[i,j] = b_vs_m[i,j] + 1

b_vs_m = b_vs_m/its

#plotting contour plot of survival rates over different levy coefficients and
#maximum step sizes

b_s = np.logspace(-5,4,50)
m_s = np.linspace(2,300,50)

fig1, ax2 = plt.subplots(constrained_layout=True)

CS = ax2.contourf(m_s,
                  b_s,
                  b_vs_m,
                  10,
                  cmap='viridis',
                  levels = 7,
                  extent=[2,300,0.0001,10],
                  vmin=0,
                  vmax=0.8)
```

```
cbar = fig1.colorbar(CS)
plt.yscale('log')
plt.xscale('log')
plt.ylabel("l vy -coefficient")
plt.xlabel("Maximum Stepsize")
cbar.ax.set_ylabel('Survival Rate')
plt.scatter(2.12,0.001, color = 'k', marker = 'x')

plt.show()
```

To finish this subsection we will discuss the implementation of the evolutionary algorithm used in Section (3.2.2). The evolutionary algorithm designed is a type of stochastic gradient descent algorithm. It should be noted that this algorithm was not developed following any existing literature, but developed originally by using the concept of natural selection as inspiration.

The algorithm works by first considering the Levy-coefficient and maximum step-size of the current animal, and generating an array of 25 combinations of Levy-coefficient and maximum step-size that closely resemble the current animal. These new combinations represent the new mutations within the animals offspring. The next step is to test each of the 25 combinations and recording the survival rates for each one in an array. The most successful of the mutations 'passes on its genes' and the process begins again. An additional feature added to the algorithm was a form of fine-tuning such that if all of the new mutations were less optimal than the previous generation, the diversity of mutations is reduced i.e. the following sets of mutations form a tighter grid of the parameter space (Levy-coefficient vs max step size). The code to implement this evolutionary algorithm is given below alongside comments to describe the individual packets of code:

Evolutionary Algorithm for Levy Flight Foraging Hypothesis:

```
#Evolutionary Algorithm for Levy Flight Foraging Hypothesis

import numpy as np
from matplotlib import pyplot as plt

def polar_to_cart(r,theta):

    #Converts polar coordinates to cartesian coordinates

    z = r * np.exp( 1j * theta )
    x_coord = np.real(z)
    y_coord = np.imag(z)
    return np.array([x_coord, y_coord])

def take_levy_steps(x_0,N,b,m,max_energy):

    #take a sequence of levy steps

    #sample from heavy tailed distribution
    t = np.linspace(0.51,m,10*N)
    s = (1/np.pi)*(1/((b)+(t-1)**2))
    stepsizes = np.random.choice(t,size = N, p = s/s.sum())
    directions = np.random.uniform(low = 0, high = 2*np.pi, size = N)
    steps = polar_to_cart(stepsizes, directions)

    #tracking position
    pos_track = x_0
    pos = x_0
    cost = np.zeros(N)
    for i in range(0,N):
        pos = pos + steps[:,i]
        pos_track = np.vstack((pos_track,pos))
        cost[i] = (b)**0.5*(steps[0,i]**2 + steps[1,i]**2)**0.5
        if cost.sum() > max_energy:
            break

    return pos_track

b = 0.001                           #starting levy coefficient
m = 2                               #starting maximum step-size
d1 = 10                             #gradient descent factor in b-axis
d2 = 5                              #gradient descent factor in m-axis
maxit = 3                           #number of iterations/'generations'
N = 500                             #max number of steps
max_energy = 750                    #max energy
x_0 = np.array([0,0])               #origin of walk
its = 250                           #number of simulations for each proposed mutation
trace = np.zeros((maxit,2))         #this was used to create the travel path of the evo alg.

for n in range(maxit):

    #construct 25 nearby mutations in parameter space
    #note : diversity of mutations depend directly on d1 & d2, see below

    trace[n,:] = [m,b]
    success_rates = np.zeros((5,5))
    b_s = np.array([(1/d1)*b,0.5*(((1/d1)*b)+b),b,0.5*((d1*b)+b),b*d1])
    m_s = np.array([(1/d2)*m,0.5*(((1/d2)*m)+m),m,0.5*((d2*m)+m),m*d2])

    #simulate each mutation over many iterations to approximate survival rate
    for i in range(5):
        for j in range(5):
            for n in range(0,its):
                #find travel path
                pos_track = take_levy_steps(x_0,N,b_s[i],m_s[j],max_energy)

                #constructing randomised food patches
                f = np.random.uniform(low = -100,high = 100, size = 8)
                food = np.array([[f[0],f[1]],[f[2],f[3]],[f[4],f[5]],[f[6],f[7]]])
                r = np.random.uniform(low = 5,high = 20, size = 4)

                #if the animal hits the food, mark as success
                pt = pos_track
                for k in range(4):
                    if (((pt-food[k,:])[:,0]**2+(pt-food[k,:])[:,1]**2)**0.5).min()<r[k]:
                        success_rates[i,j] = success_rates[i,j] + 1
            success_rates[i,j] = success_rates[i,j]/its
    print(b,m)
    print()
    print(success_rates)
    print()

    #the most optimal mutation 'passes on its genes'
    b = b_s[np.where(success_rates == success_rates.max())[0][0]]
    m = m_s[np.where(success_rates == success_rates.max())[1][0]]
    print('------------')


    #if all of the new mutations were less optimal than the previous generation,
    #the below code reduces the diversity of mutations to be closer to the previous
    #generation

    if np.where(success_rates == success_rates.max())[0][0] == 2:
        d1 = 1+0.5*np.abs(d1-1)

    if np.where(success_rates == success_rates.max())[1][0] == 2:
        d2 = 1+0.5*np.abs(d2-1)

#plotting the travel path of the evolved animal
for u in range(50):

    #plotting travel path
    pos_track = take_levy_steps(np.array([0,0]),500,b,m,750)
    f = np.random.uniform(low = -100,high = 100, size = 8)
    food = np.array([[f[0],f[1]],[f[2],f[3]],[f[4],f[5]],[f[6],f[7]]])
    r = np.random.uniform(low = 5,high = 20, size = 4)

    fig, ax = plt.subplots(figsize = (6,6))

    #plotting food
    for i in range(4):
        circle2 = plt.Circle((food[i,0],food[i,1]), r[i], color='green', alpha = 0.5)
        ax.add_artist(circle2)

    ax.set_xlim((-100, 100))
    ax.set_ylim((-100, 100))
    ax.plot(pos_track[:,0],
            pos_track[:,1],
            color = 'k',
            linewidth = 1.1,
            linestyle = 'dashed')
    #ax.axis('equal')
    ax.grid(color='k', linestyle='--', linewidth=0.5,alpha = 0.4)
    plt.show()
```

A.3 Connective Constant of the Union Jack Lattice

This section will first demonstrate the method of implementing a self-avoiding walk numerically, before moving onto the methods of constructing upper and lower bounds of the connective constant outlined in Section (2.2.2). Coding the self-avoiding walk can be done by simply adding the self-avoidance constraint to the simple random walk code. This constraint in pseudo code is 'If the co-ordinate of the next step exists in the array of already traversed co-ordinates, try a new step'. The code for this is given below:

Simulating the Self-Avoiding Walk:

```
#Simulating a self-avoiding walk on 2D square lattice

import numpy as np
import random
from matplotlib import pyplot as plt

def take_a_step(position):

    #walker takes a step

    steps = [[-1,0],[1,0],[0,1],[0,-1]]
    new_step = np.array(random.choice(steps))
    new_pos = position + new_step

    return new_pos


N = 30 #number of steps

steps_til_trap = np.array([])
for j in range(0,1): #this line allows for multiple SAWs to be produced at once

    x = np.array([[0,0]])
    pos = x
    pos_track = x
    a = 0
    for i in range(0,N):
        pos = pos_track[-1,:]
        pos = take_a_step(pos)

        #SELF - AVOIDANCE CONSTRAINT
        if np.any(np.all(pos_track == pos ,axis = 1) == True):
            continue
        else:
            pos_track = np.vstack((pos_track,pos))

plt.figure(figsize = (6,6))
for i in range(1,len(pos_track[:,0])):
   plt.plot([pos_track[i-1,0],
             pos_track[i,0]],
            [pos_track[i-1,1],
             pos_track[i,1]],
             color = 'k',
             linewidth = 2.5)
plt.axis('equal')
plt.grid(color='k', linestyle='--', linewidth=0.5,alpha = 0.4)
plt.show()
```

Moving onto the method of finder upper and lower bounds for the connective constant of the Union Jack lattice, Keston's Method of Irreducible bridges can be found in depth in Section (2.2.2.1), and Alm's Method can be found in depth in Section (2.2.2.2). The implementation of these methods required closely following the literature and making the necessary adjustments for the change in lattice. In particular, the Union Jack lattice has two vertex classes (see Section (2.2.2.2) for the definition) which meant that the construction of $G(m, n)$ required the enumeration of self-avoiding walks starting at both of the different vertex classes. This means that the implementation displays a more 'complicated' example of Alm's method, compared to the Euclidean lattice for example. It should also be noted that the code below does not explicitly demonstrate the final step of Keston's Method, which was computed manually for simplicity. To calculate the largest eigenvalue of $G(m, n)$ in Alm's method, the Power Method was implemented (see lines 98-109)[55].

The code for Alm's Method and Keston's Method of Irreducible bridges is given below:

Keston's Method of Irreducible Bridges:

```
1  #Kestons Method of Irreducible Bridges Applied to the Union Jack Lattice
2  
3  import numpy as np
4  from matplotlib import pyplot as plt
5  from tqdm import tqdm
6  
7  def take_a_step_bridges(saws):
8  
9      #This function returns the bridges of a given length on the Union Jack Lattice
10 
11     if saws[0,0,0] == 0:
        x_0 = 0.5
    else:
        x_0 = 1

    new_saws = np.array([[[]]]);
    for i in tqdm(range(0,saws.shape[0])):
        if np.floor(saws[i,-1,0]) - saws[i,-1,0] == 0:
            steps = [[0,1],
                     [1,0],
                     [0,-1],
                     [-1,0],
                     [0.5,0.5],
                     [0.5,-0.5],
                     [-0.5,-0.5],
                     [-0.5,0.5]]
            for j in steps:
                new_pos_ij = np.array([saws[i,-1] + j])
                if np.any(np.all(saws[i,:]==new_pos_ij,axis=1)==True) or new_pos_ij[0][0]<x_0:
                    continue
                else:
                    new_walk_ij = np.append(saws[i,:], new_pos_ij,axis = 0)
                    new_walk_ij = np.array([new_walk_ij])
                    if new_saws.shape[2] == 0:
                        new_saws = new_walk_ij
                    else:
                       new_saws = np.append(new_saws, new_walk_ij,axis = 0)
        else:
            steps =  [[0.5,0.5],
                     [0.5,-0.5],
                     [-0.5,-0.5],
                     [-0.5,0.5]]

            for j in steps:
                new_pos_ij = np.array([saws[i,-1] + j])

                if np.any(np.all(saws[i,:]==new_pos_ij,axis=1)==True) or new_pos_ij[0][0]<x_0:
                    continue
                else:
                    new_walk_ij = np.append(saws[i,:], new_pos_ij,axis = 0)
                    new_walk_ij = np.array([new_walk_ij])
                    if new_saws.shape[2] == 0:
                        new_saws = new_walk_ij
                    else:
                        new_saws = np.append(new_saws, new_walk_ij,axis = 0)
    c_n = int(new_saws.shape[0])
    return new_saws,c_n

#%%

N = 4    #length of bridge
```

```
b_n = [] #array to store the number of brigdes of length n
b_n_l = np.zeros((N,2*N))

#enumarating brigdes beginning at vertex class 0
for n in range(1,N):
    new_saws = np.array([[[0,0],[1,0]]])
    for i in range(0,n):
        new_saws,c_n = take_a_step_bridges(new_saws)

    K = []
    new_bridges = new_saws
    for k in range(0,new_saws.shape[0]):
        if np.any(new_saws[k,:n+1,0] > new_saws[k,n+1,0]):
            K = np.append(K,k)

    new_bridges = np.delete(new_bridges,K,0)
    for i in range(0,new_bridges.shape[0]):
        b_n_l[n,int(2*new_bridges[i,n+1,0]-1)] = b_n_l[n,int(2*new_bridges[i,n+1,0])-1]+1

    b_n = np.append(b_n,new_bridges.shape[0])

b_n1 = []
b_n_l1 = np.zeros((N,2*N))

#enumarating brigdes beginning at vertex class 1
for n in range(1,N):
    new_saws1 = np.array([[[0,0],[0.5,0.5]]])
    for i in range(0,n):
        new_saws1,c_n1 = take_a_step_bridges(new_saws1)

    K1 = []
    new_bridges1 = new_saws1
    for k in range(0,new_saws1.shape[0]):
        if np.any(new_saws1[k,:n+1,0] > new_saws1[k,n+1,0]):
            K1 = np.append(K1,k)

    new_bridges1 = np.delete(new_bridges1,K1,0)
    for i in range(0,new_bridges1.shape[0]):
        b_n_l1[n,int(2*new_bridges1[i,n+1,0])-1]=b_n_l1[n,int(2*new_bridges1[i,n+1,0])-1]+1
    b_n1 = np.append(b_n1,new_bridges1.shape[0])


b_n_l2 = b_n_l + b_n_l1

#summing the bridges of span l and similar length n
sums = np.zeros((b_n_l2.shape[0],1))
for i in range(b_n_l2.shape[0]):
    sums[i] = b_n_l2[i,:].sum()
```

Alm’s Method:

```
1  #Alms Method of Finding Upper Bounds Applied to the Union Jack Lattice
2  
3  #PLEASE NOTE : lines 78 and 91 in the code have been edited for
4  #              display purposes, a simple re-edit will make the
5  #              code once again usable for general use
6  
7  import numpy as np
8  from matplotlib import pyplot as plt
9  from tqdm import tqdm
10 
11 def take_a_step(current_walks):
12 
13     #This function enumarates the number self-avoiding walks of a given length
14 
15     new_valid_walks = np.array([[[]]]);
16     for i in range(0,current_walks.shape[0]):
17         if np.floor(current_walks[i,-1,0]) - current_walks[i,-1,0] == 0:
18             steps = [[0,1],
19                      [1,0],
20                      [0,-1],
21                      [-1,0],
22                      [0.5,0.5],
23                      [0.5,-0.5],
24                      [-0.5,-0.5],
25                      [-0.5,0.5]]
26             for j in steps:
27                 new_pos_ij = np.array([current_walks[i,-1] + j])
28                 if np.any(np.all(current_walks[i,:] == new_pos_ij ,axis = 1) == True):
29                     continue
30                 else:
31                     new_walk_ij = np.append(current_walks[i,:], new_pos_ij,axis = 0)
32                     new_walk_ij = np.array([new_walk_ij])
33                     if new_valid_walks.shape[2] == 0:
34                         new_valid_walks = new_walk_ij
35                     else:
36                        new_valid_walks = np.append(new_valid_walks, new_walk_ij,axis = 0)
37         else:
38             steps =  [[0.5,0.5],
39                      [0.5,-0.5],
40                      [-0.5,-0.5],
41                      [-0.5,0.5]]
42 
43             for j in steps:
44                 new_pos_ij = np.array([current_walks[i,-1] + j])
45 
46                 if np.any(np.all(current_walks[i,:] == new_pos_ij ,axis = 1) == True):
47                     continue
48                 else:
```

```
                    new_walk_ij = np.append(current_walks[i,:], new_pos_ij,axis = 0)
                    new_walk_ij = np.array([new_walk_ij])
                    if new_valid_walks.shape[2] == 0:
                        new_valid_walks = new_walk_ij
                    else:
                        new_valid_walks = np.append(new_valid_walks, new_walk_ij,axis = 0)
    c_n = int(new_valid_walks.shape[0])
    return new_valid_walks,c_n


def generate_G(n,m,new_walks0,new_walks0_m,new_walks1,new_walks1_m, F_m):

   #This function generates the G matrix described in Alms Method

    num00 = int(c_n0[m-1])                    #number of m-walks starting at node 0
    num01 = int(c_n0[N-1])                    #number of N-walks starting at node 0
    num10 = int(c_n1[m-1])                    #number of m-walks starting at node 1
    num11 = int(c_n1[N-1])                    #number of N-walks starting at node 1
    num20 = int(c_n0[m-1] + c_n1[m-1])        #number of m-walks starting at either node
    G = np.zeros((num20,num20))


    #finding number of n-step walks starting with a particular m-step walk and
    #ending with a translation of another m-step walk for a vertex class 0

    for k in range(0,num01):
        for j in range(0,num00):
            if np.all(new_walks0[k,:m+1,:] == new_walks0_m[j,:m+1,:]):
                for i in range(0,num20):
                    if np.all(new_walks0[k,N-m:,:]-new_walks0[k,N-m,:]+np.abs(np.floor(
                            new_walks0[k,N-m,:])-new_walks0[k,N-m,:])==F_m[i,:m+1,:]):
                        G[j,i] = G[j,i] + 1
                        break
                break

    #finding the number of n-step walks starting with a particular m-step walk and
    #ending with a translation of another m-step walk for a vertex class 1

    for k in range(0,num11):
        for j in range(0,num10):
            if np.all(new_walks1[k,:m+1,:] == new_walks1_m[j,:m+1,:]):
                for i in range(0,num20):
                    if np.all(new_walks1[k,N-m:,:]-new_walks1[k,N-m,:]+np.abs(np.floor(
                            new_walks1[k,N-m,:])- new_walks1[k,N-m,:]) == F_m[i,:m+1,:]):
                        G[j+num00,i] = G[j+num00,i] + 1
                        break
                break
    return G

def power_method(G,n):

    #This function utilises the power method of finding the largest
    #Eigenvale of a matrix

     G = np.matrix(G)
     v_k = np.matrix(np.random.rand(G.shape[1]))
     for i in range(0,n):
         G_k = v_k*G
         alpha_k = G_k.max()
         v_k = G_k*(1/alpha_k)
     return alpha_k

#%%

N_max = 5 #N-step walks to enumarate
M_max = 3 #M-step walks to enumarate

mu = np.zeros((N_max+1,M_max+1))
for b in tqdm(range(2,5)):
    for c in range(1,3):
        if c < b:
            N = b #N steps
            m = c
            new_walks0 = np.array([[[0,0]]])
            new_walks1 = np.array([[[0.5,0.5]]])
            c_n0 = np.zeros(N)
            c_n1 = np.zeros(N)

            for i in range(0,N):
                new_walks0,c_n0[i] = take_a_step(new_walks0)
                if i == m-1:
                    new_walks0_m = new_walks0

            for i in range(0,N):
                new_walks1,c_n1[i] = take_a_step(new_walks1)
                if i == m-1:
                    new_walks1_m = new_walks1

            F_m = np.append(new_walks0_m,new_walks1_m, axis = 0)

            G = generate_G(N,m,new_walks0,new_walks0_m,new_walks1,new_walks1_m, F_m)

            beta = 0 # shift coefficient for power method
            mu[N,m] = power_method(G - beta*np.identity(G.shape[1]),20)**(1/(N-m))
```

A.4 Narrow Escape Problem

The new method of analysis for the Narrow Escape Problem developed in Section (3.4) required the numerical construction of a Markov chain in order to compute the PDF, CDF and expectation of the escape time E_t. The implementation of this follows simple steps:

1. Construct system of vertices
2. Construct system of edges
3. Calculate transition matrix P
4. Calculate $\mathbb{E}(E_t)$ from Eq. (50) and method described in Section (3.4.2)

These steps can be seen in the following code:

Narrow Escape Problem Analysis:

```python
#Finding the expected escape time for the Narrow Escape Time by modelling as
#a Markov chain

import numpy as np
from tqdm import tqdm
from matplotlib import pyplot as plt

n = 5  #width/length of grid i.e. resoulation of discrete approximation

#constructing system of vertices
x_coords = np.linspace(0,n-1,n)
y_coords = np.linspace(0,n-1,n)
transition_matrix = np.zeros((n**2,n**2))
graph = np.zeros((n**2,2))

for j in range(0,n):
    for i in range(0,n):
        graph[n*j+i,0] = x_coords[i]
        graph[n*j+i,1] = y_coords[j]


#constructing set of edges and transition matrix
for k in range(n**2):
    neighbours=np.array(np.where(abs(graph[k,1]-graph[:,1])+abs(graph[k,0]-graph[:,0])<1.1))[0]
    neighbours = np.delete(neighbours, np.where(neighbours == k), axis=0)
    d = len(neighbours)
    for l in neighbours:
        transition_matrix[k,l] = 1/d

transition_matrix[0,:] = np.zeros((1,n**2))
transition_matrix[0,0] = 1

```

```

#calculating pdf,cdf and expectation for time of escape E_t
its = 1000 #up to 2*its steps
cdf = np.zeros((its,1))
pdf = np.zeros((its,1))
ratio_termdif = np.zeros((its-2,1))
expectation = 0

for i in range(its):
    tmatrix = np.matrix(transition_matrix)**(2*i+2*(n-1))
    cdf[i] = tmatrix[n**2-1,0]

for i in range(its-1):
    pdf[i+1] = cdf[i+1]-cdf[i]
    pdf[0] = cdf[0]

for i in range(its-2):
    ratio_termdif[i] =pdf[i+1]/pdf[i]

for i in range(its):
    expectation = expectation + (2*i+2*(n-1))*pdf[i]
```