SELF-AVOIDING WALK

Dissertation presented in partial fulfillment of the requirements for the degree

of

B.Sc. Statistics Hons.

by

Yenisi Das
Roll No. : 20-300-4-07-0458
Registration No. : A01-2112-0868-20

Under the guidance of
Prof. Debjit Sengupta

DEPARTMENT OF STATISTICS
ST. XAVIER'S COLLEGE (AUTONOMOUS),
KOLKATA

DECLARATION

I affirm that I identify all my sources and that no part of my dissertation paper uses unacknowledged materials.

Yenisi Das

Yenisi Das
Department of Statistics
St. Xavier's College (Autonomous), Kolkata

Date: 5th April 2023

ACKNOWLEDGEMENT

I have received a great deal of support and assistance during the course of working on my dissertation.

I would like to thank and acknowledge my supervisor Prof. Debjit Sengupta for his immense support. Additionally, I would like to thank all of the professors at St. Xavier's College, Kolkata who have assisted me in developing a research mindset, which enabled me to complete this project.

It was my pleasure to seek guidance from Prof. Siva Athreya, ICTS-TIFR, and the Indian Statistical Institute, Bangalore on the topic of my project.

Lastly, I would like to thank my parents for their supportive nature and constant guidance throughout my undergraduate studies.

Contents

1 Introduction

Self-avoiding walks have been extensively studied and analyzed in the scientific community due to their significant applications in the fields of physics, chemistry, and biology. They are an important model for describing the behavior of polymer chains, and protein folding, and are used in DNA conformational analysis [11] all of which have a similar characteristic of not allowing self-intersections. The main reference for this project is [9]. Further details can be found in [13], [5], [14], and [10].

We begin by understanding what a self-avoiding walk is through a simple example. Suppose you are at St. Xavier's College and you want to go to the Park Street Metro Station. While walking, you follow a rule that you must not revisit any intersection you have already passed through. You continue walking and turning at intersections until you have reached the metro station. Once you have reached it, we will say that you have created a self-avoiding walk from St. Xavier's College to the Park Street Metro Station.

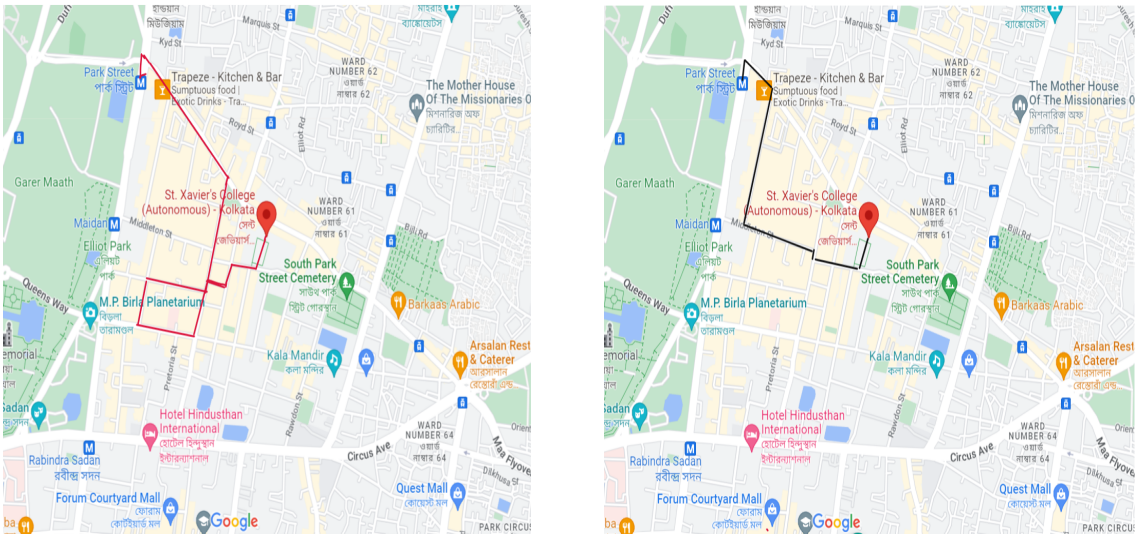

Figure 1: A self-avoiding walk (right) and a walk with intersections (left) to the metro station

There are two questions one may be interested in:

a) How many self-avoiding walks can you choose from? (so as to minimize time and distance)

b) How can you generate a random self-avoiding walk for yourself? (If there are many, how to choose one?)

In this article, we will address two related questions. First, we show the number of self-avoiding walks of length n grows exponentially with n. Secondly, we describe an algorithm called the pivot algorithm through which one can generate a self-avoiding walk from another self-avoiding walk.

We begin a discussion on how self-avoiding walks are used as models in statistical physics and chemistry.

1.1 Motivation and Literature

Self-avoiding walks have emerged as a powerful tool for modeling the behavior of polymer chains. A polymer is composed of numerous 'monomers', or groups of atoms, linked by chemical bonds. The functionality of a monomer refers to the number of chemical bonds available, indicating the number of monomers it must bond with. If each monomer has functionality two, it results in a linear polymer.

Linear polymers can be enormous, comprising over 10^5 monomers, making the length scale of the entire polymer macroscopic, A linear polymer consisting of $N+1$ monomers can be labeled from one end to the other as $0, 1, \ldots, N$. The location of the i^{th} monomer can be denoted by $x(i) \in \mathbb{R}^3$, with the i^{th} bond represented by the line segment joining $x(i-1)$ to $x(i)$. Usually, the length of each bond is roughly constant throughout the chain, along with the angle between each consecutive monomer-monomer bond.

A possible model for the spatial configuration of a linear polymer is a random walk in $\mathbb{Z}^3$, known as the ideal polymer chain. The random walk approximation for polymers was proposed around 80 years ago by a German chemist called Kuhn [8] and a Hungarian physicist Guth [3] independently. Years after this assumption was proven incorrect. A new answer to the questions about this model was discovered by the Nobel laureate Flory [2] and Orr [12] who suggested that while the random walk tends to trap themselves, the monomers try to bounce away from each other. The ideal polymer chain had a fundamental limitation, namely the excluded volume effect.

The presence of a monomer at position x prohibits any other part of the polymer from getting too close to x, meaning that other monomers are excluded from a certain volume of space. This is the excluded volume effect. Taking this effect into account reveals that a self-avoiding walk is a more suitable model for a linear polymer than a random walk. The self-avoiding walk model is ideal for the case of a dilute polymer solution, where the polymers are far apart, leading to little interaction between distinct molecules, and a good solvent, which minimizes attractive forces between monomers.

Polymer scientists are interested in understanding the behavior of the set of self-avoiding walks as the number of monomers grows toward infinity. While real polymers exist in continuous space, it turns out that the behavior of self-avoiding walks on a lattice is a good approximation for large polymers, and studying this simpler mathematical model can give valuable insights into the properties of real polymers.

Layout: The rest of the article is organized as follows. We shall begin in the next section (2) by defining the model precisely and stating the main result (Theorem 2.2) that specifies the growth rate of the self-avoiding walk. We then proceed to bound the number of the self-avoiding walk for a given n and discuss the universality of the critical exponents. Finally, we describe the pivot algorithm and discuss Theorem 2.7 stating that the algorithm is ergodic. In the next section (3), we prove the subadditive nature of self-avoiding walks (Proposition 2.3) and the main result (Theorem 2.2). In the last section (4), we look into the pivot algorithm and implement it to generate a self-avoiding walk of a specified length from a self-avoiding walk in 2 dimensions.

2 Preliminaries and Main Result

Let $\mathbb{Z}^d$ be the integer lattice. We now begin by defining the self-avoiding walk:

Definition 2.1. *An N-step self-avoiding walk can be defined as a function $\omega : \{0, 1, ..., N\} \to \mathbb{Z}^d$, beginning at the site x, i.e. $\omega(0) = x$, satisfying*

$$|\omega(j+1) - \omega(j)| = 1, and \;\; \omega(i) \neq \omega(j) \,\forall\, i \neq j$$

. We write $|\omega| = N$ to denote the length of ω, and we denote the components of $\omega(j)$ by $\omega_i(j)$ where $i = 1, ..., d$. Let C_N denote the number of N -step self-avoiding walks beginning at the origin. By convention, $C_0 = 1$.

In one dimension, a self-avoiding walk has no alternative but to continue traveling in the direction initially chosen, so there are exactly two paths for every value of N.

We now state a theorem based on the definitions above, in order to understand how self-avoiding walks of a specified length n grow with increasing n.

Theorem 2.2. *: As $n \to \infty$, $\frac{a_n}{n} \to c$ i.e.*

$$\lim_{n\to\infty} \frac{a_n}{n} = c$$

where $a_n = \log C_n$. The limit, $\lim_{n\to\infty} C_n{}^{\frac{1}{n}}$ is known as the connective constant, μ. This shows that as n increases the total number of self-avoiding walks of length n grows exponentially.

The limit μ was first shown to exist by Hammersly and Morton [4]. Roughly, C_n is of order μ^n for large n, so μ is the average number of possible next steps for a long self-avoiding walk. In order to prove the above theorem, we shall be using the subadditive property of self-avoiding walks as stated below:

Proposition 2.3. *The self-avoiding walk is subadditive, i.e.,* $\log C_{N+M} \leq \log C_M + \log C_N$.

We now discuss some bounds of C_n, i.e., the total number of self-avoiding walks of length n.

Lower Bound: Consider the set of self-avoiding walks in the d-dimensional integer lattice space which, for each axis, only take steps in positive directions. These are all obviously self-avoiding, and at each point, there are d possible choices of directions, so there are d^n such self-avoiding walks. Thus, $C_n \geq d^n$.

Upper Bound: Now consider all walks of length n that never return to the site they were at in the previous step, that is, the set of all walks for which $\omega(j-1) \neq \omega(j+1)$ for $j = 1, ..., n-1$ Then at each site after the initial one, there are $2d-1$ possible choices, and therefore there are $2d{\cdot}(2d-1)^{n-1}$ such walks. All self-avoiding walks have this property, so there are at most $2d{\cdot}(2d-1)^{n-1}$ self-avoiding walks.

Therefore we have $d^n \leq C_n \leq 2d{\cdot}(2d-1)^{n-1}$.

Note that the self-avoiding walk is not a Markovian stochastic process because it does not satisfy the properties of a process. A Markov process forgets its past and depends only

on the current state, but a self-avoiding walk must remember its entire history to ensure the self-avoidance constraint is satisfied. Additionally, a self-avoiding walk may become trapped and impossible to extend further. That is why the precise value of μ is hard to calculate in any dimension. A list of estimated values of μ for some lattices from [6] and [13] are given below:

Lattice	μ
$\mathbb{Z}^2$	2.638
$\mathbb{Z}^3$	4.683
$\mathbb{Z}^4$	6.772
$\mathbb{Z}^5$	8.838
$\mathbb{Z}^6$	10.878
Hexagonal	$\sqrt{(2+\sqrt{(2)})}$
Trihexagonal	2.560

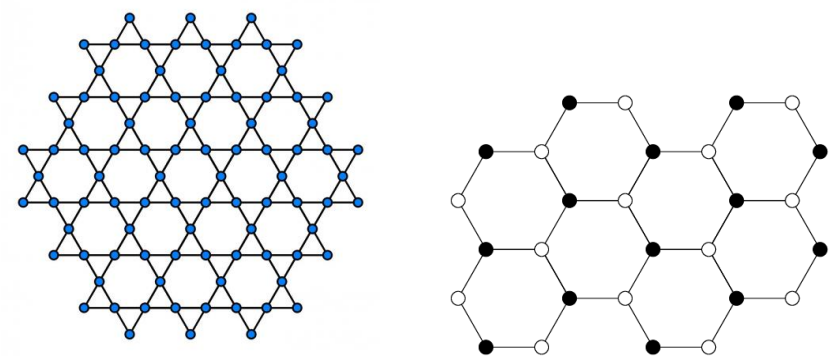

Figure 2: Trihexagonal (kagome) lattice (left) and Hexagonal (honeycomb) lattice (right)

Definition 2.4. *In statistical mechanics, universality is the observation that there are properties for a large class of systems that are independent of the dynamical details of the system such as the lattice in the context of self-avoiding walks.*

Note that the connective constant μ is not universal-it depends on the details of the allowed steps and the underlying lattice. The conjectured behaviour of C_n is $C_n \sim A\mu^n n^{\gamma-1}$ where $\sim$ is defined as follows:

Let $f(x)$ and $g(x)$ be functions. We say that $f(x)$ is asymptotic to $g(x)$, written $f(x) \sim g(x)$, if $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$.

Here, γ is known as the critical exponent.

It is widely believed that the critical exponent γ depends on the dimension, not the specific type of allowed steps. As long as the number of possible steps is finite and the allowed steps are symmetric, the critical exponent remains invariant, regardless of the lattice type. This characteristic is known as universality and the models having the same critical exponents are said to be in the same universality class.

The values of γ for different dimensions taken from [13] and [14] are given below:

Dimension	γ
2	43/32
3	1.162
4	$C_n \sim A\mu^n(\log n)^{1/4}$
≥ 5	1

Next, we discuss an algorithm called the pivot algorithm [10] in an attempt to generate a self-avoiding walk. It is a type of dynamic Monte Carlo algorithm that generates new configurations of the walk by applying random symmetry operations to the existing walk.

Definition 2.5. *The pivot algorithm is a dynamic Monte Carlo algorithm, which generates SAWs in a canonical ensemble (fixed number of steps N) with free endpoints. The elementary move of the algorithm is as follows: A site on the walk is chosen at random and used as a pivot point; a random symmetry operation of the lattice (e.g., rotation or reflection) is applied to the part of the walk after the pivot point, using the pivot point as the origin. The resulting walk is accepted if it is self-avoiding; otherwise, it is rejected, and the old walk is counted once again in the sample.*

We now look at the definition of ergodicity to understand the following theorem.

Definition 2.6. *Ergodicity is the idea that a point of a moving system, either a dynamical system or a stochastic process, will eventually visit all parts of the space that the system moves in, in a uniform and random sense.*

Next, we state a theorem stating that the pivot algorithm is ergodic. In other words, the ergodicity of self-avoiding walks means that if we run the pivot algorithm for a long enough time, it will eventually generate all possible configurations of the walk, and each configuration will be generated with the same probability. It is proved in section 3.5 of [10].

Theorem 2.7. *The pivot algorithm is ergodic for self-avoiding walks on $\mathbb{Z}^d$ provided that all axis reflections, and either all 90° rotations or all diagonal reflections are given nonzero probability. In fact, any N-step self-avoiding walk can be transformed into a straight rod by some sequence of 2N-1 or fewer such pivots.*

3 Proof of Theorem 2.2

In this section, we prove Theorem 2.3. In order to do so, we first prove Proposition 2.4. We do this by proving the following Lemma first and then defining the concatenation of two self-avoiding walks. The Lemma 3.1 states that a function $f : A \to B$ with a left inverse is necessarily injective.

Lemma 3.1. *Let A and B be non-empty sets and $f : A \to B$ be a function. Then if f has a left inverse it is injective.*

Proof of Lemma 3.1. Let g be the left inverse of f. We wish to show that f is injective. In other words, we wish to show that $f(x) = f(y) \implies x = y$

If $f(x) = f(y)$, then we know that $g(f(x)) = g(f(y))$. By definition of g, we have $x = g(f(x))$ and $g(f(y)) = y$. Putting this together, we have $x = g(f(x)) = g(f(y)) = y$ as required. □

We now define the concatenation of two self-avoiding walks of length N and M respectively to form a walk of $N + M$ steps as follows:

Definition 3.2. *The concatenation $\omega^{(1)} o \omega^{(2)}$ of an M -step self-avoiding walk $\omega^{(2)}$ to an N -step self-avoiding walk $\omega^{(1)}$ is the $(N + M)$-step walk ω, which in general need not be self-avoiding, given by*

$$\omega(k) = \omega^{(1)}(k),\ k = 0, ..., N$$

$$\omega(k) = \omega^{(1)}(N) + \omega^{(2)}(k - N) - \omega^{(2)}(0),\ k = N + 1, ..., N + M.$$

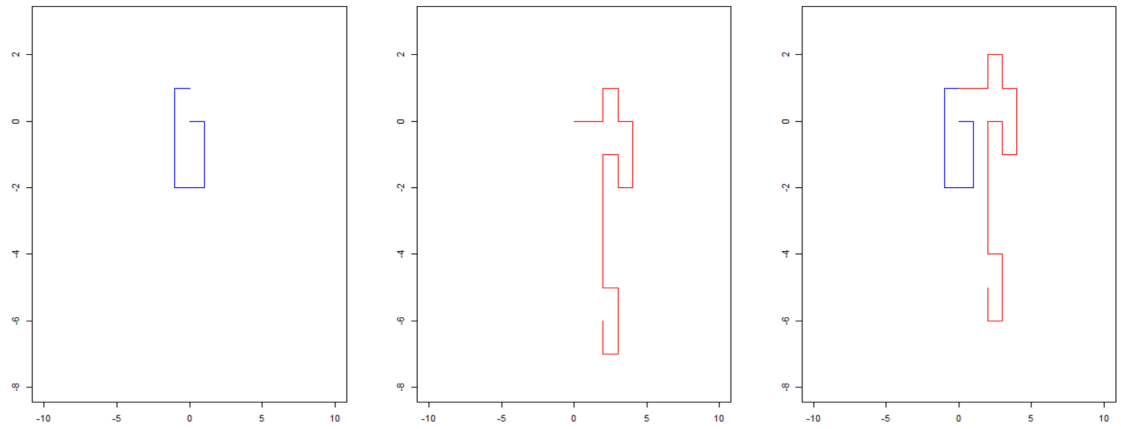

Figure 3: Concatenation of a SAW of length 10 to a SAW of length 20 creating a SAW of length 30

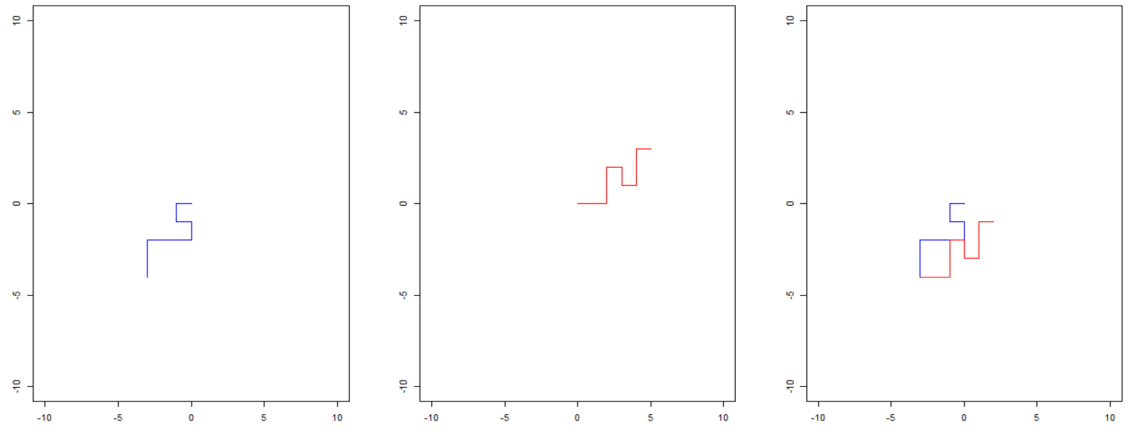

Figure 4: Concatenation of a SAW of length 10 to a SAW of length 10 resulting in intersections

Now we prove Proposition 2.4 using the following intuitive idea: Every self-avoiding walk of length $N + M$ may be split into two smaller self-avoiding walks; one of length N and one of length M but the concatenations of every N and M length walk is not self-avoiding.

Proof of Proposition 2.3. : Let S_{N+M}, S_N, and S_N be the set of self-avoiding walks of length $N + M, N$, and M respectively. Consider the function $f : S_{N+M} \to (S_N \times S_M)$ defined by splitting the walk after N steps and sending the N step walk to S_N and the M step walk to S_M. This function is injective since concatenation yields a left inverse. Therefore $C_{N+M} = |S_{N+M}| \leq |S_N||S_M| = C_N C_M$. □

Let $a_N = \log C_N$. We have $\log C_{N+M} \leq \log C_M + \log C_N$ i.e.

$$a_{N+M} \leq a_N + a_M - (i).$$

Thus, the sequence $\{a_N\}$ is sub-additive in nature.

Remark 3.3. *A lattice in the real coordinate space $\mathbb{R}^{\times}$ is an infinite set of points in this space with the properties that coordinate-wise addition or subtraction of two points in the lattice produces another lattice point, that the lattice points are all separated by some minimum distance, and that every point in the space is within some maximum distance of a lattice point.*

Note that the sub-additive property of self-avoiding walks is also true for other lattices.

Finally, we prove Theorem 2.2 using Proposition 2.3. Here is a plan of the proof. Using the subadditivity property, in Claim 3.4, we establish that for $n = mq + r$ where n is the length of the walk and m is a fixed integer, $\frac{a_n}{n} \leq \frac{qa_m}{n} + \frac{a_r}{n}$. In Claims 3.5 and 3.6, we show that as $n \to \infty$, $\frac{qa_m}{n}$ goes to $\frac{a_m}{m}$ and $\frac{a_r}{n}$ goes to 0. Finally in Claim 3.7, we prove that the largest limit point of $\frac{a_n}{n}$ is less than or equal to the largest limit point of $\frac{qa_m}{n} + \frac{a_r}{n}$. Using these claims, we show that as $n \to \infty$, $\frac{a_n}{n}$ goes to $\inf\{\frac{a_m}{m} | m \geq 1\}$ which we call c.

Proof of Theorem 2.2. : Here we show that as n increases, the number of self-avoiding walks of length n grows exponentially. We prove this using the following four claims.

Claim 3.4. *:* $a_{kn} \leq ka_n$

Proof. : Base case. For $k = 2, a_{2n} \leq 2a_n$ from (i)
Let the claim be true for $k = p$, i.e., $a_{pn} \leq pa_n$
Proof for $k = p+1$: We know $a_{(p+1)n} = a_{pn+n} \leq a_{pn}+a_n$ Again $a_{pn}+a_n \leq pa_n+a_n = (p+1)a_n$ Therefore, $a_{kn} \leq ka_n \forall\, k \geq 1$
Let $n = mq + r$ where m is a fixed integer and $0 \leq r \leq m-1$ Now, $a_n = a_{mq+r} \leq a_{mq} + a_r \leq qa_m + a_r$, therefore,

$$\frac{a_n}{n} \leq \frac{qa_m}{n} + \frac{a_r}{n} - (1)$$

□

Claim 3.5. *: As* $n \to \infty$*,*

$$\frac{qa_m}{n} \to \frac{a_m}{m}$$

Proof. : To show: $|\frac{q}{n} - \frac{1}{m}| < \epsilon$ whenever $n \geq N$

$$|\frac{q}{n} - \frac{1}{m}| = |\frac{n-r}{mn} - \frac{1}{m}| = |\frac{1}{m} - \frac{r}{mn} - \frac{1}{m}| = |\frac{r}{mn}| \leq \frac{1}{n} \quad since \quad \frac{r}{m} \leq 1$$

.
Now $\frac{1}{n} < \epsilon$ whenever $n \geq [\frac{1}{\epsilon}] + 1$.
So, we choose $N = [\frac{1}{\epsilon}] + 1$

□

Claim 3.6. *:* $As \quad n \to \infty, \quad \frac{a_r}{n} \to 0$

Proof. : We know $a_r \leq \max\{a_1, a_2, ..., a_{m-1}\} = k \quad \forall \quad n \geq 1$
To show: $|\frac{k}{n} - 0| < \epsilon$ whenever $n > N$. Choose $N = [\frac{k}{\epsilon}] + 1$.
Now, let

$$\alpha_n = \frac{a_n}{n}, \beta_n = \frac{qa_m}{n} + \frac{a_r}{n}$$

From (1), $\alpha_n \leq \beta_n$
Thus α_n and β_n are bounded.

□

Limit point of a sequence: A real number p is said to be the limit point of a sequence $\{S_n\}_{n\geq 1}$ if for a given $\epsilon > 0, S_n \in (p-\epsilon, p+\epsilon)$ for infinitely many values of n.

Claim 3.7. *: Largest limit point of* $\alpha_n \leq$ *largest limit point of* β_n

Proof. : Let the largest limit point of β_n be L.

Subclaim: Since there are no limit points of β_n above L, for $\epsilon \geq 0 \; \exists \; N$ such that

$$\beta_n \leq L + \epsilon \; \forall \, n \geq N$$

Proof: Let there be infinitely many $\beta_n s$ above $L + \epsilon$. Then for every $m \in \mathbb{N}$, the sequence $\{\beta_n\}$ where $n > m$ will have a β_n such that $\beta_n > L + \epsilon$. Consider $x_1, x_2, ...$ to be the supremum of the sequences $\{\{\beta_n\}, \{\beta_{n+1}\}, ...\}$ respectively. Then, $x_i > L + \epsilon \; \forall \; i$ But, L=largest limit point of $\beta_n = \lim_{n\to\infty} x_i > L + \epsilon$. We get a contradiction, so our assumption must be false.

Since $\alpha_n \leq \beta_n$ there are finitely many $\alpha_n s$ above $L + \epsilon$. So, there are no limit points of α_n above $L + \epsilon$, and since ϵ is arbitrary, therefore, Largest limit point of $\alpha_n \leq$ largest limit point of β_n.

□

From claims 2 and 3, we have $\lim_{n\to\infty} \beta_n = \frac{a_m}{m}$ where $m \geq 1$ is fixed.
Therefore, the largest limit point of $\alpha_n \leq \frac{a_m}{m}$
The largest limit point of $\alpha_n \leq \inf\{\frac{a_m}{m} | m \geq 1\}$
The smallest limit point of $\alpha_n \geq \inf\{\frac{a_m}{m} | m \geq 1\}$
From the above, the smallest limit point of $\alpha_n \geq$ largest limit point of α_n.
Therefore, smallest limit point = largest limit point of α_n =$\inf\{\frac{a_m}{m} | m \geq 1\} = c$ (say)
Hence, $\frac{a_n}{n} \to c$ as $n \to \infty$ So,

$$\frac{\log C_n}{n} \to c \; as \; n \to \infty$$

□

Thus we can say that C_n increases exponentially with increasing n.
Here, $\lim_{n\to\infty} C_n^{1/n}$ is known as the connective constant μ.

4 Generating a self-avoiding walk

4.1 Pivot Algorithm

We shall be using the pivot algorithm [10] to generate self-avoiding walks from a self-avoiding walk. The idea behind the pivot algorithm is to start with a self-avoiding walk and then make small perturbations to its shape by rotating segments and reflecting segments around pivot points while ensuring that the resulting path remains self-avoiding. In 2 dimensions, the following symmetry operations are needed:

4.1.1 Symmetry Operations in 2D

g_0: Identity- There is no change in the walk step directions.

g_1: Rotation by $-\pi/2$

g_2: Rotation by $-\pi$. We have to change every up step to a down step and every left step to the right step and vice versa.

g_3: Rotation by $-\frac{3\pi}{2}$

g_4: Reflecting on the x-axis. By reflecting our walk on the x-axis we change every up step to a down step and every down step to an up step.

g_5: Reflecting on the y-axis. By reflecting our walk on the y-axis we change every left step to a right step and every right step to a left step.

g_6: Reflecting on the diagonal line $x = y$

g_7: Reflecting on the diagonal line $x = -y$

4.1.2 Algorithm

The pivot algorithm is implemented as follows:

1. First a self-avoiding walk is created at random.

2. A random pivot point k along the walk $(0 \leq k \leq N-1)$ is chosen according to any preset strictly positive probabilities $p_0, ..., p_{N-1}$. Here, we consider the uniform distribution $(p_k = \frac{1}{N} \forall\, k)$.

3. We chose a symmetry operation $g_j \in G$ again according to any preset strictly positive probability. Now we apply this operation to the walk from the pivot point k onwards. Since we have only 8 possible operations in 2 dimensions and the operation g_0 (identity operation) can be ignored, the probability for any operation is again the uniform distribution with $p_g = 1/7$.

4. Now we check if our generated walk is still valid. If it is, then we find another self-avoiding walk. Otherwise, the walk is discarded and we start again from 2., with the last valid walk.

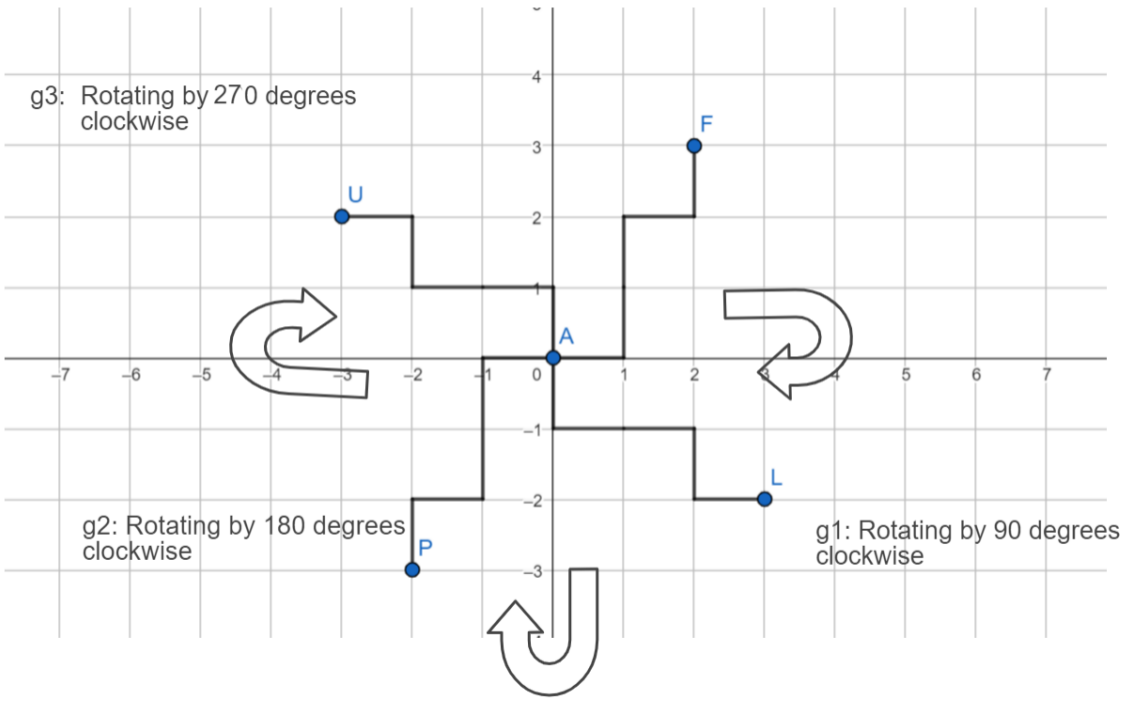

Figure 5: Rotations of a SAW of length 5

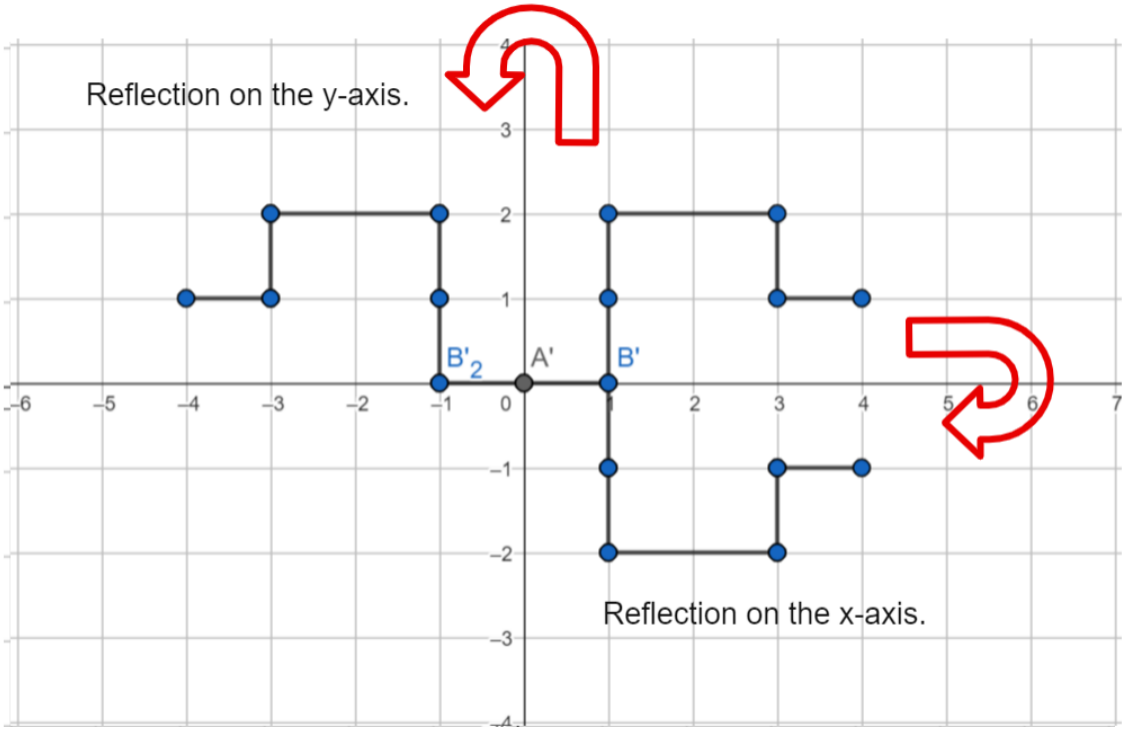

Figure 6: Reflections of a SAW of length 7 about the x and y axes.

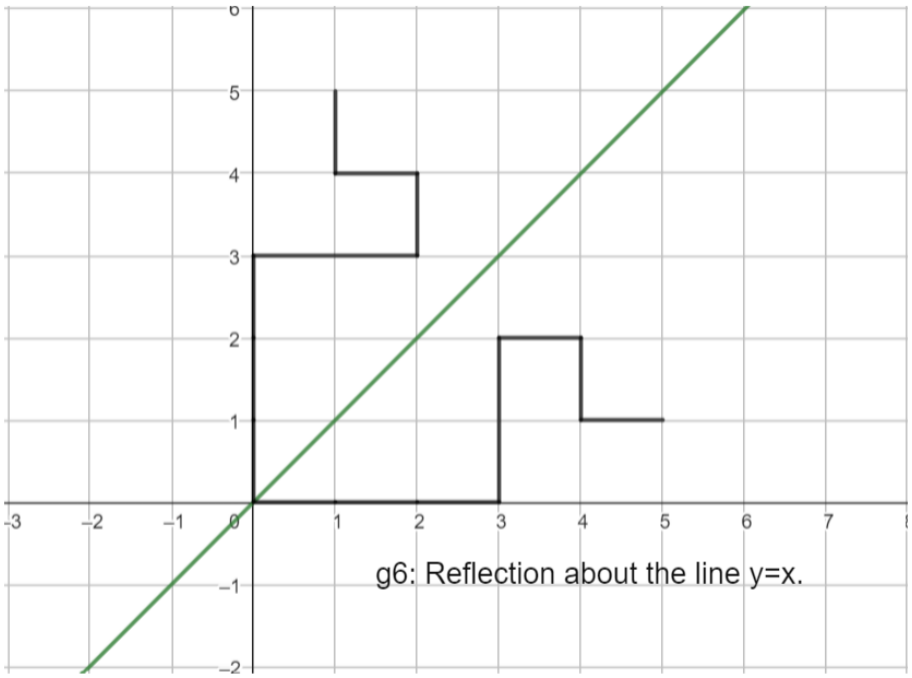

Figure 7: Reflections of a SAW of length 8 about $y = x$

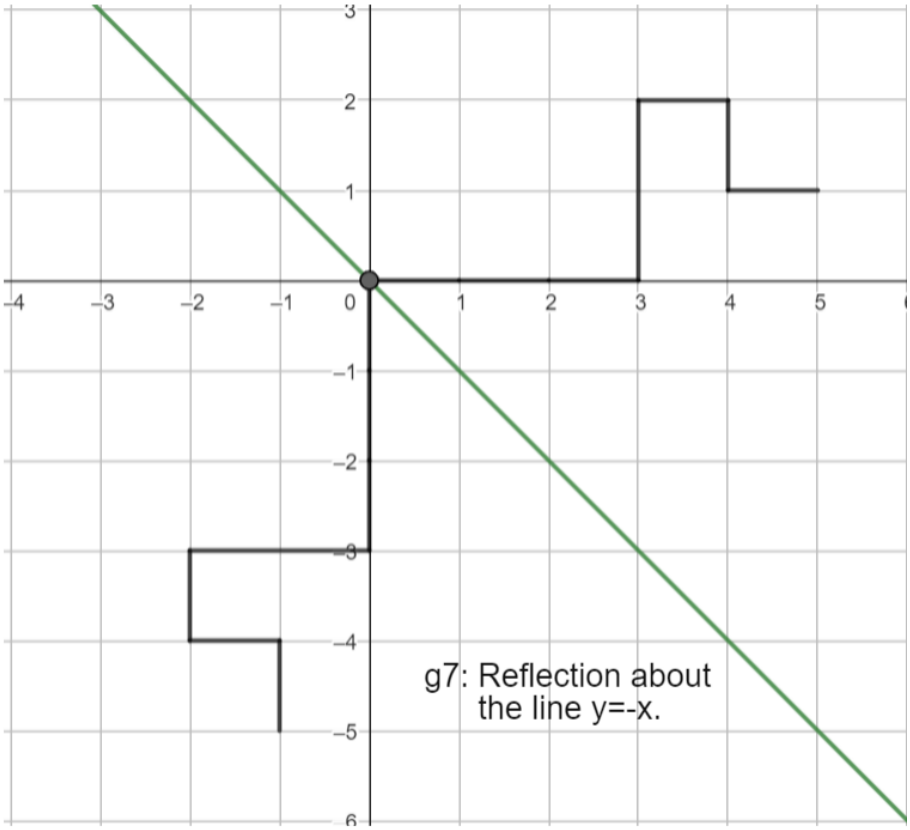

Figure 8: Reflections of a SAW of length 8 about $y = -x$

4.2 Reasons why the pivot algorithm is efficient

1. The algorithm relies on a set of local move rules that ensure that at least some of the generated paths remain self-avoiding [1]. These move rules involve rotating segments or reflecting segments around pivot points in such a way that the path remains confined to a fixed lattice spacing or mesh, which greatly reduces the computational overhead compared to more general techniques.

2. The pivot algorithm enables a large number of pivot moves to be applied to different segments of a path at the same time, without compromising the self-avoiding property of the path. This results in a more efficient generation process, as multiple perturbations can be made simultaneously. Moreover, multiple independent self-avoiding walks can be generated and counted concurrently, which helps deal with the increasing number of possible walks as the length of the path increases.

4.3 Code in R

Functions used

1. is.self.avoiding(walk): This function takes a matrix walk as input and checks if there are any duplicate rows, returning TRUE if there are none and FALSE if there are duplicates.

2. check.matrix.row(matrix, row): This function takes a matrix 'matrix' and a row 'row' as input and checks if any rows in the matrix are identical to the given row. It returns the index of the first row that matches or 0 if there are no matches.

3. move(pos, orient): This function takes a position pos and an orientation orient as input and returns a new position that is shifted one unit in the given direction. The orientation can be 'U' (up), 'D' (down), 'R' (right), or 'L' (left).

4. saw(steps): This function generates a self-avoiding random walk of steps 'steps'. It starts from the origin (0,0) and uses the move function to generate new positions. At each step, it checks which directions are available to move (i.e., not occupied by previous positions) and randomly chooses one of them. If no directions are available, it backtracks to the last available position and continues from there.

```r
rm(list=ls())
is_self_avoiding <- function(walk) {
  !any(duplicated(walk))
}


# Check if a matrix contains a specific row
check_matrix_row <- function(matrix, row) {
  matches = apply(matrix, 1, function(x) all(x == row))
  if (any(matches)) {
    return(which(matches))
  } else {
    return(0)
  }
```

```
}

# Moves the last point by a direction (U, D, R, L)
move <- function(pos, orient) {
  switch(orient,
         U = pos + c(0, 1),
         D = pos + c(0, -1),
         R = pos + c(1, 0),
         L = pos + c(-1, 0))
}
n=500
saw <- function(steps = n) {
    k = 0
    path = matrix(0, nrow = 1, ncol = 2) # starting point = c(0, 0)
    i = 1

    while (i < steps) {
        k = k + 1

        # calculates positions after all possible movements
        possible_pos = rbind(move(path[i, ], 'U'),
                             move(path[i, ], 'D'),
                             move(path[i, ], 'R'),
                             move(path[i, ], 'L'))

        # checks which directions are available
        avail_direction = apply(possible_pos, 1,
                                function(possible_pos)
                        check_matrix_row(path, possible_pos))
        avail = ifelse(avail_direction > 0, 0, 1)
        names(avail) = c('U', 'D', 'R', 'L')

        # if no direction is available, go back
        if (all(avail == 0)) {
            goback = ifelse(i > 5, rbinom(1, 10, 0.5) +
            rbinom(1, i, 0.05), 0) + 1
            path = path[1:(i - goback), ]
            i = i - goback - 1
            } else {
                # if one or more orientations are available, sample one
                #uniformly and move to it
                chosen_direction = sample(x = names(avail), size = 1,
                prob = avail)
                path = rbind(path, move(path[i, ], chosen_direction))
```

```
            }
        i = i + 1
    }
    list(path = path, total_iterations = k)
}

# Test

walk1 = saw(steps = n)
pivot = sample(2:(n-2), size = 1)
newchain = walk1$path[1:pivot,]
colnames(newchain) = c("c1","c2")
tempchain = walk1$path[(pivot+1):n,]

tempmat = data.matrix(tempchain)

rotateby=sample(c(-pi/2, -pi, -3*pi/2, 1, 2, 3, 4), size=1)

if( rotateby == -pi/2 || rotateby == -pi || rotateby == -3*pi/2)
{
c11 = tempmat[-1,1] - tempmat[1,1];c11
c22 = tempmat[-1,2] - tempmat[1,2];c22
c1 = c11*cos(rotateby) - c22*sin(rotateby) +tempmat[1,1];c1
c2 = c11*sin(rotateby) + c22*cos(rotateby) +tempmat[1,2];c2
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
walk = rbind(newchain,tempchain[1,],newchain2)
}

#reflection on the x-axis
if (rotateby == 1)
{
c2= tempmat[-1,1]-tempmat[1,1]+tempmat[1,2];c2
c1 = -(tempmat[-1,2]-tempmat[1,2])+tempmat[1,1];c1
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
walk = rbind(newchain,tempchain[1,],newchain2)
}

#reflection on the y axis
if (rotateby == 2)
{
c2= -(tempmat[-1,1]-tempmat[1,1])+tempmat[1,2];c2
c1 = tempmat[-1,2]-tempmat[1,2]+tempmat[1,1];c1
```

```
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
walk = rbind(newchain,tempchain[1,],newchain2)
}

#reflection on x=y
if (rotateby == 3)
{
c2= tempmat[-1,1]-tempmat[1,1]+tempmat[1,2];c2
c1 = tempmat[-1,2]-tempmat[1,2]+tempmat[1,1];c1

newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
walk = rbind(newchain,tempchain[1,],newchain2)
}

#reflection on the x=-y
if (rotateby == 4)
{
c1 = -(tempmat[-1,1] - tempmat[1,1])+tempmat[1,1];c1
c2 = -(tempmat[-1,2] - tempmat[1,2])+tempmat[1,2];c2
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
walk = rbind(newchain,tempchain[1,],newchain2)
}

k = 1
while (is_self_avoiding(walk)==FALSE ) {
  # Choose a random pivot

  pivot = sample(2:(n-2), size = 1,replace=F)
  k = k+1
  # Perform pivot move
  new_chain = walk[1:pivot, ]
  temp_chain = walk[(pivot + 1):n, ];temp_chain
  tempmat = data.matrix(temp_chain);tempmat
rotateby=sample(c(-pi/2, -pi, -3*pi/2, 1, 2, 3, 4), size=1);rotateby

if( rotateby == -pi/2 || rotateby == -pi || rotateby == -3*pi/2)
{
c11 = tempmat[-1,1] - tempmat[1,1];c11
c22 = tempmat[-1,2] - tempmat[1,2];c22
c1 = c11*cos(rotateby) - c22*sin(rotateby) +tempmat[1,1];c1
c2 = c11*sin(rotateby) + c22*cos(rotateby) +tempmat[1,2];c2
```

```
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
new_walk = rbind(new_chain,temp_chain[1,],newchain2);new_walk
}


#reflection on the x-axis
if (rotateby == 1)
{
c2= tempmat[-1,1]-tempmat[1,1]+tempmat[1,2];c2
c1 = -(tempmat[-1,2]-tempmat[1,2])+tempmat[1,1];c1
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
new_walk = rbind(new_chain,temp_chain[1,],newchain2);new_walk
}

#reflection on the y axis
if (rotateby == 2)
{
c2= -(tempmat[-1,1]-tempmat[1,1])+tempmat[1,2];c2
c1 = tempmat[-1,2]-tempmat[1,2]+tempmat[1,1];c1
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
new_walk = rbind(new_chain,temp_chain[1,],newchain2);new_walk
}

#reflection on x=y
if (rotateby == 3)
{
c2= tempmat[-1,1]-tempmat[1,1]+tempmat[1,2];c2
c1 = tempmat[-1,2]-tempmat[1,2]+tempmat[1,1];c1
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
new_walk = rbind(new_chain,temp_chain[1,], newchain2);new_walk
}

#reflection on the x=-y
if (rotateby == 4)
{
c1 = -(tempmat[-1,1] - tempmat[1,1])+tempmat[1,1];c1
c2 = -(tempmat[-1,2] - tempmat[1,2])+tempmat[1,2];c2
newmat1 = cbind(c1,c2);newmat1
newchain2 = data.frame(newmat1);newchain2
new_walk = rbind(new_chain,temp_chain[1,],newchain2);new_walk
```

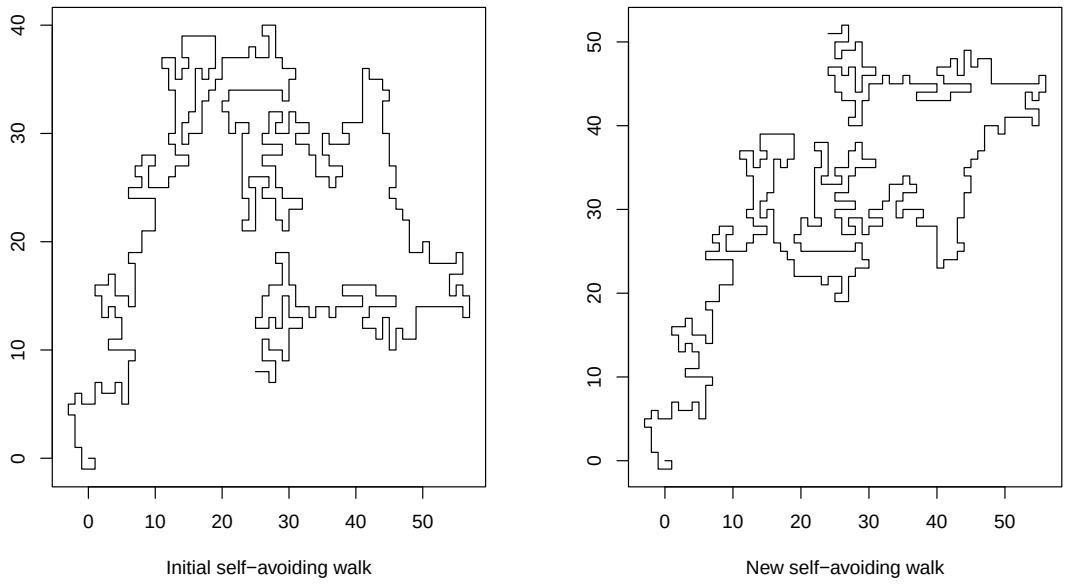

Figure 9: Self-Avoiding walks of length 500

```
}

  # Check if new walk is self-avoiding
  if (is_self_avoiding(new_walk) == TRUE) {
    walk = new_walk
  }
}

# Plot the resulting walks
par(mfrow = c(1,2))
plot(walk1$path,type='l', ylab=NA, xlab="Initial self-avoiding walk")
xrange = range(walk[ , 1])
yrange = range(walk[ , 2])
plot(walk,type ='l', ylab=NA, xlab="New self-avoiding walk")
```

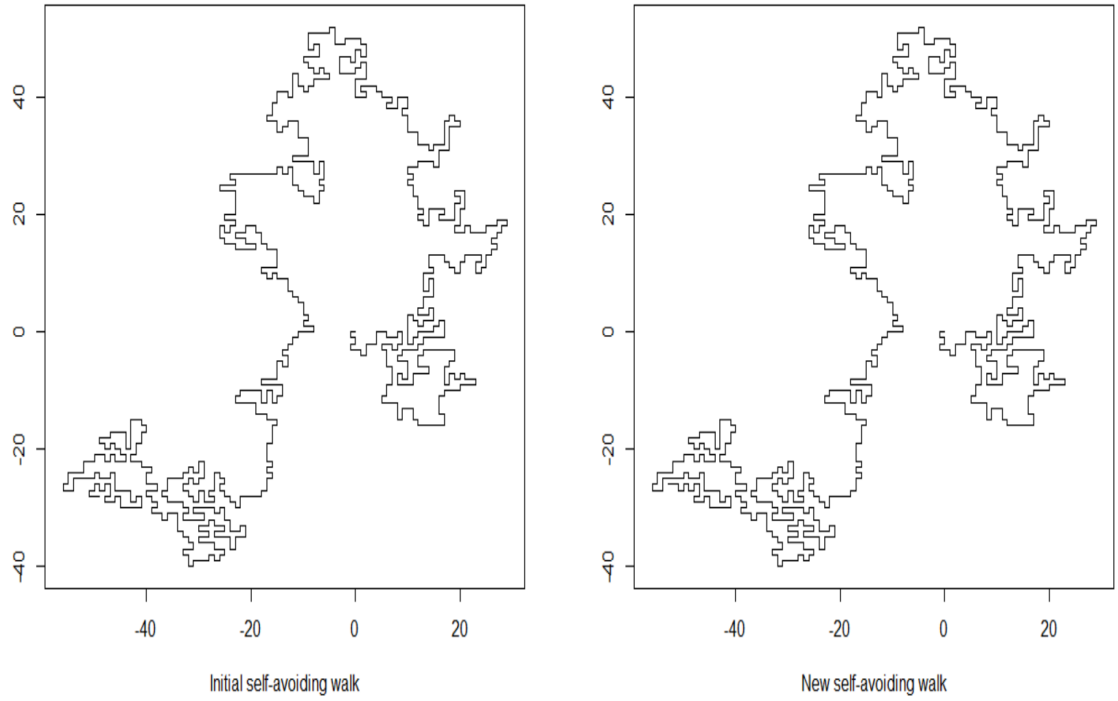

Figure 10: Self-avoiding walks of length 1000

References

[1] Nathan Clisby. Efficient implementation of the pivot algorithm for self-avoiding walks. *Journal of Statistical Physics*, 140:349–392, 2010.

[2] Paul J Flory. *Principles of polymer chemistry*. Cornell university press, 1953.

[3] Eugene Guth. Theory of filler reinforcement. *Rubber Chemistry and Technology*, 18(3):596–604, 1945.

[4] John M Hammersley and K William Morton. Poor man's monte carlo. *Journal of the Royal Statistical Society: Series B (Methodological)*, 16(1):23–38, 1954.

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

[6] Jesper Lykke Jacobsen, Christian R Scullard, and Anthony J Guttmann. On the growth constant for square-lattice self-avoiding walks. *Journal of Physics A: Mathematical and Theoretical*, 49(49):494004, 2016.

[7] Tom Kennedy. A faster implementation of the pivot algorithm for self-avoiding walks. *Journal of Statistical Physics*, 106:407–429, 2002.

[8] Werner Kuhn. Beziehungen zwischen molekülgröße, statistischer molekülgestalt und elastischen eigenschaften hochpolymerer stoffe. *Kolloid-Zeitschrift*, 76(3):258–271, 1936.

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

[10] Neal Madras and Alan D Sokal. The pivot algorithm: a highly efficient monte carlo method for the self-avoiding walk. *Journal of Statistical Physics*, 50:109–186, 1988.

[11] Berenike Maier and Joachim O Rädler. Conformation and self-diffusion of single dna molecules confined to two dimensions. *Physical review letters*, 82(9):1911, 1999.

[12] WJC Orr. Statistical treatment of polymer solutions at infinite dilution. *Transactions of the Faraday Society*, 43:12–27, 1947.

[13] Gordon Slade. Self-avoiding walks. *The Mathematical Intelligencer*, 16(1):29–35, 1994.

[14] Gordon Slade. The self-avoiding walk: A brief brief survey. *Surveys in Stochastic Processes*, 4:181, 2011.

[9] [13] [8] [12] [2] [3] [7] [6] [14] [5] [1] [10] [11] [4]