very often the growth process stops before the cluster has reached the edge. For $p < p_c$ this occurs very frequently, whereas for $p > p_c$ percolating clusters are generated almost exclusively.

Exercise

Let s be the number of particles in a percolating cluster. In the infinitely large system, the average cluster mass $\langle s \rangle$ of the *finite* clusters diverges near the percolation threshold according to the power law

$$\langle s \rangle \sim |p - p_c|^{-\gamma} \ .$$

Calculate the mean value $\langle s \rangle$ for all numerically generated clusters that do not touch the edge. Plot these values as a function of the concentration p, and attempt to use the result to determine the critical concentration p_c. Calculate the maximum of this function for different values of the lattice size L. Try to calculate the critical exponent γ with the help of *finite size scaling*. In doing this, you may use the value of ν given above.

Literature

Gould H., Tobochnik J. (1996) An Introduction to Computer Simulation Methods: Applications to Physical Systems. Addison-Wesley, Reading, MA

Stauffer D., Aharony A. (1994) Introduction to Percolation Theory. Taylor & Francis, London, Bristol, PA

5.4 Polymer Chains

Polymers play an important role in chemistry as well as in biology and medicine. But physics, too, has long been interested in general mathematical laws concerning the properties of polymers. Even a single molecule which is made up of a long chain of identical units (*monomers*) has interesting properties. In order to be able to mathematically describe such chains of many thousands of monomers, one must attempt to model the essential properties and structures as simply as possible. One proven model for this is the randomly generated path, which is called the *random walk* in scientific literature. The path consists of many short line segments that are joined together in random directions.

In Sect. 3.3 we have already established that a random walk can be regarded as a model of a polymer molecule, all of whose configurations are equally probable in a heat reservoir. In that case, the mass of the chain increases as the square of its mean size. The mathematical theory for this is very well developed. This model neglects an important mechanism, however:

the chain cannot intersect itself. Random walks with such a restriction are appropriately called *self-avoiding walks* (SAWs). Although there are only approximate analytical calculations for SAWs, simulating them on a computer is relatively easy.

Physics

A random walk is most easily defined on a lattice. Since we are only interested in the global properties of very long polymers, in particular in their fractal dimension, we may assume that it does not make a difference if we consider a *random walk* with continuous step sizes and directions or one on a lattice. In Sect. 3.3 we have shown that an unrestricted random walk describes a tangled ball of a polymer molecule whose mass increases as the square of its mean size. For if N is the number of steps and R_N is the distance between the two ends of the walk, then

$$N = \frac{\langle R_N^2 \rangle}{a^2} \,. \tag{5.35}$$

Here $\langle \ldots \rangle$ is an average over all random walks and a is the length of the individual chain links. If we define a mean length by $L = \sqrt{\langle R_N^2 \rangle}$, we obtain

$$N \propto L^D \tag{5.36}$$

with the dimension $D = 2$. This result is valid not only in the plane, but in all spatial dimensions.

Now we consider random walks which may not intersect themselves. In real polymers it is the internal volume of the chains that prohibits this penetration. Obviously, such a self-avoiding walk will not be as bunched up on average as the unrestricted random walk; the mean end-to-end distance L will be larger. This means that the structure of the corresponding polymer will no longer be as compact and the dimension D should be less. Indeed, for large values of N, one finds that the SAW obeys a law of the form (5.36) again, but now with a fractal dimension D which depends on the spatial dimension d. As early as 1949, Flory derived a formula for D, using a kind of mean-field approximation:

$$D = \frac{d+2}{3} \,. \tag{5.37}$$

Subsequent investigations have shown that this formula is exact in $d = 1$, 2, and 4 spatial dimensions and that the numerical results for $d = 3$ yield a value for D which is only slightly higher. In $d = 4$ spatial dimensions, the value $D = 2$ agrees with the one from the random walk, indicating that, in this case, the prohibition of self-intersection is no longer relevant for the global properties. In all higher spatial dimensions the value of D remains 2. As with many other phase transitions, the critical exponents are described by mean-field theory starting with an upper critical dimension $d = 4$.

In fact, there are dilute polymer solutions in which, at high temperatures, an SAW law as in (5.36) and (5.37) has been observed experimentally. In the case of real molecule chains, there can also be attractive interactions between distant monomers of a single polymer molecule, causing the chain to collapse to a compact ball with $D = d$ at low temperatures. Of course, the situation becomes even more complex if many molecule chains interact with each other. Particularly in recent years, extensive numerical simulations have been able to contribute significantly to the understanding of the polymer dynamics of such a "spaghetti soup." Here, though, we want to concern ourselves with the simulation of the relatively simple SAW.

Algorithm

A random walk on the lattice is easily generated with a computer. For each step, one selects one of the adjacent sites via a random number and moves to that site. For a random walk that is not permitted to cross itself, the algorithm seems to be obvious as well: one randomly selects one of the adjacent sites not visited before and makes a step to this site. Surprisingly, though, the algorithm yields incorrect results: on average, the polymer becomes more compact than the correct statistical mean. Chains which touch in several places are generated more often than stretched chains with few contact points.

At this point we want to provide a brief explanation for this phenomenon. Consider a growing chain and let Z_i be the number of sites next to its head that have not been visited yet. Then $1/Z_i$ is the probability that a monomer is added at one of these sites, and the total probability W_N of generating a particular chain configuration with N links is

$$W_N = \prod_{i=1}^{N} \frac{1}{Z_i} \,. \tag{5.38}$$

One can see, therefore, that chains with many contact points (Z_i small) are assigned a high probability. This cannot be, however, since each allowed configuration must occur with the same probability in the correct statistical mean.

But (5.38) immediately points out a way to correct this unwanted preference for compact chains. One has to assign the statistical weight Z_i to each link and weight the properties of the polymer by this amount. This way each chain gets the same statistical weight

$$W_N \cdot \prod_{i=1}^{N} Z_i = 1 \,, \tag{5.39}$$

and consequently the correct value of the mean end-to-end distance L for this simulation is

$$L^2 = \langle R_N^2 \rangle = \frac{\sum_l R_{N,l}^2 \prod_{i=1}^{N} Z_{i,l}}{\sum_l \prod_{i=1}^{N} Z_{i,l}}, \tag{5.40}$$

where l enumerates the configurations generated.

There is, however, an efficient method to directly generate polymer configurations with a constant probability. While the previous algorithm lets chains grow, this method, which is called *reptation*, works with chains of constant length N. In this process the polymer "slithers" across the lattice and thus constantly changes its form and direction.

Before we describe this reptation algorithm and its properties, we want to explain briefly what is to be understood by a configuration of the polymer and when two such configurations on the two-dimensional square lattice are considered to be different. For example, it makes sense to consider all those configurations that result from the possible translations of a given form of the polymer chain as being the same. To compare possibly different configurations, they can then be shifted in such a way that all begin at the same fixed point of the lattice. This already implies that the chain has a beginning and that we can consequently talk about the first monomer unit, the second, the third, etc.

There is one more symmetry of the square lattice that we should use for simplification, the symmetry with respect to 90° rotations. The direction of the first monomer might point to the north, west, south, or east.

If we ignore the direction of the first chain link, then the unique characterization of a configuration only requires the knowledge whether the next step after the first one is to the left, straight ahead, or to the right, and correspondingly for all subsequent steps. Ergo we can describe a configuration of a polymer consisting of N monomers by specifying a sequence of $N-1$ directions whose possible values are *left*, *straight*, or *right*. In this context, it is important that we go through the sequence in a well-defined order. As we have already mentioned above, the polymer chain needs an orientation. To this end, we label its two ends as *head* and *tail* respectively and define, for example, that the orientation is to be specified from the tail end to the head.

But not every sequence of $N-1$ directions represents an SAW; only those sequences that do not exhibit any self-intersections are allowed, a constraint that cannot be formulated locally. This is also the reason why there are so few analytical results concerning the statistics of SAWs up to now and why one has to rely on computer simulations instead.

The algorithm for the reptation of a polymer chain with N elements goes as follows:

1. Start with a configuration on the lattice, labeling the ends of the chain as *head* and *tail* respectively.
2. Remove the last monomer at the tail end and randomly select one of the sites adjacent to the head (three possibilities on the square lattice).

3. If this site is free, add a new head monomer there. If it is occupied, restore the old configuration, interchange the labels *head* and *tail*, and take the otherwise unmodified configuration into account in calculating averages.
4. Iterate steps 2 and 3.

It can be shown that this method generates all configurations that can possibly result from the starting configuration, with the same probability. To do this, we label the different reachable states with $l = 1, 2, 3, \ldots, \mathcal{N}$ and designate the occupancy probabilities that evolve after a suitably long time by $p_l(t)$. If in addition we know the transition probabilities $W(l \to k)$, with which in each time step a configuration l is transformed to another configuration k, we can write down the following equation for the time evolution of the $p_l(t)$:

$$\Delta p_l \equiv p_l(t+1) - p_l(t) = \sum_{k=1}^{\mathcal{N}} [W(k \to l)\, p_k - W(l \to k)\, p_l] \ . \tag{5.41}$$

This is a discretized version of the master equation. Because of the relation $\sum_k W(l \to k) = 1$, the summation of the second term can be executed. Since we are interested in the stationary distribution, i.e., in $\Delta p_l = 0$, we are looking for the solution of the following system of equations:

$$\sum_{k=1}^{\mathcal{N}} W(k \to l)\, p_k = p_l \ . \tag{5.42}$$

For some problems of this kind, the transition probabilities $W(l \to k)$ are constructed with the help of a stationary distribution w_k, in such a way that the equation

$$W(k \to l)\, w_k = W(l \to k)\, w_l \tag{5.43}$$

holds. In this case, one speaks of *detailed balance*, and one can see immediately from (5.41) that this leads to $p_k = w_k$ as a solution of the stationary master equation.

In our case, however, detailed balance is not valid, as is illustrated by the following example: The completely straight configuration has a probability of $1/3$ of having a bend to the right at its head end after the next step, while the probability for this new state to return to the straight one is zero.

Instead of detailed balance, we can use the following relation for the reptation algorithm:

$$W(k \to l) = W(l_{\text{inv}} \to k_{\text{inv}}) \ , \tag{5.44}$$

where we designate by l_{inv} the state one obtains from l by interchanging *head* and *tail*. For $l = k_{\text{inv}}$ (5.44) is fulfilled trivially. This concerns all cases in which the algorithm requires an interchange of head and tail. For the other transitions, the following holds: If it is possible to remove a monomer at the tail end and add one at the head instead, then the inverse process, removing

this new head monomer and putting it back at the tail end, is possible as well. From this we conclude that both sides of (5.44) are either equal to zero or not equal to zero. But if they are not equal to zero, they are both 1/3, as the addition of a new head monomer in a specific direction always happens with a probability 1/3, if it is at all possible. If we insert (5.44) into (5.42) we find, in a similar manner as above, that p_l = constant = $1/\mathcal{N}$ is a solution since, of course, $\sum_k W(l_{\text{inv}} \to k_{\text{inv}}) = 1$ holds as well. This means that all obtainable configurations are generated with the same probability.

There are, however, entire classes of configurations which are unattainable by using the reptation algorithm if one uses the completely straight configuration, or an equivalent one, as the initial state. Figure 5.8 shows examples from two of these classes. We can assume, though, that the number of these configurations is small compared to the number of states in the main class. On the other hand, whether these configurations are relevant or not depends, of course, on the question asked. If we use the reptation algorithm to calculate the average end-to-end distance, we expect a value that is slightly too large because we have not taken the more compact states from the other classes into account.

We want to visualize the slithering polymer on the computer screen. To do so, we program the reptation algorithm on the square lattice in C. First, we declare the variable type `vector`, which defines structures with the spatial coordinates (x, y).

```
typedef struct { float x,y;} vector;
vector direction[3], polymer[NMAX];
```

`polymer[NMAX]` is an array of vectors containing all spatial coordinates of the polymer; for example, `polymer[5].y` specifies the y-coordinate of the fifth monomer. We start by defining the straight configuration for the polymer and then plot it by using the function `circle`:

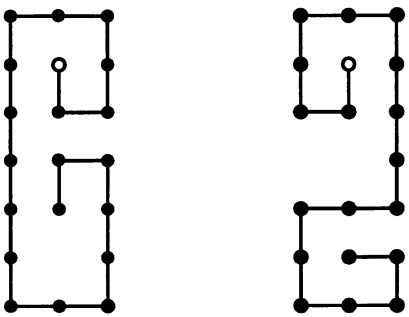

Fig. 5.8. Polymer configurations that are not reachable starting from a completely straight configuration. The open circle designates the head

```
for(i=0;i<N;i++)
 {
  polymer[i].x=i-N/2;
  polymer[i].y=0;
  circle(polymer[i]);
 }
```

Any other equivalent initial configuration is possible just as well. The labels *head* and *tail* are initially assigned to the positions $(N-1)$ and 0 respectively, and for each step these pointers are increased by 1 (modulo N).

The function `choice()` randomly chooses one of the three possible sites adjacent to the head (item 2 above). The function `intersection(c)` checks whether the chosen site is already occupied. If so, head and tail are interchanged, otherwise the new head is accepted and the tail element is overwritten. Thus, item 3 is realized in the program by:

```
while(!done)
  {
    event();
    c=choice();
    if(intersection(c))
          { head=tail; incr=-incr;
            tail=(head+incr+N)%N;
          }
     else accept(c);
  }
```

Rather than moving the polymer around in its memory location we use indices for head and tail. The variable `incr` $= \pm 1$ specifies the direction of the polymer with respect to the array `polymer[N]`. To avoid negative indices, the number N is added before the modulo operation `%`.

As in all our C programs, the function `event()` intercepts keyboard or mouse actions by the user and returns `done=1` if appropriate, which terminates the `while` loop.

The function `choice()` returns an (x, y) vector pointing from the head to one of the three adjacent sites. First the three possible directions are generated by calculating the difference $(r(head) - r(neck))$ – called `direction[0]` – and then determining the two unit vectors perpendicular to this. For example, if the vector `direction[0]` is $(0, 1)$, the variables `direction[1]` and `direction[2]` contain the vectors $(1, 0)$ and $(-1, 0)$ respectively. Finally, using a uniformly distributed random number from the interval $[0, 3)$, truncated to an integer r, one of the three directions is randomly selected and returned:

```
vector choice()
{
  int hm,r;
  hm=(head-incr+N)%N;
  direction[0].x=polymer[head].x-polymer[hm].x;
  direction[0].y=polymer[head].y-polymer[hm].y;
  direction[1].x=direction[0].y;
```

```
  direction[1].y=direction[0].x;
  direction[2].x=-direction[1].x;
  direction[2].y=-direction[1].y;
  r=random(3);
  return direction[r];
}
```

The function `intersection(c)` returns the value 1 or 0 respectively, depending on whether the selected site is already occupied by the chain or not.

```
int intersection(vector c)
{
  int i;
  for(i=0;i<N;i++)
    if(c.x+polymer[head].x==polymer[i].x  &&
       c.y+polymer[head].y==polymer[i].y &&
       i != tail)  return 1;
  return 0;
}
```

If there is no overlap, the selected site is accepted as the new head and the indices are shifted by the value `incr`. On the screen, the circle at the location of the tail is painted over with the background color and a new circle for the head is added. The result is a serpentine motion on the lattice. All this is done by the following function:

```
void accept (vector c)
{
 int hp;
 void shift();
 setcolor(BLACK);
 circle(polymer[tail]);
 hp=(head+incr+N)%N;
 polymer[hp].x=c.x+polymer[head].x;
 polymer[hp].y=c.y+polymer[head].y;
 head=hp;
 tail=(head+incr+N)%N;
 setcolor(WHITE);
 circle(polymer[head]);
 if(abs(polymer[head].x)>2*N ||
    abs(polymer[head].y)>2*N) shift()
}
```

To calculate the average end-to-end distance L, we have to take into account the length of the difference vector $\boldsymbol{r}(\mathit{head}) - \boldsymbol{r}(\mathit{tail})$ after each step, including each reversal of direction, add all lengths, and finally divide by the number of reptation steps. In addition, we have defined a function `shift()`, which shifts the polymer to the center of the lattice whenever it is about to leave the window.

Results

Running our C program generates the polymer on the screen (Fig. 5.9). At the same time, the average end-to-end distance is printed, relative to the result $L = \sqrt{N}$ of the unrestricted random walk. It can be seen that, on average, the polymer is more stretched out than the random walk and that this relative average length increases with N, in agreement with the smaller fractal dimension $D = 4/3$. Indeed, just fitting the two values for $N = 50$ and $N = 100$ shows rather accurately that the ratio of the two lengths increases proportionally to $N^{0.25}$, as predicted by theory.

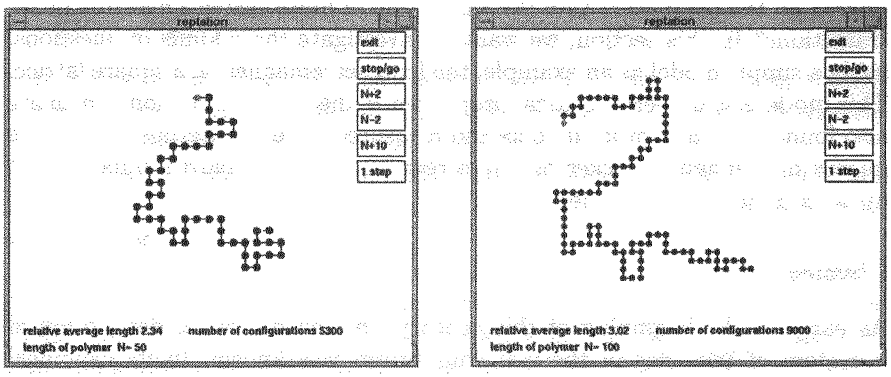

Fig. 5.9. Polymer chains of length $N = 50$ (*left*) and $N = 100$ (*right*) on the square lattice

Exercise

Program the algorithm mentioned at the beginning of the section by correctly taking into account the number Z_i of sites available in each step, as specified in (5.39) and (5.40). Compare this implementation to the reptation algorithm with respect to its speed and to the value of the mean end-to-end distance.

Literature

Binder K., Heermann D.W. (1992) Monte Carlo Simulation in Statistical Physics: An Introduction. Springer, Berlin, Heidelberg, New York

Gould H., Tobochnik J. (1996) An Introduction to Computer Simulation Methods: Applications to Physical Systems. Addison-Wesley, Reading, MA