Monte Carlo simulation of Polymers

Mankrit Singh, Noa Aarts, Raghav Juyal

May 2025

Abstract

In this report, we simulate polymer chains using the approximate importance sampling Monte Carlo method, for self-avoiding random walks on both 2D and 3D lattices. We investigate key polymer observables—such as end-to-end distance and radius of gyration—generated using the Rosenbluth method and its improved variant, the Pruned-Enriched Rosenbluth Method (PERM), and compare them to theoretical predictions. In two dimensions, we further extend the simulation to non-square lattices, including triangular and hexagonal grids. We present performance optimizations, approximate analytical error estimations, and analyze the influence of lattice geometry on polymer behavior. Our results indicate that PERM achieves closer agreement with theoretical expectations than the original Rosenbluth method and that lattice geometry has a negligible effect on the studied observables at scale.

1 Introduction

The statistical behavior of polymer chains play a central role in fields ranging from materials science to biophysics. These systems are often modeled as self-avoiding random walks (SAWs), which capture the interactions inherent in real polymers by preventing monomers from occupying the same spatial position. However, directly generating all valid polymer configurations becomes exponentially infeasible with increasing chain length, especially in high dimensions.

Monte Carlo methods, including approximate importance sampling, offer a practical alternative, allowing us to produce representative ensembles of polymer chains without generating them exhaustively. In this work, we employ approximate importance sampling to simulate self-avoiding walks, focusing on two widely used algorithms: the Rosenbluth method and the Pruned-Enriched Rosenbluth Method (PERM). While the Rosenbluth algorithm incrementally builds polymers by randomly selecting among valid steps, PERM enhances this process by adaptively pruning low-weight chains and enriching high-weight ones, improving sampling efficiency and statistical stability for longer chains.

We apply these techniques on both 2D and 3D lattices, including non-square 2D grids such as triangular and hexagonal geometries, to investigate their effect on polymer observables. Our simulations focus on physically meaningful quantities such as the end-to-end distance and radius of gyration, which provide insights into the scaling behavior and spatial extent of polymer configurations.

Additionally, we analyze the performance characteristics of our simulator, discuss key optimization strategies, and quantify the accuracy of our results using approximate analytical error estimates. Finally, we evaluate whether the underlying lattice geometry significantly influences macroscopic observables and find that—despite local differences in connectivity—the large-scale behavior remains largely invariant across grid types.

2 Methods

In this chapter we will explain background knowledge and the functioning of the simulator. To achieve this we start by elaborating on the problem in section 2.1 and sampling in section 2.2. We then describe the observables in section 2.3, which is followed by the Rosenbluth method in section 2.4, after which we will talk about the changes we made to implement PERM for this protocol in section 2.5. We continue with a section on the error calculation in section 2.6. Then in section 2.7 we discuss the performance of our simulation and what was important to improve it. Finally we will touch upon using a grid consisting of triangles or hexagons in section 2.8.

2.1 Assumptions

In order to simulate long polymers we had to make some assumptions. Firstly, we assumed that the polymers did not interact with each other, we did this to allow us to simulate a single polymer at a time. Then, we modeled the polymer to be a chain of same-length subunits where the subunits have their bonds fixed at certain angles, this forces the polymer to live on a grid. Lastly we assumed that a single lattice site of the grid can only contain a single monomer. These constraints reduce the polymer problem to a problem of the self-avoiding random walk.

2.2 Sampling

Based on these assumptions, it is theoretically possible to generate all chains of length L by generating all random walks of length L and then removing every chain which violates one of the constraints. But in the case of a square grid this means that 4^L polymers would need to be checked, since during a random walk we can go 4 ways at every step, which is computationally way too expensive for large values of L. A different approach is to instead sample from the set of valid polymers, we decided to use approximate importance sampling for this. In approximate importance sampling[1, Monte Carlo method] we sample from a distribution that is as close as possible to the real distribution while being more practical. This however may prove problematic due to Eq. 1.

$$\int p_{real}(R)A(R)dR \neq \int p_{approx}(R)A(R)dR \tag{1}$$

To fix this problem we need to rescale the integral over p_{approx} such that we find Eq. 2.

$$\int p_{approx}(R)\frac{p_{real}(R)}{p_{approx}(R)}A(R)dR \tag{2}$$

Where we can call $\frac{p_{real}(R)}{p_{approx}(R)} = w(R)$ the weight. Now we can use the central limit theorem to find Eq. 3 where R_i are sampled from p_{approx}.

$$\int p_{real}(R)A(R)dR \approx \frac{1}{N}\sum_{i=1}^{N} w(R_i)A(R_i) \tag{3}$$

This sampling method does not work in general due to the dimensionality problems that also plague other methods, but in this case we can generate a representative set of polymers using the Rosenbluth method explained in section 2.4 so these issues hit less hard.

2.3 Observables

To characterize the geometry of polymer chains, we focus on two key observables: the **end-to-end distance** and the **radius of gyration**.

The squared end-to-end distance is defined as

$$r_e^2(L) = \left|\vec{r}_L - \vec{r}_0\right|^2, \tag{4}$$

where $\vec{r}_0$ is the position of the first monomer and $\vec{r}_L$ is the position of the last monomer in a chain of length L.

An alternative observable is the squared radius of gyration, which describes how spread out or "curled up" the polymer is around its center of mass:

$$r_g^2(L) = \frac{1}{L+1}\sum_{i=0}^{L} \left|\vec{r}_i - \vec{r}_{\text{cm}}\right|^2, \tag{5}$$

where the center of mass is given by

$$\vec{r}_{\text{cm}} = \frac{1}{L+1}\sum_{i=0}^{L}\vec{r}_i. \tag{6}$$

For self-avoiding random walks (SAWs), both of these observables obey a scaling law with respect to chain length:

$$\langle r_e^2(L)\rangle \sim L^{2\nu}, \tag{7}$$

$$\langle r_g^2(L)\rangle \sim L^{2\nu}, \tag{8}$$

where ν is the critical exponent that depends on the dimensionality of the space. Specifically, for self-avoiding walks from [1, Monte Carlo simulations of polymers] we see:

- $\nu = \frac{3}{4}$ in 2D,
- $\nu \approx \frac{3}{5}$ in 3D.

2.4 The Rosenbluth Method

A possible way to sample the polymers of length L is to use the Rosenbluth method[2]. In this method one starts from a lattice site, and starts growing a polymer step-by-step, randomly picking one of the valid ways to put the next monomer. Once no more valid next site exists the chain cannot grow anymore and if the target length was not reached, start again from the beginning. It's important to have the weights of the chains as well since these allow us to approximate the fraction $\frac{p_{real}}{p_{approx}}$ which is essential for the sampling to work, so we record the weights $w^{(L)} = \prod_{i=1}^{L} c_i$ for each chain where c_i are the amount of choices there were at step i for that polymer. After creating enough chains of the length L we can use these to create the weighted average from Eq. 9 where $r(L)$ is the desired observable for a polymer and k indicates what polymer gets acted on.

$$\langle r^2(L)\rangle = \frac{\sum_{k=1}^{N} w_k^{(L)} r_k^2(L)}{\sum_{k=1}^{N} w_k^{(L)}} \tag{9}$$

2.5 PERM

For long chains L the Rosenbluth method from the previous section runs into problems. Where it will find polymers with very different weights, because we are taking a weighted average this means that a few polymers will heavily dominate which effectively reduces the number of significant polymers. Grassberger[3] introduced an extra step to take for each length which would help reduce this issue, Pruning and Enriching, to create the Pruned-enriched Rosenbluth method (PERM). In this method after each growth step there are two more steps. For each polymer k we check if the weight w_k is smaller than some threshold W_-, this is the pruning step in PERM. If this is the case we choose one of two options with equal probability. The first option is to discard the polymer k from the current step onward. The second option is to keep it but double the weight of the polymer from here onward. Now that we've finished pruning it's time for Enrichment, which happens if $w_k > W_+$, where W_+ is again some threshold. In this case we half it's weight w_k after which we add a second copy of the polymer. Both pruning and enriching keep the total weights the same on average, during pruning we're removing half but doubling the weight of the other half keeping the total the same, while enriching doubles it's polymers but halves their weight also not changing the total. In our implementation we added w_- and w_+ to the configuration, where we used them to such that $W_- = W_{mean}w_-$ and $W_+ = W_{mean}w_+$.

2.6 Error Estimation

To estimate the errors, we used the approximate analytic formula provided in [1, Approximate analytical formulas], which is given as

$$s(\langle r^2(L)\rangle) = \sqrt{\frac{N}{N-1}\frac{\sum_{k=1}^{N}(w_k^{(L)})^2(r_k^2(L)-(\langle r^2(L)\rangle))^2}{(\sum_{k=1}^{N}w_k^{(L)})^2}} \tag{10}$$

This is used to compute the errors for both Rosenbluth and PERM methods. Although PERM does introduce correlations, as mentioned in [1, Simulating polymers as self-avoiding random walks], if the pruning and enriching are done appropriately, most polymers will remain uncorrelated, justifying the use of the same formula. We also take into account these errors while fitting the curve by passing them as the `sigma` parameter into `scipy.optimize.curve_fit()`.

2.7 Performance

We chose to write the simulator in Python, this is an interpreted language, it's not very fast on it's own, we therefore used the numpy library which adds array operations running in a C++ runtime. Due to the significantly faster operations inside of numpy we decided to use it's functions for as many parts of the program as possible. These choices already made our simulation fast enough for our use case, although when we decided to profile the code we found that most of the time spent was in two functions. Calculating the gyration observable was one of these functions, to calculate the gyration for every step up to length L the function we were using was $O(L^2)$, so when we found out that it was calculating the gyration even when a polymer already stopped we were able to greatly reduce it's part of the total runtime. The other function that was performing poorly was the function implementing the PERM specific steps. This contained a function that concatenated a list of polymers to duplicate to the numpy array with all the chains, due to how numpy was storing the array and how `numpy.concatenate()` was used this meant that a lot of data would have to be copied every timestep of the simulation. Modifying the code so space would be allocated in advance removed this bottleneck and improved the performance of the PERM step by a factor of ≈ 5 as can be seen in Appendix A.2 .

2.8 Non-Square Lattice

In addition to the standard square lattice, we implemented self-avoiding random walks on two alternative 2D geometries: the **triangular** and **hexagonal** lattices. These were realized by defining appropriate sets of nearest-neighbor directions.

2.8.1 Triangular Lattice

The triangular lattice allows for six symmetric directions from each site as can be seen in the Fig 1. We define these directions as:

- A $(0,1)$
- B $(0,-1)$
- C $\left(\frac{\sqrt{3}}{2},\frac{1}{2}\right)$
- D $\left(\frac{\sqrt{3}}{2},-\frac{1}{2}\right)$
- E $\left(-\frac{\sqrt{3}}{2},-\frac{1}{2}\right)$
- F $\left(-\frac{\sqrt{3}}{2},\frac{1}{2}\right)$

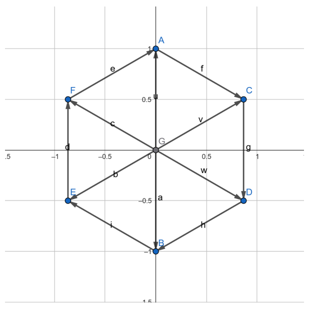


Figure 1: Triangular Lattice

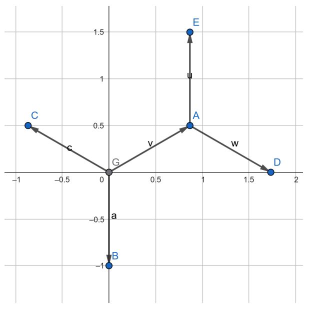


Figure 2: Hexagonal Lattice

At each step of the walk, we check all six possible directions and discard any move that would lead to a position already occupied in the current chain. To allow for floating point precision, a small numerical tolerance (10^{-3}) is used to compare positions.

2.8.2 Hexagonal Lattice

The hexagonal lattice is implemented as a staggered version of a triangular tiling, where each point has three nearest neighbors. The available directions alternate depending on the parity of the step. If we start with a lower triangle, we can move to sites A, B and C indicated in Fig 2. Assuming we move to A, we then have to take the upper triangle and move to either E, D or back to G (not allowed because of self-avoiding constraint). Then, we again take the next lower triangle and so on.

This ensures that the walker remains constrained to the hexagonal geometry. Self-avoidance is enforced in the same manner as for the triangular case, by excluding directions that revisit already occupied positions.

3 Results

In this chapter we show the results we obtained. We first show our results on 2D square lattice in section 3.1, followed by our results on 3D cubic lattice in section 3.2, and finally our results on non-square lattices in section 3.3. The parameters used for the images in this section can be found in Appendix A.1

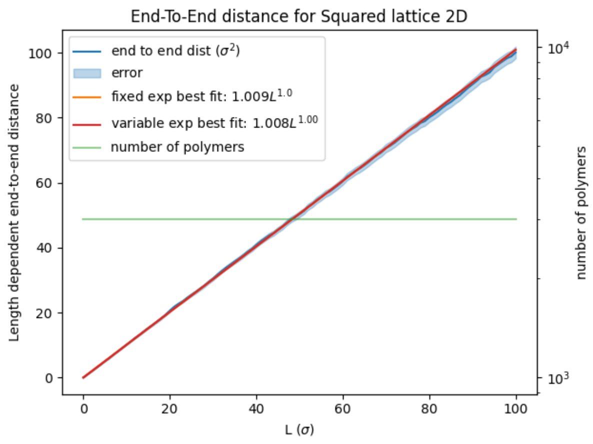
3.1 2D Square Lattice

For the 2D square lattice, we first simulated the free random walk. As seen in Fig 3, we get pretty close to the expected value of $\nu = 1/2$ or $2\nu = 1$ [1, Monte Carlo simulations of polymers]. We get an unweighted R^2 score of 1 and 0.998 for end-to-end distance and radius of gyration respectively.

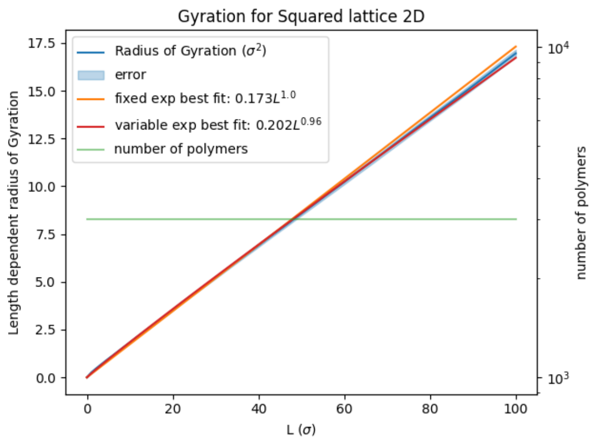
We then simulated self-avoiding random walks using the Rosenbluth and PERM methods. As seen in Figs 4 and 5, we get pretty close to the expected value of $\nu = 3/4$ or $2\nu = 3/2$ [1, Monte Carlo simulations of polymers]. We can see the comparisons of the R^2 scores in Table 1.

Method	R^2 score(end-to-end Distance)	R^2 Score(Radius of Gyration)
Rosenbluth	0.600	0.900
PERM	0.988	0.994

Table 1: Unweighted R^2 scores for curve fit on theoretical value for 2D

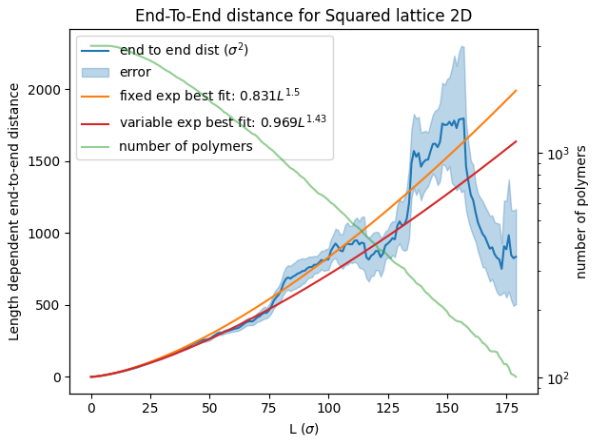


(a) end-to-end Distance

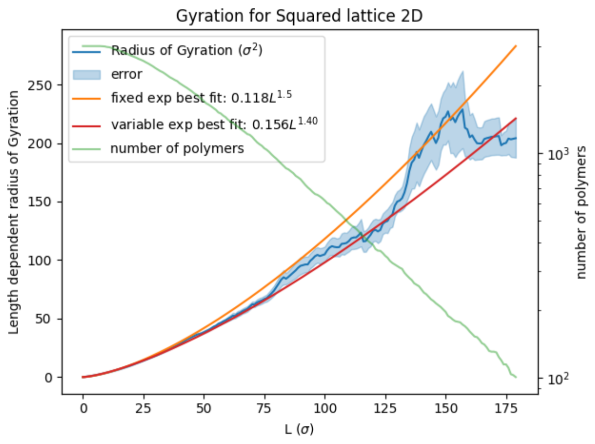


(b) Radius of Gyration

Figure 3: 3a is the end-to-end distance plot and 3b is the radius of gyration plot for 2D free random walk



(a) end-to-end Distance



(b) Radius of Gyration

Figure 4: 4a is the end-to-end distance plot and 4b is the radius of gyration plot for 2D Rosenbluth Method

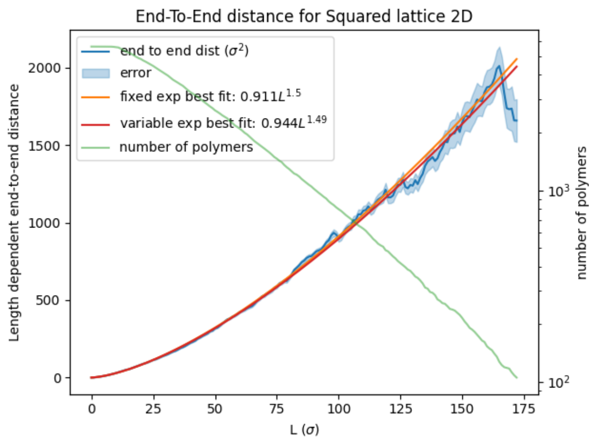
3.2 3D Cubic Lattice

For the 3D cubic lattice, we first simulated the free random walk. As seen in Fig 6, we didn't have a theoretical value, but we get pretty close to the expected value of the 2D free random walk $\nu = 1/2$ or $2\nu = 1$ [1, Monte Carlo simulations of polymers]. We get an unweighted R^2 score of 1 and 0.999 for end-to-end distance and radius of gyration respectively.

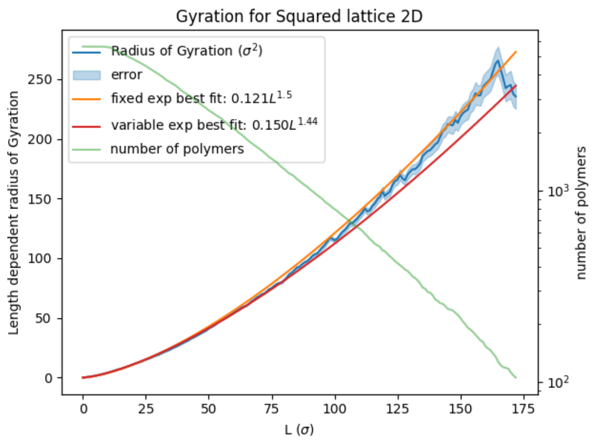
We then simulated self-avoiding random walks using the Rosenbluth and PERM methods. As seen in Figs 7 and 8, we get pretty close to the expected value of $\nu = 3/5$ or $2\nu = 6/5$ [1, Simulating polymers as self-avoiding random walks]. We can see the comparisons of the R^2 scores in Table 2.

3.3 Non-square lattice

As mentioned in 2.8, we simulated triangular and hexagonal lattices. As can be seen in Figs. 9 and 10, we didn't have a theoretical value, but we get kind of close to the expected value of 2D square lattice which has $\nu = 3/4$ or $2\nu = 3/2$ [1, Monte Carlo simulations of polymers]. For the triangular lattice, we

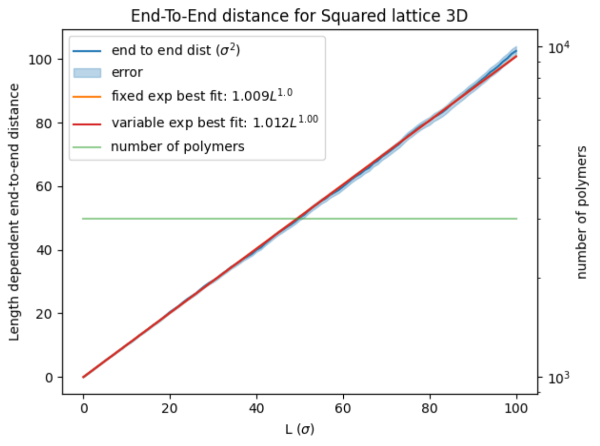


(a) end-to-end Distance

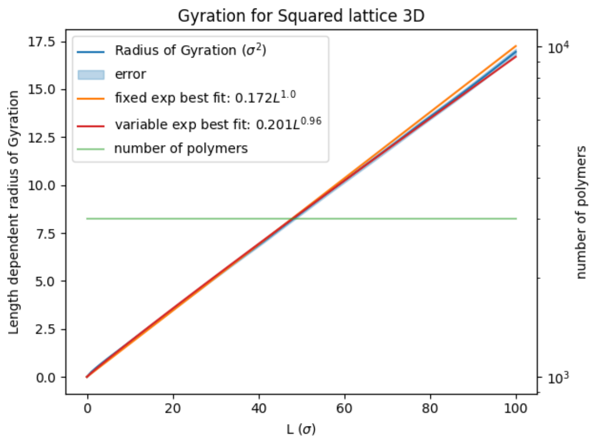


(b) Radius of Gyration

Figure 5: 5a is the end-to-end distance plot and 5b is the radius of gyration plot for 2D PERM Method



(a) end-to-end Distance



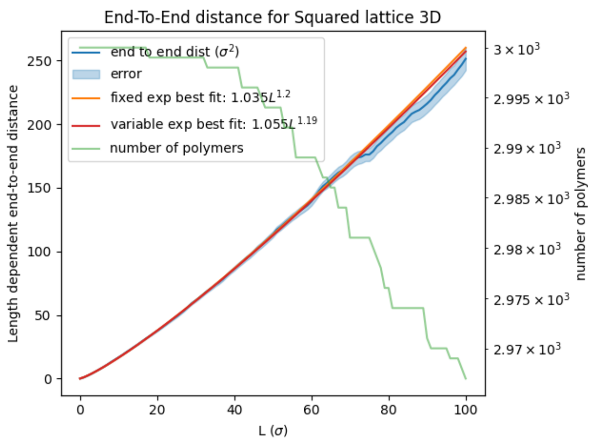
(b) Radius of Gyration

Figure 6: 6a is the end-to-end distance plot and 6b is the radius of gyration plot for 3D free random walk

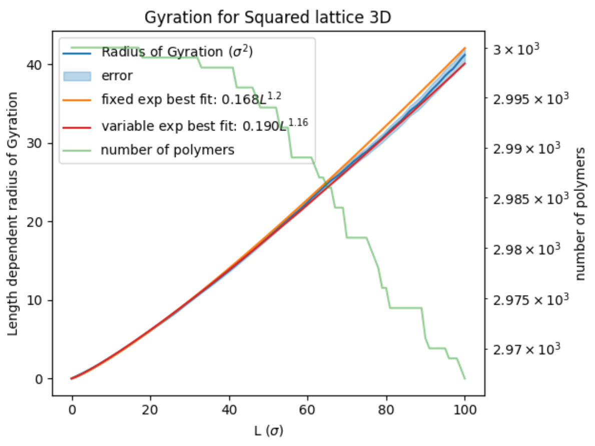
Method	R^2 score(end-to-end Distance)	R^2 Score(Radius of Gyration)
Rosenbluth	0.994	0.998
PERM	0.999	0.996

Table 2: Unweighted R^2 scores for curve fit on theoretical value for 3D

get R^2 score values of 0.715 and 0.935 for end-to-end distance and radius of gyration respectively. For the hexagonal lattice, we get R^2 score values of 0.861 and 0.960 for end-to-end distance and radius of gyration respectively.

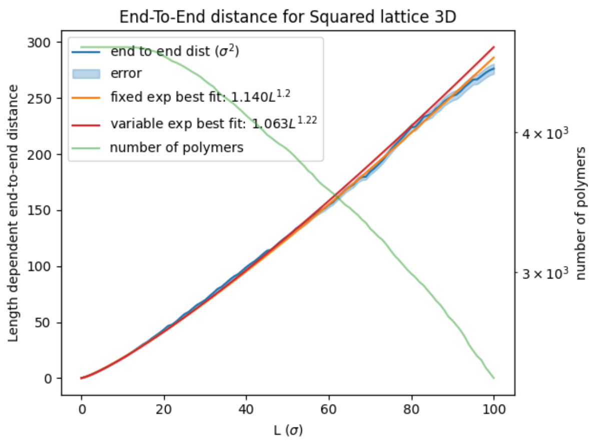


(a) end-to-end Distance

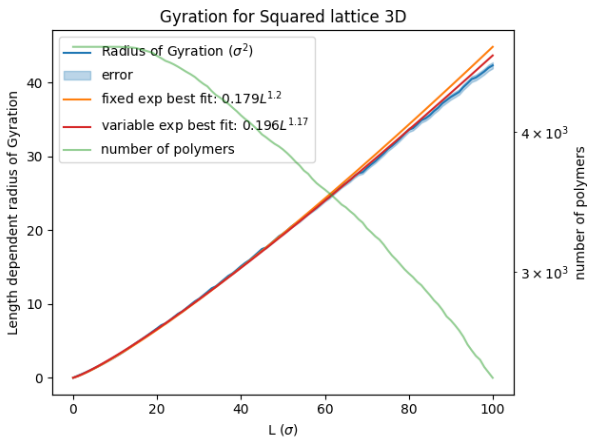


(b) Radius of Gyration

Figure 7: 7a is the end-to-end distance plot and 7b is the radius of gyration plot for 3D Rosenbluth Method

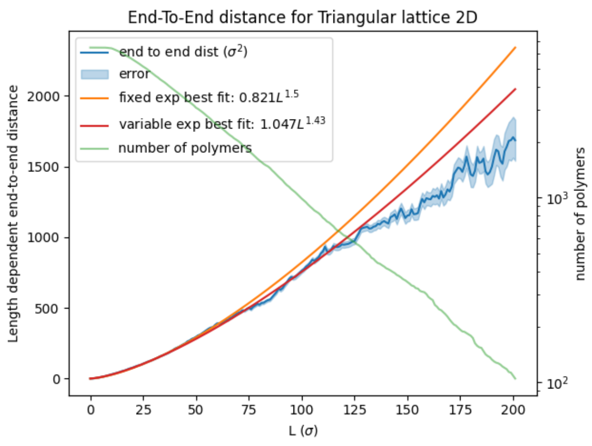


(a) end-to-end Distance

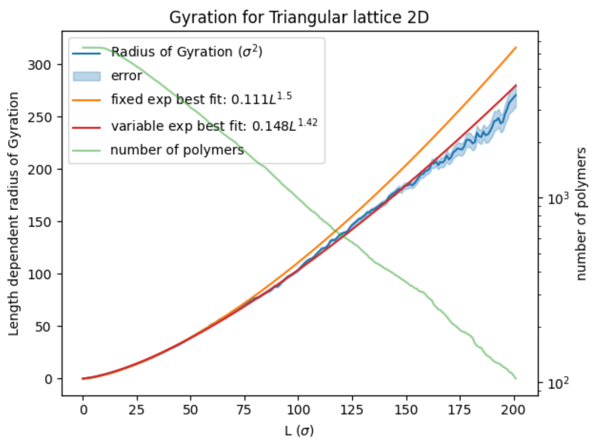


(b) Radius of Gyration

Figure 8: 8a is the end-to-end distance plot and 8b is the radius of gyration plot for 3D PERM Method

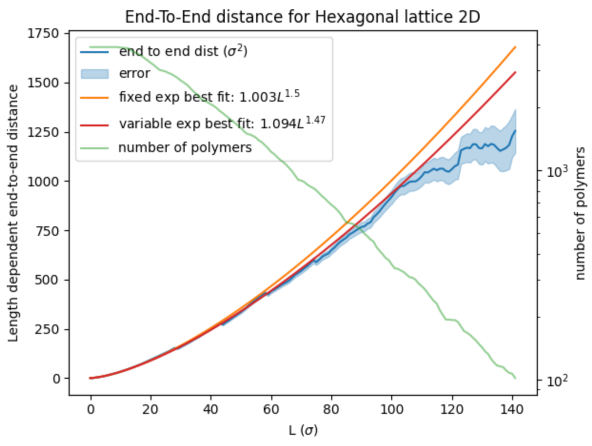


(a) end-to-end Distance

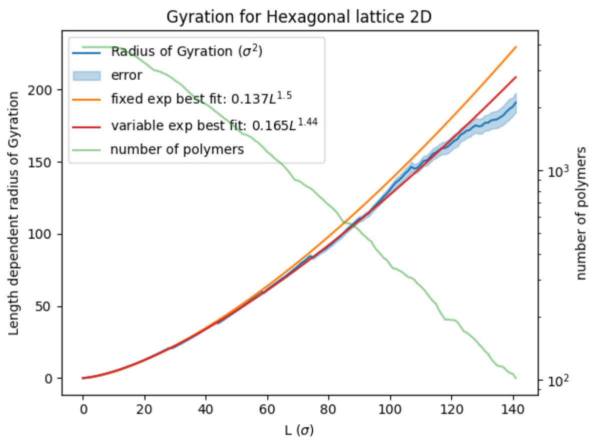


(b) Radius of Gyration

Figure 9: 9a is the end-to-end distance plot and 9b is the radius of gyration plot for triangle lattice using PERM Method



(a) end-to-end Distance



(b) Radius of Gyration

Figure 10: 10a is the end-to-end distance plot and 10b is the radius of gyration plot for hexagon lattice using PERM Method

4 Conclusion

In this project, we developed a flexible and efficient simulator for self-avoiding polymer chains using both the Rosenbluth and PERM sampling algorithms. Our approach is based on the approximate importance sampling Monte Carlo method to estimate key polymer observables, such as the end-to-end distance and radius of gyration.

The PERM algorithm used pruning and enrichment techniques to overcome the weight dispersion problem in long chains, which significantly enhanced the robustness of our results. Analytical error estimates were used to ensure the reliability of the observable averages, especially when fitting scaling exponents.

The simulator was optimized using NumPy for computational speed and memory efficiency, allowing us to probe polymers of significant lengths. Furthermore, by generalizing our framework to non-square lattices like triangular and hexagonal geometries in two dimensions, we attempted to study how lattice topology influences polymer behavior. We found that the local topology of the lattice has a negligible effect on the overall observable scaling, which aligned well with the theoretical predictions for 2D.

Altogether, our work provides a solid computational foundation for exploring polymer statistics on various lattice geometries, with potential applications in soft matter physics, biophysics, and materials science.

References

[1] M. Wimmer and A. Akhmerov, *Computational physics*, Lecture notes for Computational Physics at TUDelft, 2020-2025.

[2] M. N. Rosenbluth and A. W. Rosenbluth, “Monte carlo calculation of the average extension of molecular chains,” *The Journal of chemical physics*, vol. 23, no. 2, pp. 356–359, 1955.

[3] P. Grassberger, “Pruned-enriched rosenbluth method: Simulations of θ polymers of chain length up to 1 000 000,” *Phys. Rev. E*, vol. 56, pp. 3682–3693, 3 Sep. 1997. DOI: `10.1103/PhysRevE.56.3682`. [Online]. Available: `https://link.aps.org/doi/10.1103/PhysRevE.56.3682`.

A Appendix

A.1 Configurations

In this section, we've included all the configurations necessary to create the various figures in this document.

A.1.1 2D except for free random walk

For all 2D plots(except for free random walk), we used the following config with some changes depending on the simulation. For PERM, do_perm is true and for Rosenbluth, do_perm is false. For the square lattice, next_sides_function is simulate.get_allowed_sides_2d, for the triangular lattice it is simulate.get_allowed_sides_triangle and for the hexagonal lattice it is simulate.get_allowed_sides_hexagon.

```
{
        "amount_of_chains": 3000,
        "target_length": 1000,
        "do_perm": true,
        "w_low": 0.25,
        "w_high": 2.5,
        "next_sides_function": "simulate.get_allowed_sides_2d",
        "outputs": [
                "e2e",
                "gyration"
        ],
        "seed": 0,
        "threshold": 100
}
```

A.1.2 3D and 2D free random walk

For all 3D plots as well as 2D free random walk, we used the following config with some changes depending on the simulation. For PERM, do_perm is true and for Rosenbluth, do_perm is false. For the self avoiding random walks, next_sides_function is "simulate.get_allowed_sides_3d". For 2D free random walk, next_sides_function is "simulate.get_allowed_sides_2d_free" and for 3D free random walk it is "simulate.get_allowed_sides_3d_free"

```
{
        "amount_of_chains": 3000,
        "target_length": 100,
        "do_perm": true,
        "w_low": 0.25,
        "w_high": 2.5,
        "next_sides_function": "simulate.get_allowed_sides_3d",
        "outputs": [
                "e2e",
                "gyration"
        ],
        "seed": 0,
        "threshold": 30
}
```

A.2 Profiling results

The profiling results were generated using pyFlame before any intentional optimisations in Fig. 11, after optimising the Gyration caculations in Fig. 12 and after optimisation of the PERM step in Fig. 13.

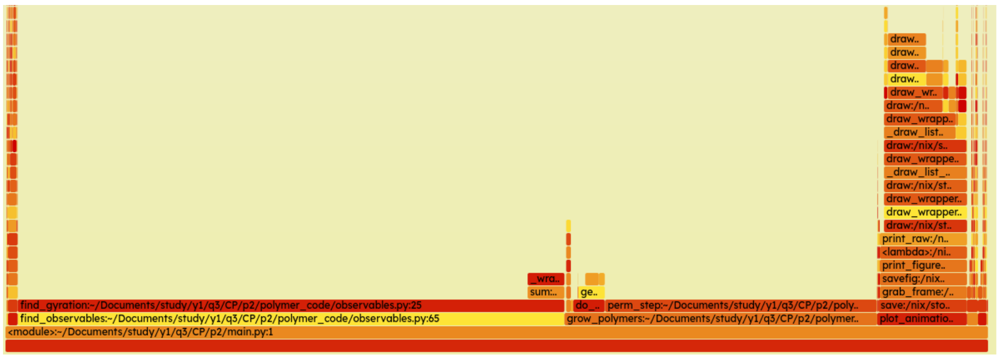


Figure 11: The profiling results before we started optimising the simulation.

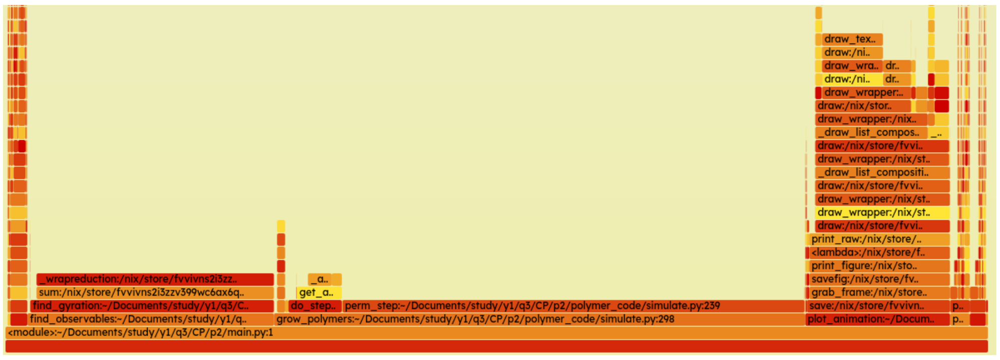


Figure 12: The profiling results using the old PERM step with `numpy.concatenate()`.

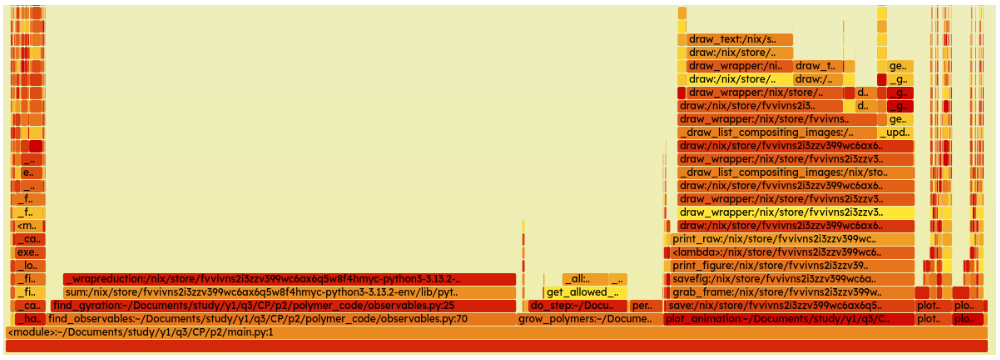


Figure 13: The profiling results after optimising the memory allocations of the PERM step.