Methods for Estimating the Connective Constant of Self-Avoiding Walks on a 2D Lattice

May 13, 2025

Group Composition:

1. Lim Yik Jane, A0240501B, e0775099@u.nus.edu
2. Ryan Toh Jun Hui, A02589995B, e0969967@u.nus.edu
3. Woodford Daniel Robert, A0312944E, e1497112@u.nus.edu

1 Introduction

The self-avoiding walk (SAW) is a classical model used to describe non-intersecting paths on a lattice. It arises in statistical physics, where it captures essential features of polymer chains, and in combinatorics, where it represents a challenging counting problem, among many other fields.

A key quantity associated with SAWs is the *connective constant* μ, which characterizes the exponential growth rate of the number of such walks of length L, denoted c_L. While exact values of c_L are known for small L, they quickly become computationally intractable as L increases. Estimating μ thus requires probabilistic and numerical methods.

This paper explores several Monte Carlo approaches to estimating μ, beginning with basic random walk sampling and progressing to importance sampling, the Rosenbluth algorithm, Sequential Monte Carlo (SMC) and lastly the Pivot Algorithm with Recursive Formulation. Each method attempts to approximate the ratio c_{L+1}/c_L, which converges to μ in the limit of large L. Alongside the estimators, we examine challenges such as particle degeneracy, trapped walks, and weight variance, and discuss the trade-offs between accuracy and computational efficiency across methods.

2 Basic Deterministic Method

In this section, we describe the implementation of a deterministic method used to compute c_L for small values of L. This method serves as a reference to validate the results obtained from the

Monte Carlo simulations.

2.1 Depth-First Search for Exact Enumeration

For small values of L, the number of self-avoiding walks c_L can be computed exactly through an exhaustive depth-first search (DFS) procedure. In this method, a walk is recursively extended one step at a time by exploring all unvisited neighboring lattice sites.

At each step, the DFS explores every possible extension that maintains the self-avoiding constraint. If a dead-end is reached (no available moves), the recursion backtracks to the previous point and attempts a different extension. This exhaustive search ensures that all valid self-avoiding walks of a given length L are counted without omission or duplication.

The DFS method is computationally feasible for small L (typically up to $L \approx 20$ on a square lattice) but becomes intractable for larger L due to the exponential growth in the number of configurations. The exact values obtained from DFS match known sequences for SAWs, notably OEIS A001411 [OEI19], and are used throughout this report to validate the performance of stochastic simulation methods.

3 Naïve Monte Carlo

In this section, Monte Carlo simulation is implemented to calculate the fraction of self-avoiding walks among standard random walks of length L. Given that the probability of a random walk being self-avoiding is $(\mu/4)^L$, where $\mu \approx 2.638$, Monte Carlo methods are used to validate this result.

3.1 Algorithm Steps

This method generates standard random walks of fixed length L by taking each step uniformly at random from the four lattice directions. A walk is marked as self-avoiding if no position is revisited. Repeating this procedure for N samples, we estimate the probability that a random walk of length L is self-avoiding.

$$\text{Fraction of SAWs} = \frac{\text{Number of self-avoiding walks}}{N}$$

The estimate for c_L is then obtained by scaling this fraction by the total number of random walks of length L, which is 4^L:

$$\hat{c}_L = \text{Fraction of SAWs} \times 4^L$$

This method becomes inefficient for large L due to the rapidly decreasing probability of generating a self-avoiding walk. For example, at $L = 20$, the success rate is typically below 0.1%.

3.2 Results for Approximating the Fraction of Self-Avoiding Random Walks between $(10 \leq L \leq 20)$

Drawing from Iwan Jensen's 2013 paper *A new transfer-matrix algorithm for exact enumerations: self-avoiding walks (SAW) on the square lattice* [CJ12], the number of self avoidant walks can be approximated by:

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

where $A \approx 1.17704242$, $\mu \approx 2.638$, and $\gamma = 43/32$. The produced sequence of n-step self-avoiding walks on a square lattice is known as A001411 in the Online Encyclopedia of Integer Sequences (link) [.

The proportion of SAW for each walk of length L can be defined by:

$$\text{Proportion of SAWs} = \frac{c_L}{4^L}$$

where c_L is the number of SAW of length L and 4^L is the total number of unrestricted 2-Dimensional lattice walks of length L. This proportion of SAW derived from Jensen's computation will be referred to as "Actual", and the estimated proportion from simple Monte Carlo Simulation will be referred to as "Predicted".

L	Actual	Predicted	Difference
10	0.0421	0.0416	.0005
11	0.0287	0.0289	.0002
12	0.0194	0.0192	.0002
13	0.0131	0.0131	$< .0001$
14	0.0088	0.0090	.0002
15	0.0059	0.0060	.0001
16	0.0040	0.0041	.0001
17	0.0027	0.0028	.0001
18	0.0018	0.0018	$< .0001$
19	0.0012	0.0013	.0001
20	0.0008	0.0008	$< .0001$

4 Rosenbluth Method

The Rosenbluth method is a classical importance sampling technique for generating self-avoiding walks (SAWs) on a lattice. Rather than sampling all random walks and checking whether they are self-avoiding, the method directly builds self-avoiding walks by choosing the next step uniformly among available unvisited neighbors at each point.

To account for the sampling bias, each generated walk is assigned a weight W, defined as the product of the number of available directions at each step:

$$W(s) = \prod_{i=1}^{L} \sigma_i(s)$$

where $\sigma_i(s)$ denotes the number of unvisited neighbors available at step i.
The probability of generating a particular walk s under this procedure is:

$$P(s) = \frac{1}{W(s)}$$

Thus, the total number of SAWs of length L can be estimated from N independent samples by:

$$\hat{c}_L = \frac{1}{N} \sum_{j=1}^{N} W_j$$

If at any point during the walk no unvisited neighbors are available ($\sigma_i = 0$), the current walk is immediately rejected and a new walk is started from the origin.

4.1 Algorithm Steps

[Tab15]

1. **Initialize:** Start a walk at the origin $(0, 0)$ with weight $W = 1$.
2. **For each step** $i = 1$ to L:
   (a) Identify the set of unvisited neighboring sites.
   (b) Let σ_i be the number of available directions.
   (c) **If** $\sigma_i = 0$ (walk is trapped):
      - Reject the walk and start a new one from the origin.
   (d) **Else**:
      - Randomly choose one of the σ_i available moves.
      - Update the weight: $W \leftarrow W \times \sigma_i$.
3. **After reaching length** L, record the weight W.
4. **Repeat** until N successful walks of length L are generated.
5. **Estimate** c_L as the average of the recorded weights.

4.2 Estimation of μ

Given the asymptotic form $c_L \sim A\mu^L L^{\gamma-1}$, where μ is the connective constant, we estimate μ by:

$$\mu_L = \hat{c}_L^{1/L}$$

Tracking μ_L over increasing L allows us to observe convergence toward the theoretical value of $\mu \approx 2.638$.

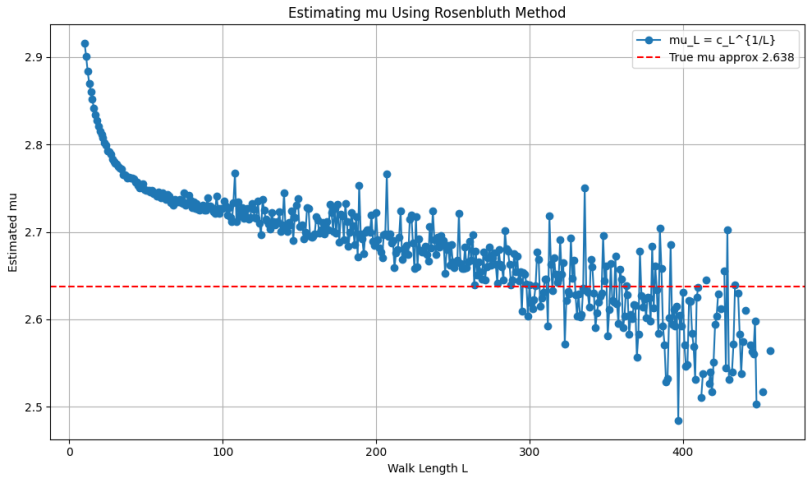

Figure 1: Running estimate of μ using the Rosenbluth method across walk lengths $10 \leq L \leq 460$

4.3 Distribution of Rosenbluth Weights

Since the weight $W(s)$ grows exponentially with the number of steps, we study its logarithm:

$$\log W(s) = \sum_{i=1}^{L} \log \sigma_i(s)$$

The distribution of $\log(W)$ becomes approximately Gaussian due to the Central Limit Theorem.

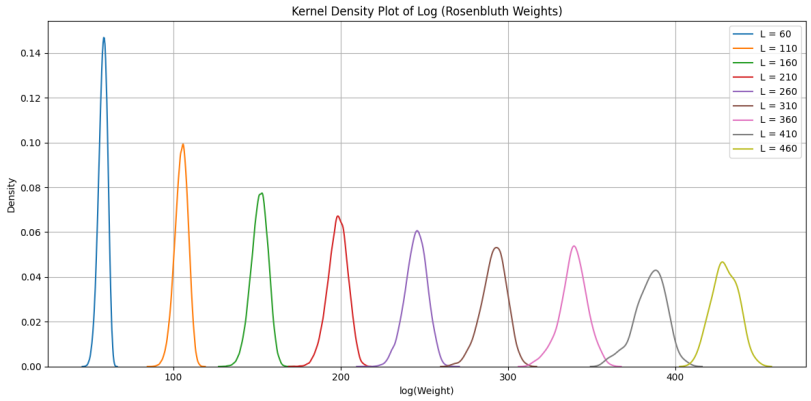

Figure 2: Kernel density plot of log-scaled Rosenbluth weights for different L

The distributions of $\log(W)$ shift linearly with increasing L. The widening of the distributions at larger L reflects increasing variance, leading to greater uncertainty in μ_L estimates.

5 Sequential Monte Carlo (SMC)

5.1 Algorithm: Sequential Monte Carlo for SAW Estimation

Sequential Monte Carlo (SMC) methods, also known as particle filters, are widely used for approximating sequences of distributions that evolve over time. In the context of self-avoiding walks, SMC provides a natural extension of importance sampling by maintaining a population of walk "particles" and resampling at each step based on their likelihood of successful continuation [DFGnd].

At each walk length L, particles attempt to extend by one valid self-avoiding step. Particles with more available extensions are more likely to be selected in the resampling step, promoting exploration of paths that are less likely to become trapped. The method adaptively reallocates computational effort to more promising regions of the configuration space, reducing variance compared to purely naive random walk simulations.

Unlike the Rosenbluth method, which treats each walk independently, SMC incorporates interaction among particles through the resampling mechanism, helping to mitigate the problem of degeneracy. This makes SMC particularly effective for estimating the connective constant μ over a wide range of walk lengths.

1. **Initialization:** Begin with N particles at the origin $(0, 0)$, each representing a possible SAW.

2. **Propagation:** For each step L, extend walks by randomly choosing one unvisited neighbor (if available).

3. **Weighting:** Let w be the number of valid extensions. Estimate:

$$\frac{c_{L+1}}{c_L} \approx \text{mean}(w)$$

This reflects the average number of valid moves across particles.

4. **Importance Sampling:** Particles with higher w are more likely to be duplicated; stuck walks ($w = 0$) are less likely to be kept. This focuses sampling on viable paths while maintaining diversity.

5. **Pivot Mutation:** After resampling, apply a random pivot move (e.g., rotation or reflection) to a portion of each walk to reduce degeneracy and enhance diversity. Many other methods can be implemented, however pivoting alone was used for efficiency.

6. **Diversity Tracking:** Monitor duplicates. In our runs, duplication hovered from 10-20%. High degeneracy may require increasing N or more aggressive mutation.

7. **Estimation:** Estimate c_L by multiplying mean weights:

$$\hat{c}_L = \prod_{i=1}^{L} \bar{w}_i$$

Example: $\bar{w}_1 = 4 \Rightarrow \hat{c}_1 = 4$; $\bar{w}_2 = 3 \Rightarrow \hat{c}_2 = 12$.

5.2 Implementation and Results

Two implementations were used. One implementation 3 used 10,000 samples, where a running average of extension ratios (c_{L+1}/c_L) was calculated for $L = (0, 1000)$. The second implementation 4 computes the extension ratios by averaging across 10 independent runs, each consisting of 1,000 samples per walk length L.

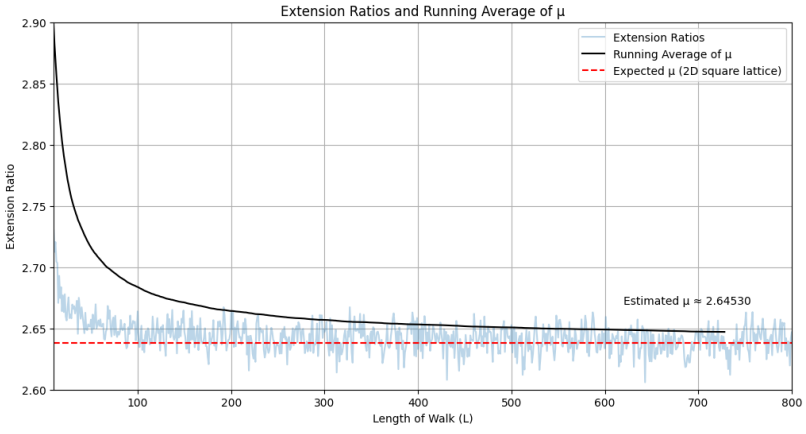

Figure 3: Running Average of μ with 10,000 Samples

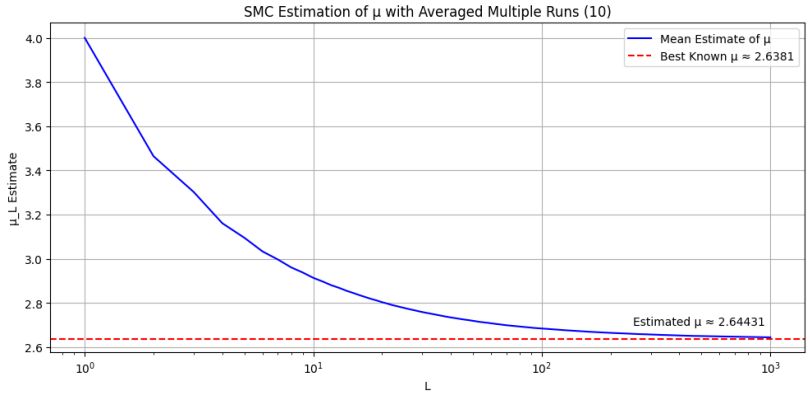

Figure 4: Running estimate of μ using SMC across 10 runs ($N = 1000$ particles each)

The final estimates (2.645 and 2.644) were slightly higher than the best known value of 2.6381. Originally we thought this may be due to the influence of higher initial estimates at small L. While averaging was considered starting from a later point (e.g., $L = 10$), this did not improve accuracy, and furthermore would have been a speculative delineation of SMC.

6 Pivot Algorithm and Recursive Formulation

This method differs from the traditional pivot algorithm by accepting only walks that, after a set number of transformations, reach equilibrium and become approximately uniformly distributed. The recursive formulation estimates $c_{2L} \approx B(L, L) \cdot c_L^2$, where $B(L, L)$ is the probability that two independent equilibrium SAWs of length L can be joined to form a valid walk of length $2L$, enabling efficient estimation of $\mu_L = c_L^{1/L}$ for larger walk lengths without direct enumeration. To prevent numerical overflow from the exponential growth of c_L, all calculations are performed in logarithmic space using the recurrence: $\log(c_{2L}) = \log(B(L, L)) + 2\log(c_L)$ This ensures numerical stability and accurate estimation of μ_L for large L.

6.1 Algorithm

1. **Initialization**
   (a) Set $L_0 = 10$ and $\log(c_{L_0}) = \log(44100)$ for recursion.
   (b) Initialize a list for μ_L estimates.
   (c) Set $L_{\max}$.
2. **Generate SAW pool using pivot algorithm**
   (a) Start with a straight line walk of length L.
   (b) Perform pivot steps: apply random symmetry transformations, accepting if the walk remains self-avoiding.

(c) After sufficient steps, consider the walk equilibrated and store it in the pool.

(d) This equilibration ensures the samples represent the correct stationary distribution, with decorrelated walks improving the accuracy of the estimates, as discussed by Sokal [Sok97].

3. **Estimate $B(L, L)$ via random sampling**

   (a) Randomly select two independent walks from the pool, align and concatenate them.

   (b) Check if the combined walk is self-avoiding.

   (c) Let B be the fraction of successful joins, the estimate for $B(L, L)$.

4. **Recursively estimate $\log(c_L)$ and μ_L**

   (a) Use the recurrence:

$$\log(c_{2L}) = \log(B(L, L)) + 2\log(c_L)$$

   (b) Calculate $\mu_{2L} = \exp\left(\frac{\log(c_{2L})}{2L}\right)$.

   (c) Repeat with $L \leftarrow 2L$ until L_{max} is reached.

6.2 Performance

Using a sample pool of $10,000$ self-avoiding walks for each walk length and $1,000,000$ randomly sampled pair trials per estimation step, the method achieved a final estimate of the connective constant with $L_{\text{max}} = 2560$:

$$\hat{\mu}_{2560} \approx 2.639781$$

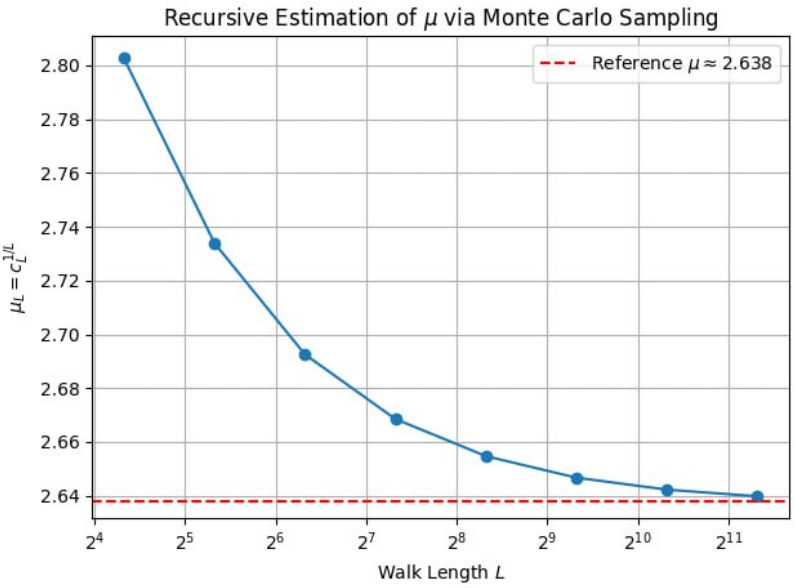

Figure 5: Recursive Estimation of μ using Pivot algorithm

7 Conclusion

In this report, we explored a series of Monte Carlo-based methods for estimating the number of self-avoiding walks (SAWs) on a 2D lattice, culminating in relatively accurate approximations of the connective constant μ. Beginning with simple random sampling, we saw its inefficiency grow quickly with L. Importance sampling through implementation of the Rosenbluth method significantly improved convergence by weighting walks based on their freedom of movement.

Across all implementations, results closely matched known exact values for small L, and estimates of μ approached the theoretical constant for larger L, validating both the effectiveness and limitations of each technique. These methods also laid the foundation for more advanced approaches like Sequential Monte Carlo and the pivot algorithm with recursive formulation, which could address particle degeneracy and further reduce variance in future work.

References

[Sok97] A. D. Sokal. "Monte Carlo methods in statistical mechanics: Foundations and new algorithms". In: *Functional Integration*. Ed. by C. DeWitt-Morette, P. Cartier, and A. Folacci. Vol. 361. Springer, 1997, pp. 131–192. DOI: 10.1007/978-1-4899-0319-8_6.

[CJ12] Nathan Clisby and Iwan Jensen. "A new transfer-matrix algorithm for exact enumerations: self-avoiding polygons on the square lattice". In: *Journal of Physics A: Mathematical and Theoretical* 45.11 (2012), p. 115202. DOI: 10.1088/1751-8113/45/11/115202.

[Tab15] M. Tabrizi. "Rosenbluth Algorithm Studies of Self-avoiding Walks". Unpublished manuscript. 2015.

[OEI19] OEIS Foundation Inc. *A001411 - OEIS*. https://oeis.org/A001411. Online Encyclopedia of Integer Sequences. 2019.

[DFGnd] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *An Introduction to Sequential Monte Carlo Methods*. https://www.stats.ox.ac.uk/~doucet/doucet_defreitas_gordon_smcbookintro.pdf. Online manuscript. n.d.