From Rosenbluth Sampling to PERM - rare event sampling with stochastic growth algorithms

Thomas Prellberg
School of Mathematical Sciences
Queen Mary University of London
Mile End Road
London E1 4NS
United Kingdom

Abstract. We discuss uniform sampling algorithms that are based on stochastic growth methods, using sampling of extreme configurations of polymers in simple lattice models as a motivation.

We shall show how a series of clever enhancements to a fifty-odd year old algorithm, the Rosenbluth method, led to a cutting-edge algorithm capable of uniform sampling of equilibrium statistical mechanical systems of polymers in situations where competing algorithms failed to perform well. Examples range from collapsed homo-polymers near sticky surfaces to models of protein folding.

Contents

1 Introduction

A large class of sampling algorithms are based on Markov Chain Monte Carlo methods. In these lectures we shall introduce an alternative method of sampling based on stochastic growth methods. Stochastic growth means that one attempts to randomly grow configurations of interest from scratch by successively increasing the system size (usually up to a desired maximal size).

For simplicity, we shall restrict ourselves to the setting of lattice path models of linear polymers, that is, models based on random walks configurations on a regular lattice, such as the square or the simple cubic lattice. If we impose self-avoidance, i.e. if we forbid those random walk configurations that repeatedly visit the same lattice site, we obtain the model of Self-avoiding Walks (SAW), used to describe polymers in a good solvent. Monte-Carlo Simulations of SAW have been proposed as early as 1951 [6]. Extensions of SAW have been used to study a variety of different phenomena, such as polymer collapse, adsorption of polymers at a surface, and protein folding.

While this setting is rich enough to allow for the simulation of physically relevant scenarios, it is also simple enough to serve as the ideal background for the description of the particular class of stochastic growth algorithms which we shall consider in this chapter.

In contrast to expositions of uniform sampling using stochastic growth algorithms elsewhere, and appropriate for the general setting of this summer school, we shall focus in these notes directly on the central issue of uniform sampling. We therefore begin in Section 2 with a discussion of the general problem of uniform sampling in the particularly simple context of one-dimensional simple random walk. In Section 3 we shift our attention to self-avoiding walks. Here, we introduce Rosenbluth sampling as the basic algorithm, and by combining this with the ideas from the previous section, discuss the Pruned and Enriched Rosenbluth Method (PERM), and its extension to uniform sampling, flatPERM. In Section 4 we conclude with a description of an extension of stochastic growth methods to settings beyond linear polymers, called Generalized Atmospheric Rosenbluth Method (GARM).

2 Sampling of Simple Random Walks

In this section we introduce the main ideas by considering simple random walks in one spatial dimension. Starting at the origin, a random walker takes n steps, each of which is independently chosen with equal probability to be either a unit step to the

left or a unit step to the right. Generated in such a way, there are 2^n possible random walks with n steps, each having equal statistical weight.

The end-point distribution of these random walks is a binomial distribution; there are $\binom{n}{k}$ different walks ending at position $-n + 2k$, and the probability $P_{n,k}$ of ending at position $-n + 2k$ is therefore

$$P_{n,k} = \frac{1}{2^n} \binom{n}{k} \quad \text{for } k = 0, 1, \ldots n. \tag{1}$$

The subsequent algorithms are written using n and k as parameters for the length and the position of the walk (a step to the left corresponds to leaving k unchanged, whereas a step to the right corresponds to increasing k by one).

2.1 Simple Sampling

The straight-forward way of sampling these random walks is by Algorithm 1, which directly implements the probabilistic growth process. While this algorithm is fairly simple, we will use it as the skeleton for more sophisticated algorithms below.

Algorithm 1 Simple Sampling of Simple Random Walk

$s_{.,.} \leftarrow 0$
$Samples \leftarrow 0$
while $Samples < MaxSamples$ **do**
  $Samples \leftarrow Samples + 1$
  $n \leftarrow 0,\ k \leftarrow 0$
  $s_{0,0} \leftarrow s_{0,0} + 1$
  while $n < MaxLength$ **do**
    $n \leftarrow n + 1$
    Draw random number $r \in [0, 1]$
    if $r > 1/2$ **then**
      $k \leftarrow k + 1$
    end if
    $s_{n,k} \leftarrow s_{n,k} + 1$
  end while
end while

As $P_{n,0} = P_{n,n} = 2^{-n}$, the probability of generating a path ending at the endpoints decays exponentially, and even for rather short walks with length $n = 20$ or so, only about one in a million samples will hit these. For walks with several hundred steps this algorithm is therefore clearly unable to sample the distribution near the endpoints, and one needs to think of ways to modify this algorithm.

2.2 Biased Sampling

An obvious way to enhance sampling towards the endpoints of the distribution is to introduce a bias by increasing the probability of jumping, say, to the left. The effect

of this is to introduce an overall drift. This skewes the distribution, as walks are no longer generated with equal probability. If we jump to the left with probability p and jump to the right with probability $1 - p$, then walks ending at position $-n + 2k$ are now generated with probability

$$P_{n,k} = \binom{n}{k} p^{n-k}(1-p)^k \quad \text{for } k = 0, 1, \ldots n. \tag{2}$$

To correct for this bias, the statistical weight needs to be corrected, which of course leads to importance sampling. In the algorithm, this can be conveniently accomplished by changing the statistical weight of a walk by a multiplicative factor of $1/2p$ or $1/2(1 - p)$ for jumps to the left or right, respectively, as shown in Algorithm 2.

Algorithm 2 Biased Sampling of Simple Random Walk

$\quad s_{.,.} \leftarrow 0,\ w_{.,.} \leftarrow 0$
$\quad Samples \leftarrow 0$
$\quad \textbf{while } Samples < MaxSamples \textbf{ do}$
$\quad\quad Samples \leftarrow Samples + 1$
$\quad\quad n \leftarrow 0,\ k \leftarrow 0,\ Weight \leftarrow 1$
$\quad\quad s_{0,0} \leftarrow s_{0,0} + 1,\ w_{0,0} \leftarrow w_{0,0} + Weight$
$\quad\quad \textbf{while } n < MaxLength \textbf{ do}$
$\quad\quad\quad n \leftarrow n + 1$
$\quad\quad\quad \text{Draw random number } r \in [0, 1]$
$\quad\quad\quad \textbf{if } r > p \textbf{ then}$
$\quad\quad\quad\quad k \leftarrow k + 1,\ Weight \leftarrow Weight/2(1 - p)$
$\quad\quad\quad \textbf{else}$
$\quad\quad\quad\quad Weight \leftarrow Weight/2p$
$\quad\quad\quad \textbf{end if}$
$\quad\quad\quad s_{n,k} \leftarrow s_{n,k} + 1,\ w_{n,k} \leftarrow w_{n,k} + Weight$
$\quad\quad \textbf{end while}$
$\quad \textbf{end while}$

This algorithm enables us to sample in the tails of the distribution. In order to recover the full distribution, one needs to run several simulations at several values of the bias p, and subsequently combine these results. As an example, Figure 1 shows results of three simulations of 50-step random walks, with bias $p = 0.15$, 0.5, and 0.85. One can see that while the method works reasonably well, the histograms overlap minimally, and the tails are still sampled rather poorly. Even for such a small system one needs to run simulations at more values of p. For n large, the width of the individual distributions scales as $\sqrt{n}$, so that one in fact needs to run simulations for $O(\sqrt{n})$ different biases to sample the whole distribution. (If the system to be sampled shows a phase transition, the situation becomes more difficult.) Note that there are methods available to generate a joint histogram from the individual simulations. The so-called multi-histogram method [3] combines the individual histograms systematically and enables one to properly analyse statistical errors.
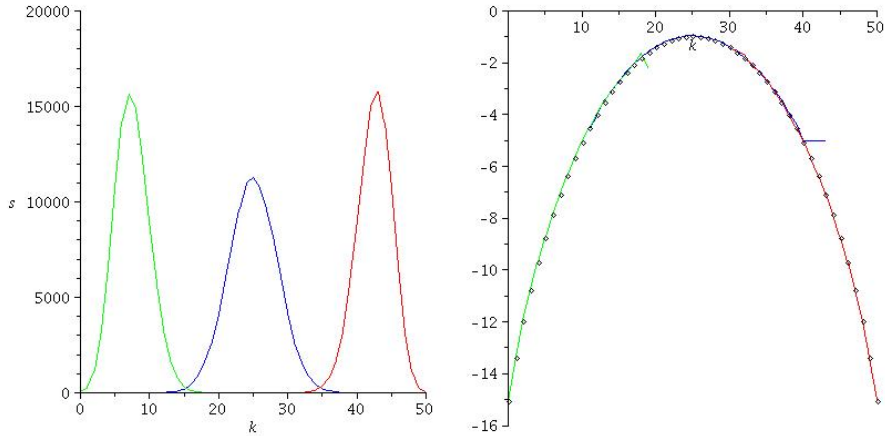
4

Figure 1: Biased sampling of simple random walk for $n = 50$ steps and bias $p = 0.85$ (green), $p = 0.5$ (blue), and $p = 0.15$ (red). For each simulation, 100000 samples were generated. On the left the total number of samples for each value of k are shown, whereas on the right the estimated distribution is shown on a logarithmic scale (the dots indicate the exact values).

2.3 Uniform Sampling

Given the disadvantages of the biased algorithm discussed so far, the question arises whether it is not possible to sample in such a way as to generate the whole end-point distribution in one single simulation. This can be achieved if we generalize the previous algorithm by allowing *local* biassing of the random walk to achieve uniform sampling at each length. For the situation discussed here, this is done by replacing the global bias p with a local bias $p_{n,k}$ that is given by

$$p_{n,k} = \frac{n + 1 - k}{n + 2} \ , \tag{3}$$

i.e. an n-step random walk at position $-n + 2k$ is required to jump to the left with probability $p_{n,k}$ and to jump to the right with probability $1 - p_{n,k}$.

> Exercise: Verify that the choice of $p_{n,k}$ given by Equation (3) leads to a uniform distribution, i.e. one that satisfies $P_{n,k} = 1/(n + 1)$. How would the result change if you considered a simple random walk in two dimensions?

The weight of the generated random walk changes accordingly by a multiplicative factor $1/2p_{n,k}$ or $1/2(1 - p_{n,k})$ for jumps to the left or right, respectively, as shown in Algorithm 3.

Algorithm 3 Uniform Sampling of Simple Random Walk

---

for $n = 0$ to $MaxLength$ **do**
    for $k = 0$ to n **do**
        $p_{n,k} \leftarrow (n + 1 - k)/(n + 2)$
    end for
end for
$s_{.,.} \leftarrow 0,\ w_{.,.} \leftarrow 0$
$Samples \leftarrow 0$
while $Samples < MaxSamples$ **do**
    $Samples \leftarrow Samples + 1$
    $n \leftarrow 0,\ k \leftarrow 0,\ Weight \leftarrow 1$
    $s_{0,0} \leftarrow s_{0,0} + 1,\ w_{0,0} \leftarrow w_{0,0} + Weight$
    while $n < MaxLength$ **do**
        $n \leftarrow n + 1$
        Draw random number $r \in [0, 1]$
        if $r > p_{n,k}$ **then**
            $k \leftarrow k + 1,\ Weight \leftarrow Weight/2(1 - p_{n,k})$
        else
            $Weight \leftarrow Weight/2p_{n,k}$
        end if
        $s_{n,k} \leftarrow s_{n,k} + 1,\ w_{n,k} \leftarrow w_{n,k} + Weight$
    end while
end while

It is noteworthy that one obtains uniform sampling over significant orders of magnitude by applying a local bias that does not vary very much at all. Figure 2 shows results of a simulation of uniformly sampled 50-step random walks. The sampling clearly is uniform, and the estimated distribution matches the exact values well across fourteen orders of magnitude.

The effect of the change from simple sampling to uniform sampling can be summarized as follows.

- Simple Sampling: n-step walks end at position $-n + 2k$ are generated with probability $P_{n,k} = \frac{1}{2^n} \binom{n}{k}$ and have statistical weight $W_{n,k} = 1$.

- Uniform Sampling: n-step walks end at position $-n + 2k$ are generated with probability $P_{n,k} = \frac{1}{n+1}$ and have statistical weight $W_{n,k} = \frac{n+1}{2^n} \binom{n}{k}$.

Necessarily in each of these cases

$$\sum_{k=0}^{n} P_{n,k} W_{n,k} = 1 \tag{4}$$

holds, as the total probability is conserved.

In our simple model system we have the advantage of *a priori* knowing the distribution to be sampled. In particular, the precise values of the local bias $p_{n,k}$ can be
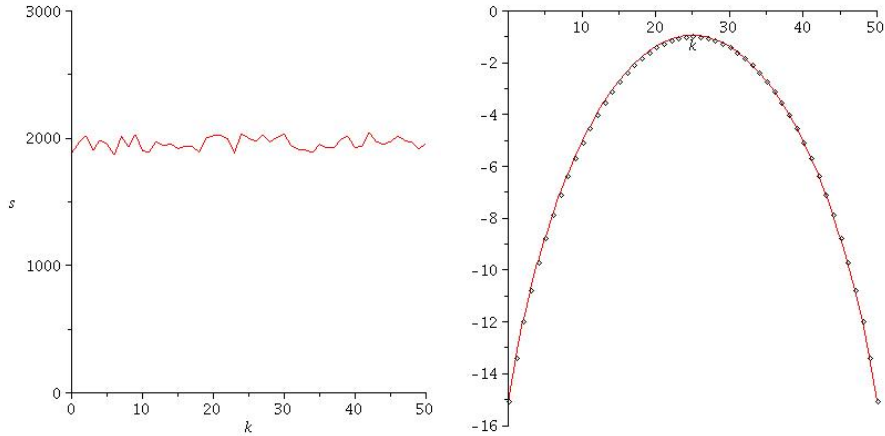
Figure 2: Uniform sampling of simple random walk for $n = 50$ steps, with 100000 samples generated. On the left the total number of samples for each value of k are shown, whereas on the right the estimated distribution is shown on a logarithmic scale (the dots indicate the exact values).

written down explicitly in a simple closed form. Generally this will not be possible, and ideally one would like to have an algorithm that is able to estimate the local bias during the simulation. The basic idea for this seems rather simple. If the local bias is incorrect, sampling will be non-uniform. This non-uniformity can then in principle be detected and corrected for. Unfortunately direct adjustment of the local bias is rather unstable. The reason for this is that a stochastic growth algorithm essentially is a blind algorithm that only knows a local growth rule. If you did the previous exercise to verify the expression for $p_{n,k}$, you will have noticed that it is important to know where walks that arrive at position k after $n+1$ steps have come from, which requires a deeper understanding of the dynamics of the growth process than that which is given by the growth rule alone.

> Exercise: Experiment with different ways of estimating the local bias $p_{n,k}$ from the data generated by the growth process. Try not to be guided by the next section. In this way you will get a feeling for the dynamics of the sampling algorithm, and good a idea might lead to new and interesting algorithm.

2.4 Pruned and Enriched Sampling

An inportant observation that enables further progress is that the performance of the algorithm depends on appropriately tuning the weight of the generated random walks, and that the uniform sampling follows from this, rather than *vice versa*. As

we have seen above, in order to achieve uniform sampling, walks have to be generated with different statistical weights.

We now adopt a different approach to achieve this. Starting with simple sampling, as in Algorithm 1, we do not bias the random walk to adjust the weights, but rather change the weight of the walk after comparing it with a target weight. This is achieved by employing pruning and enrichment procedures as detailed below. Necessarily changing the weight needs to be accompanied by adjusting the probability accordingly.

Suppose a walk has been generated that has weight w as opposed to a target weight W. In the ideal situation w is equal to W as desired. If that is not the case, either the weight w is to small, i.e. the ratio $R = w/W < 1$, or the weight w is too large, i.e. $R = w/W > 1$. In the first case we will employ pruning, i.e. we will probabilistically remove walks.

- If $R = w/W < 1$, continue growing with probability R and weight w set to W, and stop growing with probability $1 - R$.

In the second case we will employ enrichment, i.e. we will continue to grow multiple copies of the walk.

- If $R = w/W > 1$, make $\lfloor R \rfloor + 1$ copies with probability $p = R - \lfloor R \rfloor$ and $\lfloor R \rfloor$ copies with probability $1 - p$. Continue growing with the weight of each copy set to W.

While we chose to describe pruning and enrichment as different strategies, note that enrichment procedure is actually identical to the pruning procedure if $R < 1$: when $\lfloor R \rfloor = 0$ then enrichment reduces to making 1 copy with probability R and 0 copies with probability $1 - R$, which is just the pruning procedure.

Whereas in simple (or uniform) sampling the generated walks are each grown independently from length zero, pruning and enrichment leads to the generation of a large tree-like structure of more or less correlated walks grown from one seed. We call the collection of these walks a *tour* of the algorithm. The tree structure of a tour allows for successively growing all copies obtained during the enrichment in a natural way.

As this algorithm is more elaborate than the ones discussed previously, we sketch the structure of the main loop separately. At the beginning of the main loop it is assumed that there is a newly generated configuration available.

The essentials of the algorithm are summarised as follows:

repeat
  if maximal length reached **then**
    set number of enrichment copies to zero
  else
    prune/enrich step: compute number of enrichment copies
  end if
  if number of enrichment copies is zero **then**
    prune: shrink to previous enrichment

8

Algorithm 4 Pruned and Enriched Sampling of Simple Random Walk

---

for $n = 0$ to $MaxLength$ **do** {Assign target weights}
  for $k = 0$ to n **do**
    $W_{n,k} \leftarrow (n+1)\binom{n}{k}/2^n$
  end for
end for
$s_{\cdot,\cdot} \leftarrow 0$, $w_{\cdot,\cdot} \leftarrow 0$
$Tours \leftarrow 0$, $n \leftarrow 0$, $k_0 \leftarrow 0$, $Weight_0 \leftarrow 1$
$s_{0,k_0} \leftarrow s_{0,k_0} + 1$, $w_{0,k_0} \leftarrow w_{0,k_0} + Weight_0$
while $Tours < MaxTours$ **do** {Main loop}
  if $n = MaxLength$ **then** {Maximal length reached: don't grow}
    $Copy_n \leftarrow 0$
  else {pruning/enrichment by comparing with target weight}
    $Ratio \leftarrow Weight_n/W_{n,k_n}$
    $p \leftarrow Ratio \mod 1$
    Draw random number $r \in [0,1]$
    if $r < p$ **then**
      $Copy_n \leftarrow \lfloor Ratio \rfloor + 1$
    else
      $Copy_n \leftarrow \lfloor Ratio \rfloor$
    end if
    $Weight_n \leftarrow w_{n,k_n}$
  end if
  if $Copy_n = 0$ **then** {Shrink to last enrichment point or to size zero}
    while $n > 0$ and $Copy_n = 0$ **do**
      $n \leftarrow n - 1$
    end while
  end if
  if $n = 0$ and $Copy_0 = 0$ **then** {start new tour}
    $Tours \leftarrow Tours + 1$, $n \leftarrow 0$, $k_0 \leftarrow 0$, $Weight_0 \leftarrow 1$
    $s_{n,k_n} \leftarrow s_{n,k_n} + 1$, $w_{n,k_n} \leftarrow w_{n,k_n} + Weight_n$
  else {Grow by one step}
    $Copy_n \leftarrow Copy_n - 1$
    Draw random number $r \in [0,1]$
    if $r > 1/2$ **then**
      $k_{n+1} \leftarrow k_n + 1$
    else
      $k_{n+1} \leftarrow k_n$
    end if
    $Weight_{n+1} \leftarrow Weight_n$, $n \leftarrow n + 1$
    $s_{n,k_n} \leftarrow s_{n,k_n} + 1$, $w_{n,k_n} \leftarrow w_{n,k_n} + Weight_n$
  end if
end while

---

      end if
     if configuration shrunk to zero **then**
        start new tour
        store data for new configuration
     else
        decrease number of enrichment copies
        grow new step
        store data for new configuration
     end if
   until enough data is generated

A more detailed pseudocode implementation is given in Algorithm 4. Note that the number of enrichment copies to be grown from step n is stored in the array $Copy_n$. One also needs to keep track of the position of the walk at step n and the weight of the walk at step n, i.e. k and $Weight$ get replaced by the arrays k_n and $Weight_n$, respectively.
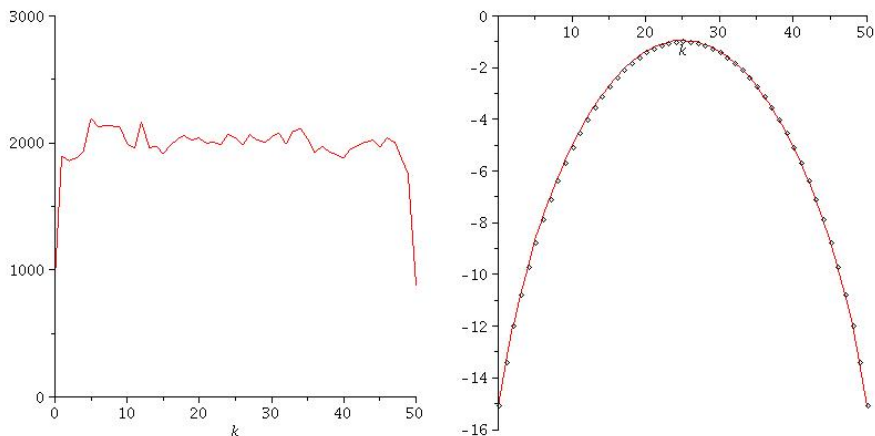

Figure 3: Pruned and enriched sampling of simple random walk for $n = 50$ steps, with 100000 tours generated. On the left the total number of samples for each value of k are shown, whereas on the right the estimated distribution is shown on a logarithmic scale (the dots indicate the exact values).

Figure 3 shows results of a simulation of 50-step random walks sampled by pruning and enrichment with respect to the *a priori* target weight $W_{n,k} = \frac{n+1}{2^n}\binom{n}{k}$. A comparison with Figure 2 shows that there is a somewhat larger fluctuation of the number of samples as compared to the results obtained from Algorithm 3. Note that the endpoints of the distribution get sampled with half the frequency; this is due to the fact that the pruning and enrichment is done strictly locally.

2.5   Blind Pruned and Enriched Sampling

Clearly pruned and enriched sampling performs less well than uniform sampling when the distribution is known. Enrichment produces correlated data, and much enrichment and pruning slows the algorithm down. Its major advantage (in addition to being able to overcome the attrition present in the Rosenbluth algorithm, as will be discussed below) is that it also performs well when the sampling distribution is not known *a priori*. It actually suffices to add one single line to the pseudocode of Algorithm 4.

Instead of a given target distribution $W_{n,k}$, one estimates $W_{n,k}$ from the generated data by computing

$$W_{n,k_n} \leftarrow (n+1) w_{n,k_n} / \sum_l w_{n,l}$$

right before determining the pruning/enrichment ratio via

$$Ratio \leftarrow Weight_n / W_{n,k_n}$$

on the fly.

As any fixed choice of the target weight $W_{n,k}$ gives an algorithm that samples correctly, one can expect that replacing the optimal choice of $W_{n,k}$ by an estimate of just that quantity obtained from the data already generated should converge towards the optimal values, and hence lead to uniform sampling. While this is not yet mathematically proven, this heuristic argument is supported by the results of numerous simulations.
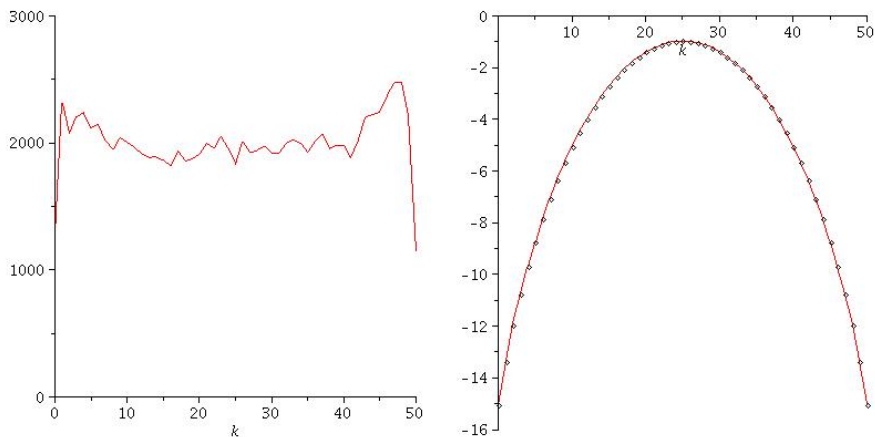

Figure 4: Pruned and enriched sampling of simple random walk for $n = 50$ steps, with 100000 tours generated. On the left the total number of samples for each value of k are shown, whereas on the right the estimated distribution is shown on a logarithmic scale (the dots indicate the exact values).

For the case of simple random walk discussed here, Figure 4 shows results of a simulation of 50-step random walks sampled by blind pruning and enrichment. A

comparison with Figure 3 shows while the fluctuation of the number of samples has increased, sampling is again reasonably uniform, and the algorithm is converging to the correct distribution.

Note that while there is some freedom in the choice of pruning and enrichment strategies, we have opted for the simplest version in which the weight w is forced to be *equal* to the target weight W. Alternatively one can allow for some fluctuations around the target weight, with the aim of reducing the overall amount of enrichment and pruning, but doing so invariably requires the introduction of parameters into an essentially parameter-free algorithm, and finding reasonable values for these parameters can be somewhat fickle.

The algorithms from this section generalize immediately to the simulation of simple random walks in higher spatial dimensions, say, on the square lattice $\mathbb{Z}^2$. One generates an $(n+1)$-step random walk from an n-step random walk by stepping to one of the neighboring lattice sites, which is chosen uniformly at random. In such a way, each n-step random walk is generated with equal probability. On the square lattice, this probability is $1/4^n$, as each lattice site on the square lattice has four neighbouring sites.

> Exercise: Extend the uniform sampling and the blind pruned and enriched sampling algorithms to simple random walk on the square lattice, using
>
> - uniform or pruned and enriched sampling in x-coordinate, bias in y-coordinate,
>
> - uniform or pruned and enriched sampling jointly in both x and y-coordinates.
>
> For uniform sampling, you need to determine the target weight distribution and derive the associated local bias.

3   Sampling of Self-Avoiding Walks

In this section we want to consider the simulation of self-avoiding random walks (SAW), i.e. random walks obtained by forbidding any random walk that contains multiple visits to a lattice site. SAW is the canonical lattice model for polymers in a good solvent. Moreover, it forms the basis for more realistic models of polymers with physically and biologically relevant structure, as indicated in Figure 5.

However, the introduction of self-avoidance turns a simple Markovian random walk without memory into a complicated non-Markovian random walk; when growing a self-avoiding walk, one needs to test for self-intersection with all previous steps, leading to a random walk with infinite memory.
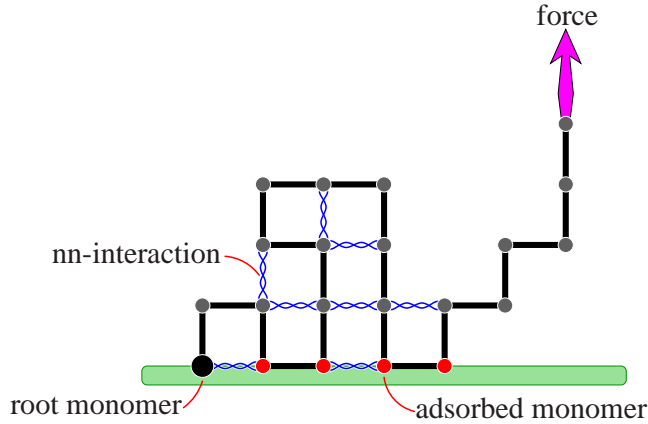
Figure 5: A lattice model of a polymer tethered to a sticky surface under the influence of a pulling force.

3.1 Simple Sampling

Algorithm 5 Simple Sampling of Self-Avoiding Walk

$s. \leftarrow 0$
$Samples \leftarrow 0$
while $Samples < MaxSamples$ **do**
  $Samples \leftarrow Samples + 1$
  $n \leftarrow 0$, Start at origin
  $s_0 \leftarrow s_0 + 1$
  while $n < MaxLength$ **do**
    Draw one of the neigboring sites uniformly at random
    if Occupied **then**
      Reject entire walk and exit loop
    else
      Step to new site
      $n \leftarrow n + 1$
      $s_n \leftarrow s_n + 1$
    end if
  end while
end while

It is straight-forward to generate SAW by simple sampling. Generating an n-step self-avoiding walk with the correct statistics, i.e. such that every walk is generated with the same probability, is equivalent to generating n-step random walks and reject those random walks that self-intersect. Algorithm 5 accomplishes this by generating

two-dimensional random walks and rejecting the *complete* configuration when self-intersection occurs.

At each step, the walk has four possibilities to continue, and chooses one of these with probability $p = 1/4$. Therefore an estimator for the total number of n-step SAW after S samples have been generated is given by $4^n s_n/S$.

Generating SAW with simple sampling is very inefficient. There are 4^n n-step random walks, but only about 2.638^n n-step self-avoiding walks on the square lattice. The probability of successfully generating an n-step self-avoiding walk therefore decreases exponentially fast, leading to very high attrition[1]. Longer walks are practically inaccessible, as seen in Figure 6.
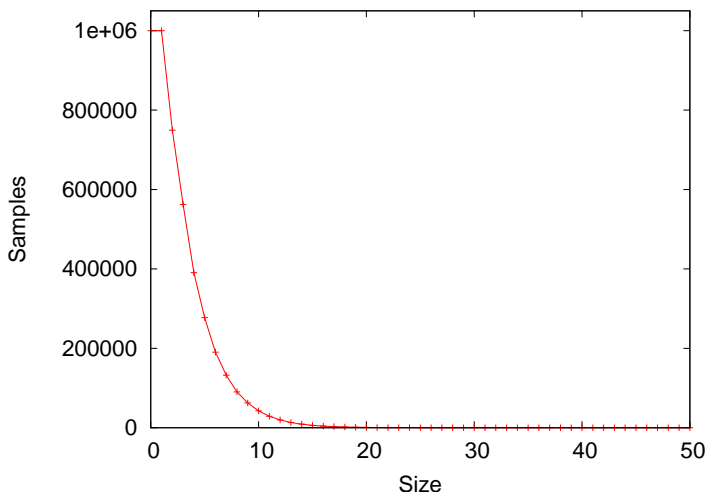

Figure 6: Attrition of started walks generated with Simple Sampling. From 10^6 started walks none grew more than 35 steps.

3.2 Rosenbluth Sampling

A slightly improved sampling algorithm was proposed in 1956 by Rosenbluth and Rosenbluth. The basic idea is to avoid self-intersections by only sampling from the steps that lead to self-avoiding configurations. In this way, the algorithm only terminates if the walk is trapped in a dead end and cannot continue growing. While this still happens exponentially often, Rosenbluth sampling can produce substantially longer configurations than simple sampling.

---

[1] The algorithm can be improved somewhat by forbidding immediate reversals of the random walk, but the attrition remains exponential

While simple sampling generates all configurations with equal probability, configurations generated with Rosenbluth sampling are generated with different probabilities. To understand this in detail, it is helpful to introduce the notion of an *atmosphere* of a configuration; this is the number of ways in which a configuration can continue to grow. For one-dimensional simple random walks the atmosphere is always two, for two-dimensional simple random walks on the square lattice the atmosphere is always four (and if one forbids immediate self-reversals, the atmosphere is always three except for the very first step). However, for self-avoiding walks on the square lattice the atmosphere is a configuration-dependent quantity assuming values between four (for the first step) and zero (for a trapped configuration that cannot be continued). We shall denote the atmosphere of a configuration ϕ by $a(\phi)$. If it is clear from the context, we will drop the argument and speak about the atmosphere a.

If a configuration has atmosphere a, this means that there are a different possibilities of growing the configuration, and each of these can get selected with probability $p = 1/a$. To balance this, the weight of this configuration is therefore multiplied by the atmosphere a. An n-step walk grown by Rosenbluth sampling therefore has weight

$$W_n = \prod_{i=0}^{n-1} a_i \ ,$$

where a_i are the atmospheres of the configuration after i growth steps. This walk is generated with probability $P_n = 1/W_n$, so that $P_n W_n = 1$ as required. Algorithm 6 shows a pseudocode implementation of Rosenbluth sampling.

Algorithm 6 Rosenbluth Sampling of Self-Avoiding Walk

$s_. \leftarrow 0, w_. \leftarrow 0$
$Samples \leftarrow 0$
while $Samples < MaxSamples$ **do**
  $Samples \leftarrow Samples + 1$
  $n \leftarrow 0, Weight \leftarrow 1$, Start at origin
  $s_0 \leftarrow s_0 + 1, w_0 \leftarrow w_0 + Weight$
  while $n < MaxLength$ **do**
    Create list of neighboring unoccupied sites, determine the atmosphere a
    if $a = 0$ (walk cannot continue) **then**
      Reject entire walk and exit loop
    else
      Draw one of the neigboring unoccupied sites uniformly at random
      Step to new site
      $n \leftarrow n + 1, Weight \leftarrow Weight \times a$
      $s_n \leftarrow s_n + 1, w_n \leftarrow w_n + Weight$
    end if
  end while
end while

Figure 7 shows the improvement gained by Rosenbluth sampling over simple sampling.
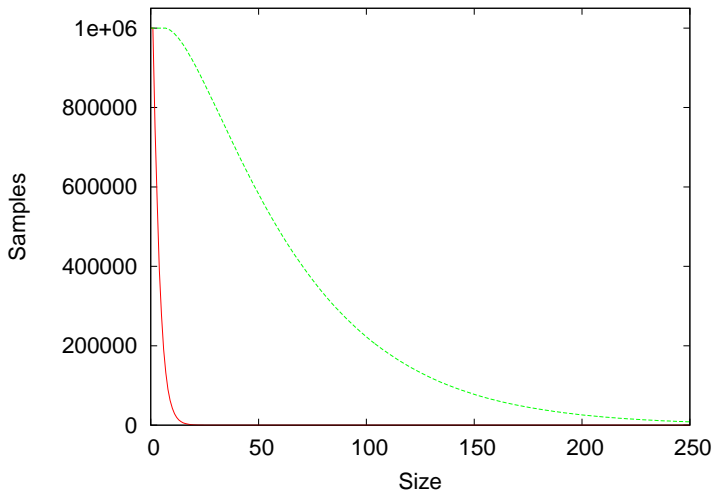

Figure 7: Attrition of started walks generated with Rosenbluth Sampling compared with Simple Sampling. Walks with a few hundred steps become accessible.

3.3   Pruned and Enriched Rosenbluth Sampling

It took four decades before Rosenbluth sampling was improved upon. In 1997 Grassberger augmented Rosenbluth sampling with pruning and enrichment strategies, calling the new algorithm Pruned and Enriched Rosenbluth Method, or PERM [4]. There are a variety of pruning and enrichment strategies that are possible, and the strategies used in [4] were somewhat different from the ones encountered earlier in Section 2.5. However, we shall present here our simplified version of PERM only. For alternate versions and enhancements we refer to [5] and references therein.

From the point of view of the previously encountered pruned and enriched sampling of simple random walk, the only thing that is different here is that the atmosphere of the walk is variable and can be zero. This can be incorporated quite easily as follows.

repeat
    if zero atmosphere or maximal length reached **then**
       set number of enrichment copies to zero
    else
       prune/enrich step: compute number of enrichment copies
    end if
    if number of enrichment copies is zero **then**

```
      prune: shrink to previous enrichment
    end if
    if configuration shrunk to zero then
      start new tour
      store data for new configuration
    else
      decrease number of enrichment copies
      if positive atmosphere then
        grow new step
        store data for new configuration
      end if
    end if
  until enough data is generated
```

Note that in case of constant atmosphere this reduced precisely to the pruned and enriched sampling for simple random walks encountered earlier. Algorithm 7 contains a more detailed pseudo-code version of PERM for self-avoiding walks.

Figure 8 shows the significant improvement gained by adding pruning and enrichment strategies to Rosenbluth Sampling.
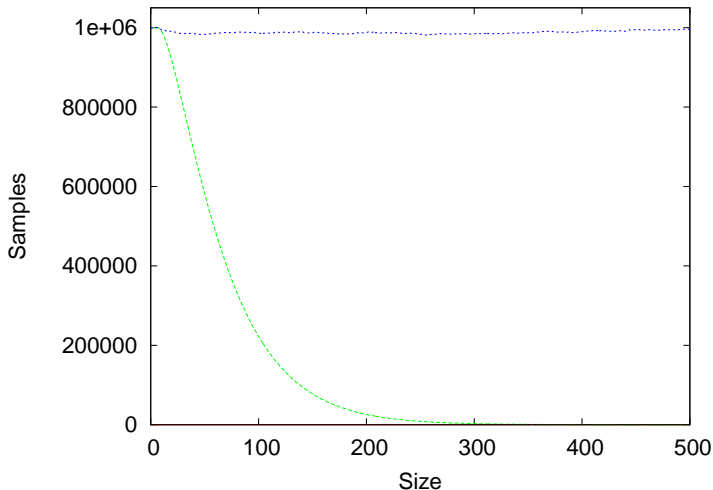

Figure 8: Attrition of started walks with PERM compared with Rosenbluth Sampling. In the case of PERM, a virtually constant number of samples is obtained.

Algorithm 7 Pruned and Enriched Rosenbluth Sampling of Self-Avoiding Walks

---

$s. \leftarrow 0$, $w. \leftarrow 0$
$Tours \leftarrow 0$, $n \leftarrow 0$, $Weight_0 \leftarrow 1$
Start new walk with step size zero
$a \leftarrow 0$, $Copy_0 \leftarrow 1$
$s_0 \leftarrow s_0 + 1$, $w_0 \leftarrow w_0 + Weight$
while $Tours < MaxTours$ **do** {Main loop}
  if $n = MaxLength$ or $a = 0$ **then** {Maximal length reached or atmosphere zero: don't grow}
    $Copy_n \leftarrow 0$
  else {pruning/enrichment by comparing with target weight}
    $Ratio \leftarrow Weight_n / w_n$
    $p \leftarrow Ratio \mod 1$
    Draw random number $r \in [0, 1]$
    if $r < p$ **then**
      $Copy_n \leftarrow \lfloor Ratio \rfloor + 1$
    else
      $Copy_n \leftarrow \lfloor Ratio \rfloor$
    end if
    $Weight_n \leftarrow w_n$
  end if
  if $Copy_n = 0$ **then** {Shrink to last enrichment point or to size zero}
    while $n > 0$ and $Copy_n = 0$ **do**
      Delete last site of walk
      $n \leftarrow n - 1$
    end while
  end if
  if $n = 0$ and $Copy_0 = 0$ **then** {start new tour}
    $Tours \leftarrow Tours + 1$,
    Start new walk with step size zero
    $a \leftarrow 0$, $Copy_0 \leftarrow 1$
    $s_0 \leftarrow s_0 + 1$, $w_0 \leftarrow w_0 + Weight$
  else
    Create list of neighboring unoccupied sites, determine the atmosphere a
    if $a > 0$ **then**
      $Copy_n \leftarrow Copy_n - 1$
      Draw one of the neigboring unoccupied sites uniformly at random
      Step to new site
      $n \leftarrow n + 1$, $Weight_n \leftarrow Weight_n \times a$
      $s_n \leftarrow s_n + 1$, $w_n \leftarrow w_n + Weight_n$
    end if
  end if
end while

3.4   Flat Histogram Rosenbluth Sampling

The next advance was made by two groups in 2003/4. Motivated by work of Wang and Landau on uniform sampling [10], Bachmann and Janke, using ideas from Berg and Neuhaus [2], implemented Multicanonical PERM [1]. This was followed by Prellberg and Krawczyk [7], who designed flatPERM, a flat-histogram version of PERM estimating directly the microcanonical density of states.

Within the context of the algorithms developed here, incorporating uniform sampling into PERM is straightforward. First we note that PERM already is a uniform sampling algorithm in system size. This is not apparent at all from the algorithm, as the guiding principle has been to adjust pruning and enrichment with respect to a target weight, not with respect to any criterion of poor local sampling. It is rather that uniform sampling is a consequence of adjusting pruning and enrichment around the desired target weight.
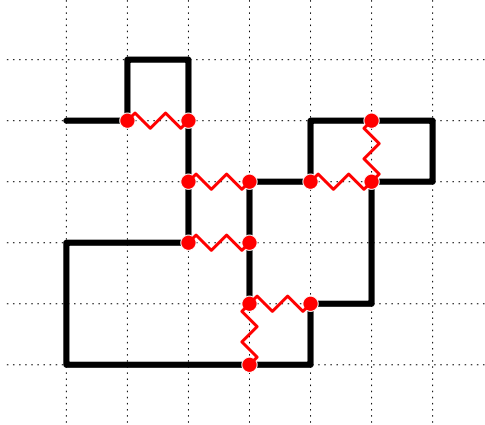

Figure 9: An interacting self-avoiding walk on the square lattice with $n = 26$ steps and $m = 7$ contacts.

It is therefore reasonable (and very much in the spirit of the previous section) to extend PERM to a microcanonical version, in which configurations of size n are separated with respect some additional parameter. One simply determines this parameter when growing the configuration and stores the data by binning with respect to this additional parameter. Then, when considering pruning and enrichment, the target weight is computed from the binned data. More precisely, if the additional parameter is called m, storing the data is changed from

$s_n \leftarrow s_n + 1,\ w_n \leftarrow w_n + Weight_n$

to

$s_{n,m} \leftarrow s_{n,m} + 1,\ w_{n,m} \leftarrow w_{n,m} + Weight_n$

and computing enrichment ratio is changed from

$Ratio \leftarrow Weight_n/w_n$

to

$Ratio \leftarrow Weight_n/w_{n,m}$

and this is about it.

In the previous section this additional parameter has been the end-point position of the random walk. Here, we shall consider by example the case of *interacting self-avoiding walks*, where each walk configuration has an energy proportional to the number of non-consecutive nearest-neigbour contacts between occupied lattice sites.
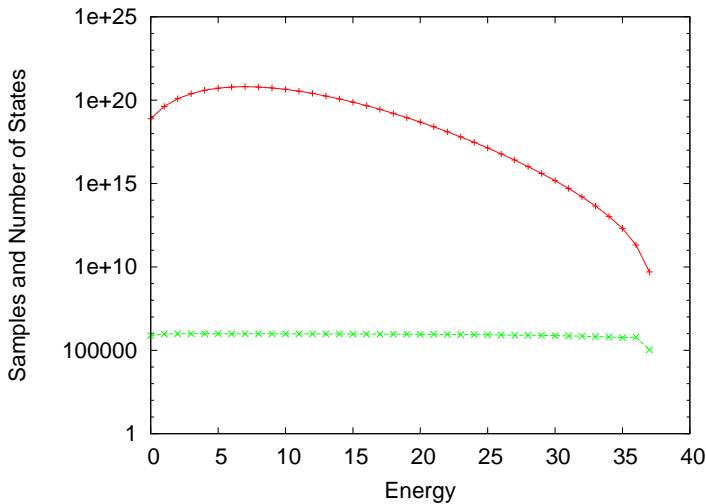

Figure 10: Number of States of Interacting Self-Avoiding Walks with 50 steps at fixed energy estimated from 10^6 flatPERM tours. The lower graph shows the number of actually generated samples for each energy.

Figure 10 shows the simulation results of a simulation of interacting self-avoiding walks of up to 50 steps using 10^6 tours starting at size zero. This led to the generation of about 10^6 samples for each value of m at $n = 50$ steps, and enabled the estimation of the number of states over ten orders of magnitude.

Figure 11 shows the corresponding simulation results for all intermediate lengths from the same run. While the histogram of samples is not as flat as in the case of random walks discussed above, especially for large values of m, it is reasonably flat on a logarithmic scale and leads to sufficiently many samples for each histogram bin. Reference [7] contains results of a simulation of interacting self-avoiding walks with up to $n = 1024$ steps, where the density of states ranges over three hundred orders of magnitude, all obtained from one single simulation.
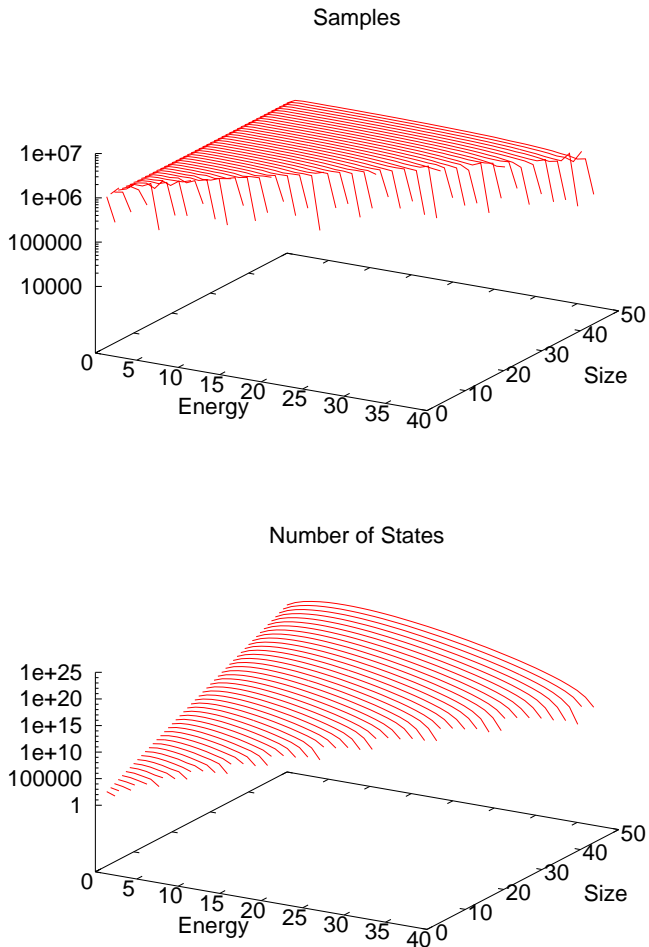
Figure 11: Interacting Self-Avoiding Walks with up to 50 steps generated with flatPERM. The upper figure shows that a roughly constant number of samples is obtained across the whole range of sizes and energies, and the lower figure shows the estimated number of states for a given Size n and Energy m.

> Exercise: Use flatPERM to uniformly sample self-avoiding walks with respect to several different parameters, such as number of contacts, position of the end point, number of turns in the walk (measuring stiffness), etc. While some parameters are immediately accessible by the algorithm or easy to calculate, others will necessitate a more careful design of data structures.

4 Extensions

In the excellent review article [5] the algorithms of Section 3 and several extensions are discussed.

Extensions of algorithms are usually motivated by the need to simulate systems inaccessible with established algorithm. For example, algorithms based on Rosenbluth sampling are well-suited to the simulation of objects that can be grown uniquely from a seed. In the case of linear polymer models, this is accomplished by appending a step to the end of the current configuration[2].

However, if one wants to simulate polymers with a more complicated structure, such as branched polymers, there no longer is an easy way to uniquely grow a configuration. A lattice model for a two-dimensional branched polymer is given by lattice trees, i.e. trees embedded in the lattice $\mathbb{Z}^2$. For a given lattice tree it is no longer clear how it has been grown from a seed; this could have happened in a variety of ways.

4.1 Generalized Atmospheric Rosenbluth Sampling

It turns out that there is an extension to Rosenbluth sampling, called Generalized Atmospheric Rosenbluth Method, or GARM [8], that is suitable for these more complicated growth processes. The key idea is to generalize the notion of atmosphere by introducing an additional negative atmosphere a^- indicating in how many ways a configuration can be reduced in size. For linear polymers the negative atmosphere is always unity, as there is only one way to remove a step from the end of the walk. However, for a given lattice tree the removal of any leaf of the tree gives a smaller lattice tree, and the negative atmosphere a^- can assume rather large values.

Surprisingly there is a very simple extension to the Rosenbluth weights discussed above. If a configuration has negative atmosphere a^-, this means that there are a^- different possibilities in which the configuration could have been grown. An n-step configuration grown by GARM therefore has weight

$$W_n = \prod_{i=0}^{n-1} \frac{a_i}{a_{i+1}^-} \ , \tag{5}$$

---

[2] In a more abstract setting, Rosenbluth sampling has for example been used to study the number of so-called pattern-avoiding permutations. Permutations can be grown easily by inserting the number $n + 1$ somewhere into a permutation of the numbers $\{1, 2, \ldots, n\}$, allowing for easy implementation of Rosenbluth sampling.

where a_i are the (positive) atmospheres of the configuration after i growth steps, and a_i^- are the negative atmospheres of the configuration after i growth steps. It can be shown that the probability of growing this configuration is $P_n = 1/W_n$, so again $P_n W_n = 1$ holds as required.

The implementation of GARM is not any more complicated than the implementation of Rosenbluth sampling. Algorithm 6 gets changed minimally by inserting the lines

Compute negative atmosphere a^-
$Weight \leftarrow Weight/a^-$

immediately after having grown the configuration.

Algorithm 8 Generalised Atmospheric Sampling

$s. \leftarrow 0, w. \leftarrow 0$
$Samples \leftarrow 0$
while $Samples < MaxSamples$ **do**
  $Samples \leftarrow Samples + 1$
  $n \leftarrow 0, Weight \leftarrow 1$, Start with seed configuration
  $s_0 \leftarrow s_0 + 1, w_0 \leftarrow w_0 + Weight$
  while $n < MaxSize$ **do**
    Create list of growth possibilities, determine the atmosphere a
    if $a = 0$ (no growth possible) **then**
      Reject entire configuration and exit loop
    else
      Draw one of the growth possibilities uniformly at random
      Grow configuration
      $n \leftarrow n + 1, Weight \leftarrow Weight \times a$
      Compute negative atmosphere a^-
      $Weight \leftarrow Weight/a^-$
      $s_n \leftarrow s_n + 1, w_n \leftarrow w_n + Weight$
    end if
  end while
end while

While implementing GARM is quite straightforward, there generally is a need for more complicated data structures for the simulated objects, and one needs to find efficient algorithms for the computation of positive and negative atmospheres.

It is now possible to add pruning and enrichment to GARM, and to extend this further to flat histogram sampling, just as has been described in the previous section for Rosenbluth sampling.

For further extensions to Rosenbluth sampling, and indeed many more algorithms for simulating self-avoiding walks, as well as applications, see [5].

References

[1] M. Bachmann and W. Janke, Phys. Rev. Lett. **91** 208105 (2003)

[2] B. A. Berg and T. Neuhaus, Phys. Lett. B **267** 249 (1991)

[3] A. M. Ferrenberg, Phys. Rev. Lett. **61** 2635 (1988)

[4] P. Grassberger, Phys. Rev E **56** 3682 (1997)

[5] E. J. Janse van Rensburg, J. Phys. A **42** 323001 (2009)

[6] G. W. King, in *Monte Carlo Method*, volume 12 of *Applied Mathematics Series*, National Bureau of Standards, 1951

[7] T. Prellberg and J. Krawczyk, Phys. Rev. Lett. **92** 120602 (2004)

[8] A. Rechnitzer and E. J. Janse van Rensburg, J. Phys. A **41** 442002 (2008)

[9] M. N. Rosenbluth and A. W. Rosenbluth, J. Chem. Phys. **23** 356 (1955)

[10] F. Wang and D. P. Landau, Phys. Rev. Lett. **86** 2050 (2001)