ST3247 Assignment

April 24, 2025

Group Composition:

1. Ethan Keck Jun Wei, A0180060W, e0271215@u.nus.edu

1 Introduction

Consider the two-dimensional integer lattice $\mathbb{Z}^2$. A *self-avoiding walk (SAW)* of length L is a sequence of distinct points $(z_0, z_1, \ldots, z_L)$ in $\mathbb{Z}^2$ where:

- $z_0 = (0, 0)$ (starts at origin)
- $z_i \neq z_j$ for all $i \neq j$ (no self-intersections)

Let c_L denote the number of distinct SAWs of length L. The sequence grows exponentially, and the limit:

$$\mu = \lim_{L \to \infty} c_L^{1/L}$$

defines the *connective constant*, which is known to exist but whose exact value remains unknown for $\mathbb{Z}^2$ (current best estimate: $\mu \approx 2.638158533032790$).

1.1 Goal

Develop and implement Monte Carlo (SMC) methods to:

- Compute μ from the geometric mean of these ratios
- Maintain numerical stability up to $L_{\text{max}} = 1000$
- Achieve convergence to the known μ value within $1 - 2\%$ relative error

Detailed Implementation in https://github.com/EkMi00/SAW-Monte-Carlo

2 Basic deterministic methods

2.1 Method

The implemented deterministic algorithm uses **depth-first search with backtracking** to enumerate all possible SAWs:

Algorithm 1 SAW Enumeration

Input: Target length L
Output: Count c_L of SAWs
Initialize with $walk = [(0, 0)]$, $visited = \{(0, 0)\}$
return generate_saws$(walk, L, visited)$

Algorithm 2 generate_saws (recursive helper)

if length($walk$) = $L + 1$ **then**
  return $[walk]$
end if
$saws \leftarrow [\,]$
for each neighbor $(dx, dy) \in \{(0, 1), (1, 0), (0, -1), (-1, 0)\}$ **do**
  $(x', y') \leftarrow walk[-1] + (dx, dy)$
  if $(x', y') \notin visited$ **then**
    $visited$.add((x', y'))
    $saws$.extend(generate_saws$(walk + [(x', y')], L, visited)$)
    $visited$.remove((x', y'))
  end if
end for
return $saws$

2.2 Results

The algorithm computes c_L values for $L \leq 10$:

L	c_L
1	4
2	12
3	36
4	100
5	284
6	780
7	2172
8	5916
9	16268
10	44100

These values are corroborated by [Jensen, 2025], which makes these values our point of validation for any of the estimators for SAWs of small values of L.

3 Basic Monte Carlo I

3.1 Method

The implemented Monte Carlo approach uses random sampling:

Algorithm 3 Monte Carlo SAW Estimation

Input: Walk length L, sample size N
Output: Fraction α_L, estimate $\hat{c}_L$
$count \leftarrow 0$
for $i \leftarrow 1$ to N **do**
    Generate random walk W of length L
    if W is self-avoiding **then**
        $count \leftarrow count + 1$
    end if
end for
$\alpha_L \leftarrow count/N$
$\hat{c}_L \leftarrow \alpha_L \times 4^L$
return $(\alpha_L, \hat{c}_L)$

3.2 Results

For $L = 1, \ldots, 10$ with $N = 10^5$ samples:

Table 1: Comparison of Estimated and Deterministic c_L Values

L	Estimated c_L	Deterministic c_L	Percentage Error (%)
1	4.00000	4	0.000000
2	12.00416	12	0.034667
3	35.96224	36	0.104889
4	100.87936	100	0.879360
5	287.03744	284	1.069521
6	776.72448	780	0.419938
7	2163.67104	2172	0.383470
8	5846.46656	5916	1.175346
9	16171.66336	16268	0.592185
10	43935.33440	44100	0.373391

We observe that the estimated c_L seems to adhere close estimates to the deterministic c_L.

4 Basic Monte Carlo II

4.1 Method

The implemented algorithm uses a more sophisticated Monte-Carlo method that generates a walk of length L by sampling the next step z_{i+1} uniformly from the set of possible neighbors of z_i that do not lead to a self-intersection. If there is no such neighbor, which can happen, the walk remains still at z_i until the end of the walk, i.e. $z_{i+1} = z_i$ until $i = L - 1$.

The core procedure consists of two components:

Algorithm 4 Dynamic SAW Generation with Importance Weighting

Input: Target length L
Output: SAW W, importance weight ω
Initialize $W \leftarrow [(0,0)]$, $\mathcal{V} \leftarrow \{(0,0)\}$, $\omega \leftarrow 1$
for $i \leftarrow 1$ to L **do**
  $(x, y) \leftarrow W[-1]$
  $\mathcal{N} \leftarrow \{(x \pm 1, y), (x, y \pm 1)\} \setminus \mathcal{V}$
  if $\mathcal{N} = \emptyset$ **then**
    W.append((x, y)) {Trapped - zero-length step}
  else
    $(x', y') \leftarrow$ UniformSample($\mathcal{N}$)
    W.append((x', y'))
    $\mathcal{V}$.add((x', y'))
    $\omega \leftarrow \omega \times |\mathcal{N}|$
  end if
end for
return (W, ω)

Algorithm 5 Monte Carlo Estimation of c_L

Input: L, sample size N
Output: Estimate $\hat{c}_L$
$\Omega \leftarrow 0$
for $i \leftarrow 1$ to N **do**
  $(W, \omega) \leftarrow$ generate_dynamic_saw(L)
  $\Omega \leftarrow \Omega + \omega$
end for
return Ω / N

4.2 Results

For $L = 1, \ldots, 10$ with $N = 10^5$ samples:

We observe that the estimated c_L seems to adhere close estimates to the deterministic c_L.

Table 2: Comparison of Estimated and Deterministic c_L Values

L	Estimated c_L	Deterministic c_L	Percentage Error (%)
1	4.00000	4	0.000000
2	12.00000	12	0.000000
3	36.00000	36	0.000000
4	99.94176	100	0.058240
5	283.87800	284	0.042958
6	779.52240	780	0.061231
7	2173.56696	2172	0.072144
8	5924.68020	5916	0.146724
9	16283.84796	16268	0.097418
10	44069.56632	44100	0.069011

5 Importance Sampling

5.1 Method

The implemented algorithm adapts the Rosenbluth method [Rosenbluth and Rosenbluth, 1955] to estimate the connective constant μ for self-avoiding walks (SAWs) on $\mathbb{Z}^2$. The key features are:

- Sequential growth of SAWs with importance weighting
- Dynamic weight adjustment based on available steps
- Direct estimation of c_L through weighted averaging

The core procedure consists of two phases:

Algorithm 6 Rosenbluth SAW Generation

Input: Target length L
Output: Walk W, importance weight ω
Initialize $W \leftarrow [(0,0)]$, $\omega \leftarrow 1$
for $i \leftarrow 1$ to L **do**
  $\mathcal{N} \leftarrow \{\text{unvisited neighbors of } W[-1]\}$
  if $\mathcal{N} = \emptyset$ **then**
    return $(None, 0)$ {Trapped walk}
  end if
  $v \leftarrow \text{UniformSample}(\mathcal{N})$
  $W.\text{append}(v)$
  $\omega \leftarrow \omega \times |\mathcal{N}|$
end for
return (W, ω)

Algorithm 7 Connective Constant Estimation

Input: N (number of samples), L_{max} (maximum length)
Output: μ estimates
for $L \leftarrow 1$ to L_{max} **do**
  $\Omega \leftarrow 0$, $K \leftarrow 0$
  for $i \leftarrow 1$ to N **do**
    $(W, \omega) \leftarrow$ GenerateWalk(L)
    if $W \neq None$ **then**
      $\Omega \leftarrow \Omega + \omega$
      $K \leftarrow K + 1$
    end if
  end for
  $c_L \leftarrow \Omega / K$
  $\mu_L \leftarrow c_L^{1/L}$
end for

5.2 Results

Table 3: Comparison of Estimated and Deterministic c_L Values

L	Estimated c_L	Deterministic c_L	Percentage Error (%)
1	4.000000	4	0.000000
2	12.000000	12	0.000000
3	36.000000	36	0.000000
4	99.396000	100	0.604000
5	286.308000	284	0.812676
6	787.428000	780	0.952308
7	2186.028000	2172	0.645856
8	6091.394790	5916	2.964753
9	16273.413897	16268	0.033279
10	44552.286290	44100	1.025592

We observe that the estimated c_L seems to adhere close estimates to the deterministic c_L, with the exception of $L = 8$ and $L = 10$ having a percentage error of $> 1\%$. This suggests the algorithm is able to get a good general estimate of c_L.

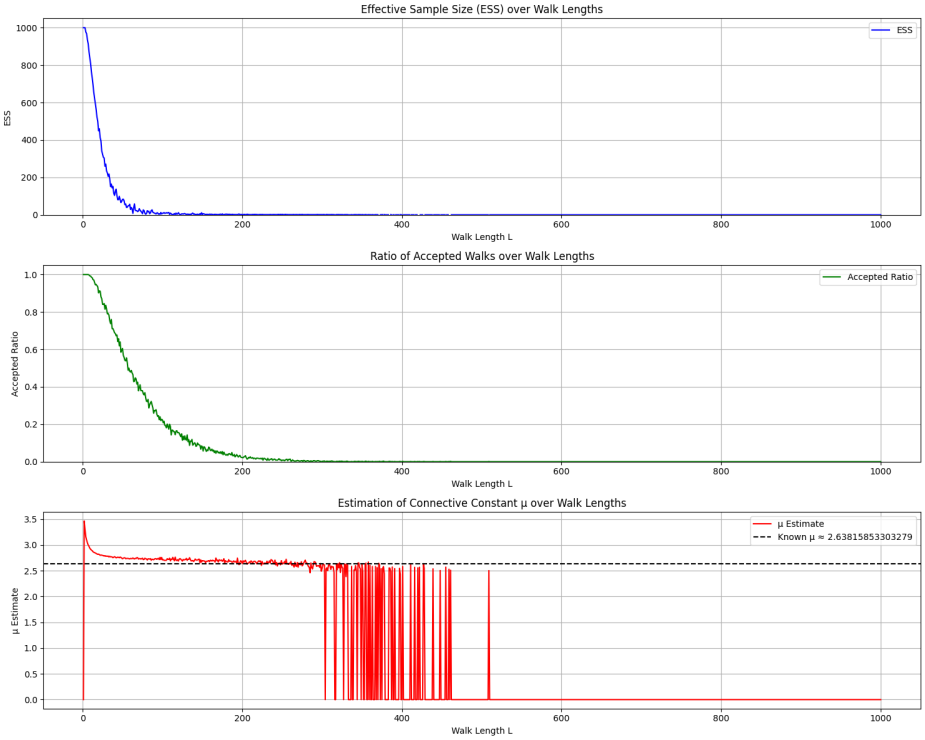

Figure 1: Evolution of Importance Sampling μ Estimate

We observe that the algorithm slowly converges to the known best estimate $\mu = 2.638158533032790$ from $L = 1, \ldots, \approx 300$. However, it rapidly deteriorates from $L = 300$ to $L = 1000$. This is expected as the Effective Sample Size and ratio of accepted walks gradually becomes smaller. This implies the importance weights are highly unequal as well. We may consider better algorithms.

6 Sequential Monte Carlo

6.1 Method

The implemented SMC algorithm is as follows:

Algorithm 8 SMC for SAWs

Initialize N walkers at origin
for $L = 1$ to L_{max} **do**
  for each walker **do**
    Propose extension to unvisited neighbors (k choices)
    Assign weight $w = k$
  end for
  Compute average weight $\bar{w}_L$
  Update $\log c_L = \sum_{i=1}^{L} \log \bar{w}_i$
  Estimate $\mu_L = \exp(\log c_L / L)$
  Resample walkers proportionally to weights
  Record ESS and acceptance ratios
end for

6.2 Results

For $L = 1, \ldots, 10$ with $N = 10^5$ samples:

Table 4: Comparison of Estimated and Deterministic c_L Values

L	Estimated c_L	Deterministic c_L	Percentage Error (%)
1	4.000000	4	0.000000
2	12.000000	12	0.000000
3	36.000000	36	0.000000
4	98.640000	100	1.360000
5	282.406320	284	0.561155
6	769.274816	780	1.375024
7	2143.968911	2172	1.290566
8	5820.875594	5916	1.607918
9	16234.422032	16268	0.206405
10	44238.800038	44100	0.314739

We observe that the estimated c_L seems to adhere close estimates to the deterministic c_L, with the exception of $L = 4, 6, 7, 8$ having a percentage error of $> 1\%$. This suggests the algorithm is able to get a good general estimate of c_L with some error.

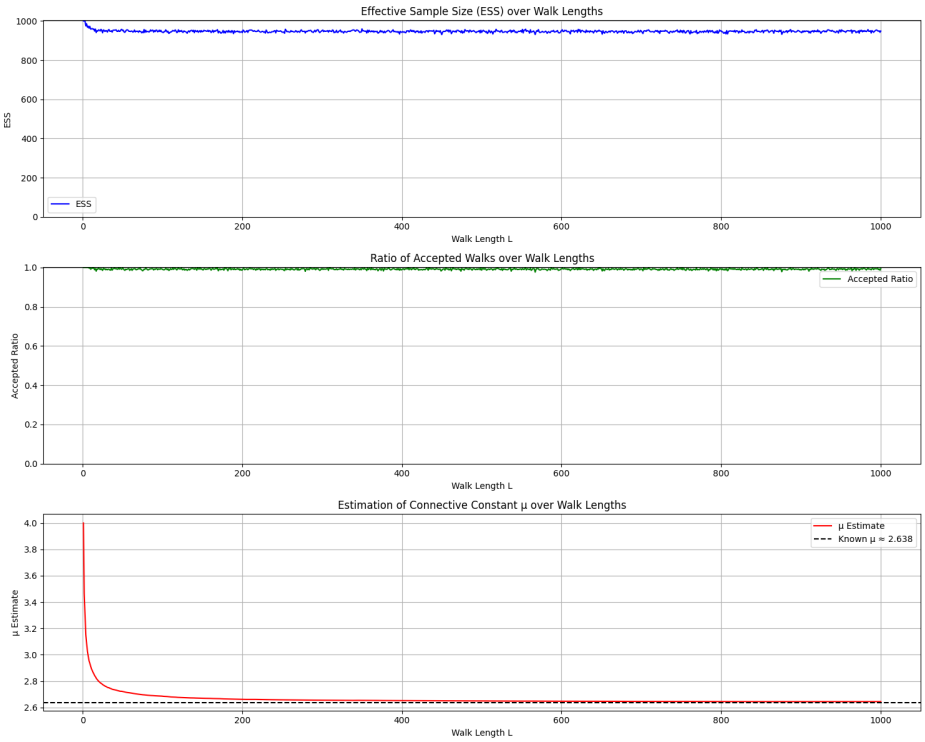

Figure 2: Convergence of μ estimates with walk length

We observe that the algorithm is able to converge on known best estimate $\mu = 2.638158533032790$ from $L = 1, \ldots, 1000$. This is supported by the high Effective Sample Size and Ratio of accepted walks, which demonstrates the algorithms stability and consistency. The algorithm's final μ estimate at $L = 1000$ is $\mu = 2.64463$.

7 Conclusion

Overall each method has demonstrated consistency in providing a good estimate for μ at smaller values of L. The exception being Sequential Monte Carlo implmentation that is able to converge on the known $\mu = 2.638158533032790$. It is the most stable and shows the most promise as it maintains a high ESS and ratio of accepted walk. Future works can explore improving the simulations to run at larger values of L and sample sizes to further results to a higher level of accuracy.

References

[Jensen, 2025] Jensen, I. (2025). A new transfer-matrix algorithm for exact enumerations: self-avoiding walks on the square lattice.

[Rosenbluth and Rosenbluth, 1955] Rosenbluth, M. N. and Rosenbluth, A. W. (1955). Monte carlo calculation of the average extension of molecular chains. *The Journal of Chemical Physics*, 23:356–359.