Monte Carlo Simulation of Lattice Polymer Models

Gerard T. Barkema

1 Introduction

1.1 Polymers and random walks

Polymers are substances whose molecules have high molar masses and are composed of a large number of repeating units. They occur naturally, and are also produced in large quantities. The materials commonly called plastics are all synthetic polymers. Polymers consist of a large number of units called monomers, which are joined sequentially to form a long chain. Perhaps the simplest polymer is polyethylene (PE), which consists of many repeating CH$_2$ units. It is produced by the addition polymerization of ethylene, CH$_2$=CH$_2$ (ethene). Other common synthetic polymers are polypropene (PP), made of polymerized CH$_2$=CH-CH$_3$ (propene), polystyrene (PS), made of polymerized CH$_2$=CH-C$_6$H$_5$ (styrene), and polyvinyl chloride (PVC), which is polymerized CH$_2$=CHCl (chloroethene). Naturally occurring polymers are for instance proteins.

The length of a polymer can be expressed in the number of monomers, or alternatively in the number of bonds between monomers. Here, we adapt the latter choice, so that a polymer of length $N$ consists of $N+1$ monomers, connected through $N$ bonds. The shape (configuration) of a polymer of length $N$ is characterized by specifying for each monomer $i=0\ldots N$ its position $\mathbf{r}_i$. Alternatively, one can specify the bond directions $\mathbf{u}_i \equiv \mathbf{r}_i - \mathbf{r}_{i-1}$, in which $i=1\ldots N$. The chemical bond between neighboring monomers is usually rather stiff, so that in good approximation the bond length equals some constant:

$$|\mathbf{u}_i| \approx b.$$ (1)

However, the angle between consecutive bonds is usually less constrained, with the consequence that polymers show a high degree of flexibility. Often, the correlation between the orientation of two bonds decays exponentially with their separation along the chain:

$$\langle \mathbf{u}_{i+k} \cdot \mathbf{u}_i \rangle \approx b^2 \exp(-kb/l_p).$$ (2)

In this expression, as well as in the remainder of this chapter, the brackets denote averaging over all possible polymer configurations. The parameter $l_p$ in this expression is called the persistence length.
Two related fundamental geometric properties that characterize a polymer are the mean end-to-end length, given by

$$\langle r^2_e(N) \rangle = \langle |r_N - r_0|^2 \rangle,$$

and the radius of gyration, given by

$$\langle r^2_g(N) \rangle = \langle \frac{1}{N+1} \sum_{i=0}^{N} (r_i - r_{cm})^2 \rangle,$$

in which $r_{cm}$ is the center of mass of the polymer. The radius of gyration is easily accessible from the structure factor as obtained in scattering experiments, while the mean end-to-end length is computationally easier. They are usually related via a constant factor, which is independent of polymer length, persistence length, and all other variables that one typically encounters.

If equations (1) and (2) were exact, we can obtain analytic expressions in the limit of long polymers ($l_p \ll b N$):

$$\langle r^2_e(N) \rangle = \exp\left(\frac{b}{l_p}\right) + 1 - \exp\left(\frac{b}{l_p}\right) - 1 \frac{b^2 N}{6} \langle r^2_e(N) \rangle,$$

$$\langle r^2_g(N) \rangle = 6 \langle r^2_e(N) \rangle.$$

Within theoretical physics, research is usually done on model polymers with zero persistence length, which are known as freely jointed chains. From a geometric point of view, a good approximation for polymers with a non-zero persistence length is obtained by identifying the freely jointed chain with the same length along the chain, as well as the same expected end-to-end length and radius of gyration. Assuming the approximation eq.(2), this is the freely jointed chain with bond lengths $b_{eff} = \left[\left(\frac{e^{b/l_p} + 1}{e^{b/l_p} - 1}\right)\right] b$ and containing $N_{eff} = (b/b_{eff}) N$ bonds.

A further idealization is to restrict the angle between consecutive bonds to 0, 90 or 180 degrees. This has the result that all monomers of the effective chain reside on a lattice with lattice spacing $b_{eff}$. With this sequence of approximations, we have reduced the polymer to a so-called random walk on a lattice: polymer configurations are exactly equal to the sequence of lattice sites visited by a random walker that starts in location $r_0$ and makes $N_{eff}$ steps, each time to a neighboring site.

1.2 self-avoidance

It is known from experiment that real-life polymers in highly diluted solutions — such that they do not overlap — are in many respects essentially different from random walks. Most notably, their radius of gyration does not grow with the square root of the polymer length. This different behavior can be largely attributed to the property that monomers have finite dimensions, so that a polymer cannot intersect itself; a pair of monomers cannot occupy nearby
positions. To incorporate this into the random-walk model on a lattice, an additional constraint is enforced: no lattice site is visited more than once. The resulting polymer model is known as a **self-avoiding walk**. Self-avoiding walks are much harder to treat analytically, and consequently far fewer analytic results are known for them, as compared to random walks. As often, computer simulations are employed to fill the gap.

![Fig. 1. Example of a random-walk and a self-avoiding walk with 15 monomers](image)

A first quantity of obvious interest is the mean squared end-to-end length, as defined in eq. (3). As shown before, this quantity is exactly known for random walks. It is not known exactly, however, for two- and three-dimensional self-avoiding walks of arbitrary finite lengths. In the limit of long polymers, the mean squared end-to-end length scales as

\[
\langle r^2_e(N) \rangle \sim N^{2\nu}.
\]

(7)

The exponent \(\nu\), known as the **growth exponent**, is an important quantity, since it does not depend on microscopic details of the lattice. On all regular two-dimensional lattices the growth exponent is known to be exactly \(\nu_{d=2} = \frac{3}{4}\), and on all regular three-dimensional lattices it is numerically determined to be \(\nu_{d=3} \approx 0.588\). The universality of the growth exponent indicates that also the mean end-to-end length of real-life self-avoiding polymers scales with polymer length in the same way, with the same exponent, even though real-life polymers do not live on a lattice.

Another quantity of interest is the number of distinct configurations that the polymer can take. As usual in statistical physics, this quantity is known as the **partition function**, and indicated with the symbol \(Z\).\(^1\) For random walks, with the first monomer placed in the origin of the lattice, this quantity is also exactly known. For polymers on a \(d\)-dimensional hypercubic lattice (which

---

\(^1\) In general in statistical physics, the partition function is obtained by the summation over all configurations of the Boltzmann weights; since energy does not play a role here, all Boltzmann weights are equal, and the partition function is simply the number of configurations.
is the square lattice in two, and the cubic lattice in three dimensions), the partition function is

$$Z_{RW}(N) = (2d)^N,$$

(8)

since at every step the polymer can be extended in $2d$ equally likely directions.

For two- or three-dimensional self-avoiding walks, the partition sum $Z_{SAW}$ is not known exactly for all lengths. Its scaling for long polymers is however known to be

$$Z_{SAW} \approx A \cdot N^{\gamma_s - 1} \cdot \mu^N.$$

(9)

The three parameters $A$, $\gamma_s$ and $\mu$ are not equally important. Two of them, $A$ and $\mu$, depend on microscopic details of the lattice. Only one of them, $\gamma_s$, is universal and therefore tells us something about real-life polymers that do not reside on a lattice. In two dimensions, its value is known exactly as $\gamma_s = \frac{43}{32}$ [1]. In three dimensions, its value is approximately $\gamma_s = 1.16$.

## 2 Computer simulation of equilibrium properties

In the coming sections, we will describe several computational approaches to simulate a single self-avoiding walk of $N$ steps on a square or cubic lattice. The algorithms are discussed for self-avoiding walks in which the monomer do not have interactions other than self-avoidance. In most simulations of single polymers, the nature of the interactions and their effect on the polymer are the topic of research. The simulation methods are however based on those proposed for non-interacting self-avoiding walks, and easier explained in that context.

### 2.1 complete enumeration

For short polymers, it is possible to generate all polymer configurations by computer. The partition function $Z$ is then obtained by counting these, and the mean squared end-to-end length $\langle r^2_e \rangle$ by averaging of all squared end-to-end lengths. We will now present one possible implementation on a computer that does this, because it provides the basis for some Monte Carlo methods that will be discussed below.

Probably the easiest implementation uses a concept that is known in computer science as a recursive program, in which a subroutine calls itself. The main program initializes the partition sum $Z$ and the sum of squared end-to-end distances $S_c$ to zero, calls \texttt{step(0,0,0)}, after which it reports the final results for $Z$ and $\langle r^2_e \rangle = S_c/Z$. The heart of the program for enumerating random-walk polymers on a square lattice is a recursive subroutine \texttt{step(x,y,i)}, in which the random walker has already made $i$ steps, after which it has arrived at site $(x,y)$. If the polymer has reached the required
length (i.e., \(i = \text{equal to } N\)), \(Z\) is incremented by one, and \(S_c\) is incremented by \(x^2 + y^2\). Otherwise, the subroutine calls itself four times, with arguments \((x \pm 1, y, i + 1)\) and \((x, y \pm 1, i + 1)\). With this approach, all random walks of length 10 (slightly over one million) can be generated in a fraction of a second.

It is more interesting to enumerate all self-avoiding walks, whose properties are not known exactly. The basic approach is as for random-walk polymers, with the addition that the recursive subroutine \texttt{step} is only allowed to increment \(Z\) and \(S_c\), or to call itself, if the site \((x, y)\) is not visited before. With this addition, all self-avoiding walks of length 20 (close to 10 billion) can be generated in a minute.

The bottleneck of this complete enumeration method is of course the exponential growth of the number of configurations with increasing length. Although a length of 20 is easily reached on a standard computer, a length of 30 already requires a fast computer and a lot of patience, and a length of 40 will still be out of the question with this method, for at least a decade after the publication of this book. Luckily, methods exist that are much more efficient (and unfortunately also much more elaborate). Using an approach they called \textit{Finite-lattice method plus transfer matrices}, Conway, Enting and Guttmann were able to extend the complete enumeration to length 39. They reported that on a square lattice, \(Z_{SAW}(39) = 113\,101\,676\,587\,853\,932\). A few years later, with a parallel computer, Conway and Guttmann [3] went even as far as to length 51, and obtained \(Z_{SAW}(51) = 14\,059\,415\,980\,606\,650\,644\,844\). Obviously, generating all these self-avoiding walks is out of the question.

### 2.2 Statistical enumeration

An alternative approach, strongly related to the complete enumeration described in the previous section, is known as \textit{statistical enumeration}. Instead of exploring all possible directions in which we can extend a polymer, we randomly choose only one such direction. Extending the polymer one step at a time, we keep track of the number of choices \(m_i\) available to us after \(i\) steps, and also keep track of the \textit{Rosenbluth weight}, after Rosenbluth and Rosenbluth [4], given by

\[
\text{\(w_i = \prod_{i'=0}^{i} (m_{i'})\).} \tag{10}
\]

Once the polymer has reached the desired length \(N\), we determine the observable we are interested in, for instance its squared end-to-end length \(r_e^2\). We repeat this procedure many times, and thus obtain a large number of values \(\{w_k, (r_e^2)_k\}, k = 1 \ldots K\). In the ensemble that we want to study, each polymer configuration should contribute equally, whether it is likely to be generated or not. However, the procedure above does not generate all polymer configurations with equal probability. Consequently, a straight average of all values
\[(r_e^2)_k \text{ does not give a correct estimate of the mean squared end-to-end length.}
\]
The probability to obtain a certain polymer configuration is the product of the probabilities for all the choices that were made. Every time that one out of \(m_i\) possible ways to extend the polymer was chosen adds a factor of \(1/m_i\) to this probability. The probability to obtain the final configuration is thus equal to the inverse of its Rosenbluth weight. To properly correct for the over-sampling of configurations with a large inverse Rosenbluth weight and let each polymer configuration contribute equally to the result, each measurement has to be weighted with its Rosenbluth weight. The correct estimation of the average squared end-to-end length is thus obtained by
\[
\langle r_e^2 \rangle = \frac{\sum_k w_k (r_e^2)_k}{\sum_k w_k},
\]
(11)

It can happen that after a number of steps, but well before the desired length \(N\) is reached, all nearest-neighbor sites of the end of the polymer are occupied, and the polymer cannot be extended. The generation of the polymer cannot be completed up to the desired length, and thus its squared end-to-end length after \(N\) steps cannot be computed! Luckily, at that point the Rosenbluth weight (eq. (10) has become zero anyway, so we are allowed to simply stop with this polymer as it would contribute zero to both the numerator and denominator in eq. (11).

To illustrate the properties of the statistical enumeration method that we outlined so far, we performed simulations for self-avoiding-walks on a square lattice with lengths 39 (for which the exact result is mentioned above), 100, 150 and 200. For each length we did 1000 runs, each with \(K = 10^5\) iterations. Such a run takes about one second CPU-time. The resulting estimate for the partition sum \(Z\) is obtained by averaging over the one thousand runs. To indicate the statistical error, we also computed the standard deviation among the one thousand runs. The results are presented in Table 1.

Table 1. Estimates of the partition sum \(Z\) for several polymer lengths, with statistical error, the standard deviation between the \(K\) different runs, and the estimated mean end-to-end length \(\sqrt{\langle r_e \rangle}\) with statistical error.

<table>
<thead>
<tr>
<th>(N)</th>
<th>(Z) (\pm) (\Delta Z)</th>
<th>(\sigma(Z))</th>
<th>(\sqrt{\langle r_e \rangle} \pm \Delta r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>(1.1305 \cdot 10^{17}) (\pm) (2.9 \cdot 10^{13})</td>
<td>(9.1 \cdot 10^{14})</td>
<td>(13.881) (\pm) (0.0015)</td>
</tr>
<tr>
<td>100</td>
<td>(7.910 \cdot 10^{42}) (\pm) (2.0 \cdot 10^{40})</td>
<td>(6.2 \cdot 10^{41})</td>
<td>(28.423) (\pm) (0.028)</td>
</tr>
<tr>
<td>150</td>
<td>(1.08 \cdot 10^{64}) (\pm) (1.8 \cdot 10^{62})</td>
<td>(5.8 \cdot 10^{63})</td>
<td>(39.03) (\pm) (0.13)</td>
</tr>
<tr>
<td>200</td>
<td>(1.68 \cdot 10^{85}) (\pm) (1.9 \cdot 10^{84})</td>
<td>(5.9 \cdot 10^{85})</td>
<td>(49.69) (\pm) (0.26)</td>
</tr>
</tbody>
</table>

For longer polymers, the distribution of the estimates for \(Z\) of the individual runs is clearly non-gaussian. Already for \(N = 150\), one run obtains an
estimate for $Z$ which is actually more than ten times the average, and for $N=200$, several runs obtain even more than a hundred times the average! The trend towards non-gaussian behavior is shown in Fig. 2, which shows for the runs with $N=39, 100$ and $150$ a histogram of the deviation from the average of $Z$, measured with respect to the standard deviation in $Z$.

![Fig. 2. Histograms of the measured values of $Z$, shifted around the average value $\langle Z \rangle$, and scaled with the standard deviation $\sigma$. These histograms evolve from gaussian at short lengths, to highly non-gaussian.](image)

For the same number of runs and iterations, but even longer polymers, the computational results become unreliable. The final estimation for $Z$ is dominated by the one single polymer configuration that happens to have a large Rosenbluth weight, and the average is drawn downwards by the majority of iterations resulting in zero Rosenbluth weight. Consequently, the computational results are irreproducible, since these depend on the occurrence of a single dominant iteration.

Up to this point, complete enumeration gives exact results for polymers with a length of a few tens of monomers, and statistical enumeration gives estimates for polymers with a length of a few hundred monomers.
2.3 Pruning and enrichment

At this point, we still cannot handle very long polymers. The reason for this is that during its growth, the Rosenbluth weight of a polymer is multiplied with a varying number, resulting in an exponential divergence in Rosenbluth weights within the population of polymers, as a function of polymer length. As discussed in the previous section, this causes dominance of the one single polymer configuration with the largest Rosenbluth weight, and consequently badly reproducible results.

A first useful observation is that at any given time, we are at liberty to “flip a coin” and either – with a probability $p$ – reduce the Rosenbluth weight to zero and halt the polymer growth, or – with a probability $1 - p$ – continue the polymer growth but with $1/p$ times its Rosenbluth weight. A common choice for $p$ is 50%. Since the set of polymer configurations can naturally be represented as a tree, in which the single-monomer configuration is the root, this operation is known as pruning, since it cuts branches in the tree. Pruning does not bias the results, simply because adding to the cumulative quantity some observable once is statistically equivalent to adding twice this observable with a probability of a half.

The effect of pruning is that it reduces computational effort, at the expense of increased statistical errors. An indiscriminate use of pruning is basically equivalent to simply making fewer iterations, and of no use. However, it is perfectly legitimate to apply pruning only if the Rosenbluth weight is small compared to its average value. In such situations, the reduction in computational effort is still obtained, but the increase in statistical errors is very small since all results including the statistical error are dominated by the (few) polymer configurations with large weight.

Instead of 50% probability to continue with twice the Rosenbluth weight, one can also continue twice with half the Rosenbluth weight. More generally, one can also continue $q$ times after dividing the Rosenbluth weight by $q$. This process is known as enrichment. As with pruning, it does not bias the results.

Enrichment, opposite from pruning, increases the computational effort, while decreasing the statistical errors. The latter desirable effect is mostly obtained if enrichment is applied in case the Rosenbluth weight is large in comparison with its average value. As a result of enrichment, the single dominant polymer configuration that one typically obtains for long polymer lengths is replaced by a large number of polymer configurations.

The combination of pruning polymers with a small Rosenbluth weight and enriching those with a large weight allows for a complete population control, in which all polymer configurations contribute about equally to the final estimates. One should however be cautious: the real aim is not a uniform Rosenbluth weight, but statistical errors as small as possible (obviously, without introducing bias). These two are different, since overly-used population control causes strong correlations between the final polymers, many of which share the first number of steps. Finding a good strategy for pruning
and enrichment is a difficult task. For a more detailed discussion regarding pruning and enrichment, we refer to the work of Peter Grassberger, who is the leading authority in this area at the time of writing.

To illustrate the effectiveness of pruning and enrichment, we applied these techniques to the same calculation as before, and added some computations for longer polymers. As before, we performed simulations for self-avoiding walks on a square lattice with lengths 39, 100, 150 and 200. Additionally, we report simulation results for length 250 and 500. For each length we did 1000 runs, each with $K = 10^5$ iterations. Both pruning and enrichment are not applied during the first 1000 iterations, in which reliable statistics are not yet available. Afterwards, pruning is applied as soon as the Rosenbluth weight (excluding the factors of 2 and 1/2) is smaller than 0.5 times its average; enrichment is applied if this weight exceeds 5 times its average. With this choice, each such a run takes again about one second of CPU-time. The results are presented in Table 2.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$Z$</th>
<th>$\Delta Z$</th>
<th>$r_e$</th>
<th>$\Delta r_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>$1.1312 \times 10^{17}$</td>
<td>$2.5 \times 10^{13}$</td>
<td>13.883</td>
<td>$0.0011$</td>
</tr>
<tr>
<td>100</td>
<td>$7.762 \times 10^{42}$</td>
<td>$4.3 \times 10^{39}$</td>
<td>27.972</td>
<td>$0.005$</td>
</tr>
<tr>
<td>150</td>
<td>$1.036 \times 10^{64}$</td>
<td>$1.5 \times 10^{61}$</td>
<td>37.83</td>
<td>$0.016$</td>
</tr>
<tr>
<td>200</td>
<td>$1.35 \times 10^{85}$</td>
<td>$3.6 \times 10^{83}$</td>
<td>47.25</td>
<td>$0.021$</td>
</tr>
<tr>
<td>250</td>
<td>$1.65 \times 10^{106}$</td>
<td>$3.8 \times 10^{103}$</td>
<td>55.39</td>
<td>$0.018$</td>
</tr>
</tbody>
</table>

2.4 Markovian simulation approach to polymers

A radically different approach to sample polymer configurations is based on the concept of Markov processes. Instead of starting from a single monomer and adding monomers one by one, here the simulation starts with a complete polymer configuration, which is then evolved in time. If $C_i$ denotes the configuration after $i$ steps, the evolutionary process has to be designed such that in the limit of infinitely many steps, the probabilities to be in any allowed configuration are equal. Usually, the so-called Markov chain of configurations is obtained by generating configuration $C_{i+1}$ from its precessor $C_i$ by making a

\[ \text{In the more general case with energetic interactions, the probability to be in configuration } C_i \text{ has to be proportional to its Boltzmann weight } \exp(-E_i/k_B T); \] in the case at hand, their are no energetic interactions and thus all Boltzmann weights are equal.
small change (often called a *move*) in it. A commonly used type of move involves the displacement of a single randomly chosen monomer to a randomly chosen new position. For a specific configuration of a two-dimensional self-avoiding-walk polymer, the possible single-monomer displacements — local moves — are illustrated in Figure 3.

A typical simulation then proceeds as follows. Starting from some initial polymer configuration $C_0$, many moves are made until the polymer is *thermalized*, i.e. it has reached a configuration $C_T$ which is not correlated to the initial configuration $C_0$. Next, many more moves are made until after $M$ moves we have spent the amount of computational effort that we were prepared to invest. At this point, a set of configurations $\{C_T \ldots C_N\}$ is generated that is representative for the set of all possible polymer configurations. An estimate for some observable quantity, for instance the expectation value for the mean squared end-to-end length $r_e^2$, is obtained by averaging the end-to-end lengths of all polymers in the set generated. The best estimate is then

$$\langle r_e^2 \rangle \approx \frac{1}{N-T+1} \sum_{i=T}^{N} r_e^2(C_i).$$

(12)

There are two important issues to be addressed before the Markovian approach sketched here can be used for actual computations. First of all, the algorithm — defined by the choice for the type of moves and the probability to propose each of them — has to obey the condition of *detailed balance*. In the case at hand, in which there are no energetic interactions, the transition probability $T(C_i \rightarrow C_j)$ to move from some configuration $C_i$ to another configuration $C_j$ has to be equal to the probability $T(C_j \rightarrow C_i)$ for the reverse move from $C_j$ to $C_i$. If this condition of detailed balance is met for all pairs of configurations, and if at some time $t$ the Markov process is sampling all polymer configurations with the same probability, then at all times after $t$ also all probabilities for polymer configurations will be equal.
Besides detailed balance, there is another condition that needs to be satisfied in order to obtain a correct Monte Carlo algorithm. This condition — known as **ergodicity** — is the requirement that starting from any configuration $C_0$, it has to be possible to reach any other configuration $C_i$ through a sequence of moves. In the absence of ergodicity, the configurational space contains separate regions, from each of which the polymer cannot escape once it is in one of these regions, even after infinitely many moves.

Thirdly, an important practical consideration is how much computational effort needs to be invested in order to evolve from some configuration to another statistically independent configuration. The fewer effort required, the more efficient the algorithm. The efficiency of an algorithm contains two separate parts: the computational effort per move and the correlation time $\tau$, which is the number of such moves after which a statistically uncorrelated configuration is reached. The first depends on the actual implementation and the hardware used; usually, however, it is possible to discuss how it scales with polymer length. The second part is implementation-independent once the algorithm is specified. We now will proceed with discussing three different algorithms.

### 2.5 Local moves

The first, straightforward algorithm to simulate polymer configurations was already briefly mentioned above. It is a Markovian process in which the moves consist of single-monomer displacements. Here we address the three issues of detailed balance, ergodicity and correlation times for this algorithm.

First, it is important to be precise about the algorithm. One possible choice to obtain configuration $C_{i+1}$ from configuration $C_i$ would be:

- randomly select one of the $N$ monomers, hereafter to be called monomer $s$;
- identify all $n_s$ lattice sites where monomer $s$ can be placed, under the constraints imposed by the other monomers. One of these $n_s$ sites is the site where monomer $s$ was located in configuration $C_i$. If $n_s = 1$, there is no other option but to replace monomer $s$ where it was, and thus $C_{i+1} = C_i$. If $n_s > 1$, randomly choose one of the sites other than the original site.

This algorithm obeys the condition of detailed balance: the transition probability for this algorithm equals $T(C_i \to C_{i+1}) = N^{-1} \cdot \min(1, n_s)^{-1}$, and the reverse transition probability $T(C_{i+1} \to C_i)$ is exactly equal to it.

To illustrate that detailed balance is not automatically guaranteed, consider an alternative algorithm in which all possible single-monomer moves are listed and randomly one of them is selected. Since the total number of such moves is not constant, such an algorithm would violate detailed balance. For instance in the polymer configuration depicted in Figure 3, there are five...
possible moves (corresponding to the five arrows), but the configuration after
displacing monomer 4 would allow six possible moves since the movement of
monomer 6 is not blocked any longer. Thus, the forward transition proba-
bility, equal to $1/5$, differs from its reverse, equal to $1/6$. This serves as a
warning to be cautious.

The second ingredient to prove correctness of a Monte Carlo algorithm
involves ergodicity. The local algorithm outlined above can be shown to vio-
late this condition: configurations exist in which no single-monomer move is
possible. One such configuration is drawn in Figure 4.

![Polymer configuration in which no single-monomer move is possible. The existence of such configurations proves the violation of ergodicity.](image)

The lack of correctness of algorithms using local moves has not prevented
their wide-spread use. For a long time, the justification for nevertheless using
these has been the absence of provably correct algorithms, combined with
the observation that the fraction of phase space unaccessible via local moves
is rather small. If one reduces the phase space by excluding all polymer
configurations that cannot be reached from a completely elongated stating
configuration, than within this reduced phase space, the algorithm is correct
since it obeys detailed balance and ergodicity.

### 2.6 Slithering snake

A more efficient method to sample polymer configurations is known as the
“slithering snake”. In this algorithm, one distinguishes both ends of the poly-
mer: one is called the “head” and the other is the “tail”. The algorithm then
proceeds as follows:

- Remove the monomer at the tail.
- Attempt to add a monomer to the head of the polymer, on one of the
  lattice sites which is a nearest-neighbor of the last monomer, excluding
  the site occupied by the second-last monomer.
• If the attempt violates self-avoidance, return to the original configuration but interchange the head and tail.

If at every step we would randomly decide which end is the head and which the tail, this algorithm would obey detailed balance. As presented here, and in fact as presented in the original paper of Wall and Mandel [5], this algorithm does violate detailed balance: if all nearest-neighbor sites of the end-monomer at the head are occupied but the end-monomer at the tail not, the probability to exchange head and tail is 100%, whereas the reverse probability is zero. As shown by Wall and Mandel, however, the violation of detailed balance does not bias the results.

However, like the previously outlined algorithm with local moves, also the “slithering snake” algorithm violates the condition of ergodicity: in the polymer configuration depicted in Figure 4, no moves are possible. Again, justification of the use of this algorithm is based on the observation that the fraction of phase space unaccessible via local moves is rather small. If the phase space is reduced to those configurations that can be reached from a completely elongated starting configuration, the algorithm is provably correct. Accepting this restriction, the “slithering snake” algorithm is one of the few useful Monte Carlo algorithms which are provably correct even though the condition of detailed balance is violated.

2.7 The pivot algorithm

Until now, the Markov chain of configurations was constructed via a sequence of local deformations of the polymer. As was shown, these algorithms suffer from a serious problem: their lack of ergodicity. We will now discuss a global algorithm that does not suffer from this problem. This global algorithm is known as the pivot algorithm [6], and proceeds as follows:

- in the current polymer configuration \( C_i \), select randomly one monomer, which will serve as a center for rotation or pivot.
- construct a trial configuration \( C'_{i+1} \) by rotating the polymer segment to one side of this pivot with respect to the segment on the other side over ± 90 or 180 degrees.
- if the resulting trial configuration \( C'_{i+1} \) is self-avoiding, accept the new configuration \( (C_{i+1} = C'_{i+1}) \), otherwise continue with the configuration before the rotation \( (C_{i+1} = C_i) \).

The pivot algorithm is illustrated in Figure 5. It clearly obeys the condition of detailed balance. In contrast to the two previous algorithms, it also is ergodic. Allowing also moves with rotations over 180 degrees turns out to be essential for ergodicity. Thus, even if it is tempting to limit the pivot moves to rotations over ±90 degrees since the moves with a rotation over 180 degrees are much less likely to result in a self-avoiding configuration, one should not do this.
Fig. 5. One elementary move of the pivot algorithm: the pivot is chosen to be monomer 6, and the direction 90 degrees counter-clockwise. Since the resulting configuration is self-avoiding, this particular pivot move will be accepted.

3 The project

In this project, you are expected to determine the growth exponent $\nu$ for self-avoiding walks. There are three different, but related, approaches:

3.1 approach I: exact enumeration

1. Write a program to generate all SAWs on a square lattice. I suggest to do this with a recursive program, as discussed above.
2. Determine the average squared end-to-end length $\langle r_e^2(N) \rangle$, for $N = 1, \ldots$, up to the maximal length that you can handle without excessive use of computer resources; this will probably be somewhere between 20 and 30.
3. Make a double-logarithmic plot of $\langle r_e^2(N) \rangle$ versus $N$ and fit a straight line. Remember that we are interested in the slope for long polymers, so the fit does not need to be a good one for small $N$.
4. Since $\langle r_e^2(N) \rangle \sim N^{2\nu}$, one can derive that

$$\frac{N}{\langle r_e^2(N) \rangle} \frac{\partial \langle r_e^2(N) \rangle}{\partial N} \approx \frac{N}{\langle r_e^2(N) \rangle} \frac{\langle r_e^2(N + 1) \rangle - \langle r_e^2(N - 1) \rangle}{2} \sim 2\nu. \quad (13)$$

Plot this estimate of $2\nu$ as a function of $N$.

3.2 approach II: pruned-enriched Rosenbluth method

1. Write a program for the statistical enumeration of SAWs on a square and a cubic lattice.
2. Determine the average squared end-to-end length $\langle r_e^2(N) \rangle$, for various values of $N$ up to about $N = 200$.
3. Make a double-logarithmic plot of $\langle r_e^2(N) \rangle$ versus $N$ and fit a straight line. Remember that we are interested in the slope for long polymers, so the fit does not need to be a good one for small $N$.  

3.3 approach III: Monte Carlo Markov chain

1. Write a program for the simulation of two-dimensional self-avoiding walks, evolving via the local moves as described above.

2. **Thermalization** Start from a completely stretched polymer configuration with \( r_0 = (0, 0) \) and \( r_N = (N, 0) \), with \( N = 50, 100 \) or 150. Next, let the configuration evolve through local moves.
   a) How many local moves does it take before the distance between the ends as projected on the \( x \)-axis (i.e., \( \Delta x = x_N - x_0 \)) becomes negative for the first time, for these lengths?
   b) How many local moves does it take before the number of horizontal steps (in the \( \pm x \)-direction) minus the number of vertical steps becomes zero?

3. Replace the local moves by 'slithering snake'-like moves and repeat the above measurement.

4. **Optional** Repeat the same, but now with pivot moves.

5. For lengths \( N = 10, 20, \ldots, 150 \), use the most efficient algorithm and measure the expectation value of the squared end-to-end length \( \langle r^2_e(N) \rangle \). Make a double-logarithmic plot of \( \langle r^2_e(N) \rangle \) versus \( N \) and fit a straight line. The slope of this straight line gives you an estimate for the growth exponent \( \nu \); which value do you obtain? Remember that we are interested in the slope for long polymers, so the fit does not need to be a good one for small \( N \).

6. **Optional** Study polymers with attractive interactions, for instance between monomers on nearest-neighbor sites, or between parallel steps on opposite sides of the same square; do they become more compact? Do they show other striking features such as winding?

References