Bull. Mater. Sci., Vol. 7, No. 1, March 1985, pp. 35–41. © Printed in India.

# Simulation of polymer systems

S L NARASIMHAN, P S GOYAL and B A DASANNACHARYA
Nuclear Physics Division, Bhabha Atomic Research Centre, Trombay, Bombay 400 085, India

**Abstract.** The present paper describes an algorithm which can generate, even on a small computer, arbitrarily long polymer chains, making sure that the configurations generated do not suffer from boundary effects. This has been achieved by employing the concept of a window, which is an analogue of virtual memory scheme. The algorithm has been tested for the case of dilute polymer solution.

**Keywords.** Polymer chains; self avoiding random walks; Monte-Carlo methods.

## 1. Introduction

The configurational properties of polymer chains in polymer solutions or in polymer melts can be studied experimentally using light scattering or neutron scattering techniques. While the light scattering technique is employed for studying dilute polymer solutions, small angle neutron scattering (SANS) technique is employed for studying dilute as well as concentrated polymer solutions, and also polymer melts (Higgins and Stein 1978). The configurational properties of linear polymer chains have also been studied by computer-simulating self avoiding random walks (SAW) on various lattices (Mckenzie 1976) using either the exact enumeration method or the Monte Carlo method.

Configurational properties of short SAWs consisting of about 24 steps or less have been studied by enumerating all possible configurations of the SAW of a given length. But it is not possible to list all the configurations of a long SAW which consists of, say, hundreds of steps. Therefore one has to be content with generating as large a set of configurations as one can, using Monte Carlo simulation technique.

One of the attempts at generating long SAWs using Monte Carlo technique is due to Wall (1954). His method consists in generating SAWs on a finite lattice whose size is determined by the available main memory of the computer used. One usually starts generating a walk from the centre of the lattice. Steps are generated by choosing, at random, one of all the nearest neighbours of the currently occupied site and remembering that backward step is not allowed. Should the site chosen be already occupied, the attempt is discarded and a fresh attempt to generate the chain is made from the centre of the lattice. The effect of this phenomenon, called "attrition", becomes more and more severe as one tries to generate longer and longer walks. It has been reported that out of 140000 attempts made at generating a 121-step walk, only one was successful (Wall 1954). But the successful attempts lead to configurations which are equally probable. To reduce the effect of attrition, Wall has proposed an enrichment scheme (Wall *et al* 1963) with the help of which walks consisting of about 800 steps have been generated.

Rosenbluth and Rosenbluth (1955) independently proposed a method which reduces

the effect of attrition to a large extent but at the cost of biasing the walks. In this method steps are generated by choosing, at random, one of the unoccupied nearest neighbours—not one of all the nearest neighbours as in Wall's method—of the currently occupied site. This way, generation of the walk proceeds until a site which has no unoccupied nearest neighbours is encountered. In such a situation, generation of the walk is terminated, and a fresh attempt is made. As this method generates walk with some bias the configurations generated are weighted so as to make them equally probable.

In either of these methods, attrition is not zero. Moreover, for a given size of the lattice, walks which consist of more than a certain number of steps will suffer from the effects of the boundary (Wall *et al* 1976). To give an example, suppose we have a 16-bit microcomputer which provides a program space of about 20 K words to the user, and we want to generate SAWs on a cubic lattice. Assume that, in this program environment, we are constrained to generate SAWs within a cubic array, LAT, of dimensions $15 \times 15 \times 15$. Clearly, even in principle, we cannot generate as long a SAW as we want to. Also, walks which consist of more than, say 30 or 40 steps, will have finite probability of hitting the boundary of LAT, and therefore will have to be coiled back into the interior of LAT.

In the present paper we report an algorithm which treats LAT as merely a 3-dimensional window through which we can look into the $3d$-real space. Starting from the centre of the window, it generates a segment of the walk, stores the coordinates of the sites visited, and uses the same window after proper initialization, to generate another segment of the same length. This process is repeated till a walk of desired length has been generated. If and when it encounters a walk which enters a blocked state and cannot proceed further, it enforces a back-tracking algorithm so that the walk gets out of its blocked state and proceeds further. This way it can generate as long a walk as is desired even in a restricted program environment, and every attempt made at generating such a walk is successful. The walks generated are weighted using a procedure due to Rosenbluth and Rosenbluth (1955).

The concept of a window is discussed in § 2. Section 3 describes the algorithm and § 4 gives the results we have so far obtained. A discussion of results and the capabilities of the algorithm are given in § 5.

## 2. Concept of a window

Let LAT be a finite cubic array, having dimensions $L \times L \times L$, so that any location can be accessed by specifying the array indices $I, J, K$, where $I, J, K = 1, \ldots, L$. Imagine that LAT is kept in $3d$-real space. Let $X_0, Y_0, Z_0$ be the space coordinates in $3d$-real space of the centre of LAT. If $b$ is the given step length, then the site $(I, J, K)$ of LAT will map into a point with coordinates $(X, Y, Z)$ in $3d$-real space according to the transformation:

$$\begin{aligned} X &= [I-(L+1)/2]b + X_0 \\ Y &= [J-(L+1)/2]b + Y_0 \\ Z &= [K-(L+1)/2]b + Z_0 \end{aligned} \tag{1}$$

LAT can be moved in space in such a way that its centre coincides with the point $(X, Y, Z)$. Then any site $(I', J', K')$ of LAT will map into a point with coordinates $(X', Y', Z')$ according to the transformation.

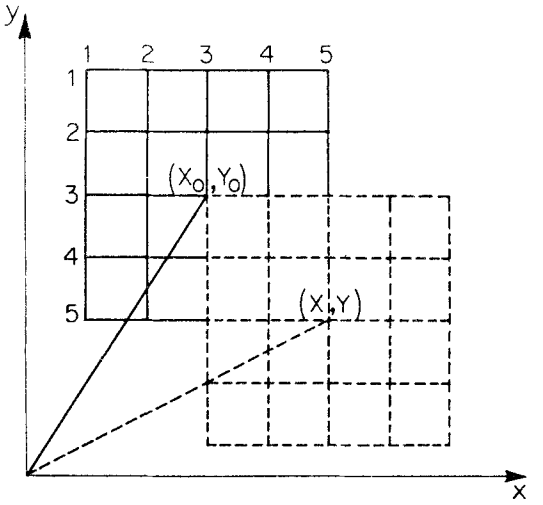


**Figure 1.** Illustration of a two dimensional 5 × 5, moving window.

$$\begin{aligned} X' &= [I' - (L+1)/2]b + X \\ Y' &= [J' - (L+1)/2]b + Y \\ Z' &= [K' - (L+1)/2]b + Z \end{aligned} \tag{2}$$

We will call (1) the "window transformation" and LAT the "window". Figure 1 illustrates a $2d$-window, for $L = 5$.

## 3. Algorithm

Suppose we want to generate a 4095-step SAW and are constrained to work with a minimal data base consisting of the following three arrays: (a) LAT (15, 15, 15)—the window; (b) COORD (512)—an array in which to store the space coordinates of the sites generated and (c) NRAND (512)—array in which to store the number of unoccupied neighbours each site has when the walk proceeds from there. Assume that our machine configuration includes a floppy subsystem.

(i) Choose some point $(X_0, Y_0, Z_0)$ in space to start the walk from; start with a clean window, *i.e.*, initialize array LAT with zeros. Assume that in the window site with coordinates ($I = 8$, $J = 8$, $K = 8$), the centre of the window, maps on to $(X_0, Y_0, Z_0)$. Let $(I, J, K)$ be the window indices of the current site in LAT. Occupy the site $(X_0, Y_0, Z_0)$ by entering data 1 in location $(I, J, K)$ of LAT. Store the coordinates $(X_0, Y_0, Z_0)$ in array COORD. Let $p$ be the number of unoccupied neighbours of $(I, J, K)$. Store $p$ in array NRAND. Choose one of the unoccupied neighbours of $(I, J, K)$ at random and occupy it. Using the window-transformation (1) calculate the space coordinates of the point into which the chosen site maps into. Store these coordinates in COORD and so on. Repeat this process until a walk segment consisting of 6 steps is generated. We restrict ourselves to generating a walk segment consisting of 6 steps in one window because a segment consisting of more than 6 steps has a non-zero

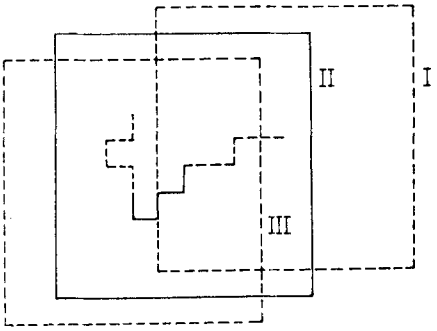


**Figure 2.** Schematic representation of chain generation process in two dimensions. Three segments of the chain generated using windows I, II and III are shown by dashed, solid and dashed lines respectively.

probability of hitting the boundary of LAT and therefore would have to be artificially coiled into LAT. Thus we eliminate boundary effects.

(ii) Define the next window, which is first wiped clean, such that its centre maps into the last point of the previous segment generated, as shown in figure 2 for the case of two dimensional walk. Initialize the window with respect to the previous segment(s) generated. This is done by mapping the coordinates stored in COORD back into the window-indices and checking if they are within the bounds of LAT. If they are, then those locations of LAT are initialized with data 1. Otherwise, they are ignored.

One more segment is generated in this window.

(iii) Process (ii) is repeated till the arrays COORD and NRAND are full. Once they are full, they are dumped into a floppy file, and are initialized for fresh use.

(iv) Process (iii) is repeated till a walk of desired length has been generated.

(v) If in the process of generating this walk, we encounter a situation where the walk cannot proceed further because it has reached a site which has no unoccupied neighbours, then we back-track on the walk, keeping track of which window we are in, to a point from where the walk could proceed along a different path. This is done by scanning the array NRAND from the current location towards the top till we encounter a location which contains $p \geqslant 2$. We determine the window in which this site was generated. Proceed the walk from that site making sure that we avoid those locations of this window which mapped onto points which got the walk blocked. It may be noted that window initialization and some time back-tracking also may require that we retreive the coordinates and values from the floppy files back into the arrays COORD and NRAND. Our algorithm can take care of these data movements back and forth between the buffers COORD and NRAND and their corresponding files on floppy diskette.

Once a walk is generated, it is weighted according to the scheme of Rosenbluth and Rosenbluth (1955) in the following way:

Let the walk consist of $N$ steps. Let $p(i)$ be the number of options available for generating the $i$th link. ($2 \leqslant i \leqslant N$; $0 \leqslant p(i) \leqslant 5$ for a cubic walk). These values are retreived from the array NRAND. Then the weight associated with the walk is given by:

$$W = \prod_{i=2}^{N} \frac{p(i)}{5}. \tag{3}$$

The algorithm also calculates $R^2$, the square of end-to-end distance of the walk. To get an average value $\langle R^2 \rangle$ of $R^2$, we generate a sample consisting of large number, say $n$ of such walks.

## 4. Results

The above algorithm has been used to calculate* the mean square end-to-end distance $\langle R^2 \rangle$ as a function of the number, $N$, of monomers in the chain for a cubic lattice**. This calculation essentially gives information about the dependence of $\langle R^2 \rangle$ on $N$ in a dilute polymer solution. Calculations have been done for $N = 8, 16, 32, 48, 64, 80, 128$. The window used in these calculations had a size of $31 \times 31 \times 31$. For a given position of the window, only a small segment (16 monomers) of the walk was generated to ensure that the boundaries of the window are not encountered in the process of generating the walks. The mean value $\langle R^2 \rangle$ was evaluated using a large number, $n$, of walks by either giving equal weightage to all the walks or by weighting them as per the procedure outlined in earlier section.

To get an accurate value of $\langle R^2 \rangle$, it is desirable to have as large a sample as possible. Restriction to sample size comes from the computation time. The optimum value of $n$ was arrived at by calculating $\langle R^2 \rangle$ as a function of $n$. Figure 3 shows $\langle R^2 \rangle$ as function $n$ for a 128 monomers chain for the case when all the chains were given equal weightage. It may be noted that though the value of $\langle R^2 \rangle$ fluctuates for small values of $n$, it stabilizes to within 2% for $n = 750$. For smaller walks ($N < 128$), the accuracy on $\langle R^2 \rangle$ is expected to be better than 2% for $n = 750$. In view of this, when all the walks had same weightage we used sets of 750 walks to compute the mean value of $R^2$.

In the case of weighted chains, we have used larger samples as it was seen that some walks acquired very low weightage in the process of weighting. The sample sizes used in various calculations are given in table 1. Also given in the same table are the calculated values of $\langle R^2 \rangle$ for different values of $N$ both for weighted and unweighted situations. The dependence of $\langle R^2 \rangle$ on $N$ is shown in figure 4, where $\langle R^2 \rangle$ is plotted against $N$

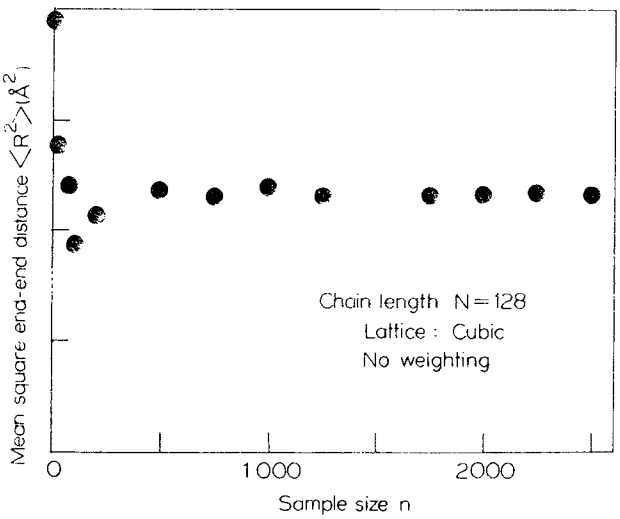


**Figure 3.** Variation of mean square end-end distance, $\langle R^2 \rangle$, as a function of the sample size $n$.

* Calculations performed on the super-mini computer, PRIME 450.

** Dependence of $\langle R^2 \rangle$ on $N$ is known (Domb 1963) to be independent of the lattice used.

**Table 1.** Calculated values of mean square end-to-end distance $\langle R^2 \rangle$ for a polymer chain for different values of $N$.

| Chain length ($N$) | Unweighted | | Weighted | |
|---|---|---|---|---|
| | Sample size ($n$) | $\langle R^2 \rangle$ | Sample size ($n$) | $\langle R^2 \rangle$ |
| 8 | | | 9999 | 10.93 |
| 16 | 750 | 24.14 | 8000 | 27.26 |
| 32 | 750 | 51.74 | 8000 | 66.97 |
| 48 | 750 | 80.68 | 5000 | 101.91 |
| 64 | 750 | 104.08 | 5000 | 129.73 |
| 80 | 750 | 131.91 | 6000 | 167.83 |
| 96 | 750 | 161.64 | | |
| 112 | 750 | 188.45 | | |
| 128 | 750 | 215.23 | 1000 | 333.10 |

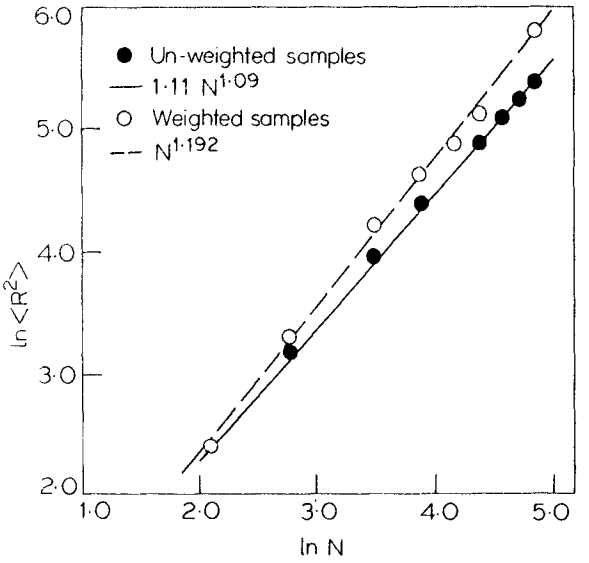


**Figure 4.** Variation of mean square end to end distance, $\langle R^2 \rangle$, for a single polymer chain as a function of number of monomers $N$. Filled circles correspond to unweighted average and open circles correspond to weighted average as discussed in the text.

on a log-log scale. It may be mentioned that for real polymer systems, to get an appropriate value of $\langle R^2 \rangle$ using Monte Carlo method, one has to make sure that the walks used in computing $\langle R^2 \rangle$ are generated with equal probability. In view of this, the open circles represent a more relevant situation.

## 5. Discussion

It is well established both experimentally and theoretically (De Gennes 1979) that the mean square end to end distance $\langle R^2 \rangle$ for a linear polymer chain having $N$ monomers is given by an expression of the type:

$$\langle R^2 \rangle = AN^{\nu}$$

where the exponent $\nu$ depends on the environment in which polymer chain resides. For example, the value of $\nu$ is 1·2 for dilute polymer solutions (De Gennes 1979). Our calculations, where we have generated an isolated polymer chain in an infinitely large vessel, essentially corresponds to dilute polymer solution. It is seen that calculated (open circles in figure 4) values of $\langle R^2 \rangle$ lie on the dashed line $\langle R^2 \rangle = N^{1\cdot192}$. That is, our algorithm gives $\nu = 1\cdot192$ in agreement with earlier simulation studies (Rosenbluth and Rosenbluth 1955) as well as light and neutron scattering experiments on dilute polymer solutions (Flory 1953; Cotton *et al* 1974). We also note that the solid points, corresponding to unweighted samples, give $\nu = 1\cdot09$ again in agreement with earlier simulation work (Rosenbluth and Rosenbluth 1955).

In short, we find that the above algorithm for examining the chain configurations in polymer systems using Monte Carlo method gives results similar to those obtained using existing algorithms. Using the present algorithm, however, it is possible to examine very long polymer chains ($N \geqslant 1000$) even on a small computer without encountering boundary effects, which is not possible with the other existing algorithms. Moreover, as this algorithm keeps track of all the occupied neighbours of each monomer of the chain, unlike earlier algorithm, it can be easily used for examining concentrated polymer solutions and polymer melts. In particular, using the above algorithm we are examining the configurations of a long ($N$ monomer) polymer chain in a melt of small ($P$ monomer) chains (Joanny *et al* 1981).

## References

Cotton J P, Decker D, Benolt H, Farnoux B, Higgins J S, Jannik G, Ober R, Picot C and des Coiseaux J 1974 *J. Macromol.* 7 863

De Gennes P G 1979 *Scaling concepts in polymer physics* (Ithaca, New York: Cornell University press)

Domb C 1963 *J. Chem. Phys.* **38** 2957

Flory P J 1953 *Principles of polymer chemistry* (Ithaca, New York: Cornell University press)

Higgins J S and Stein R S 1978 *J. Appl. Crystallogr.* **11** 346

Joanny J H, Grant P, Turkevich L A and Pincus P 1981 *J. Phys.* **42** 1045

Mckenzie D S 1976 *Phys. Rep.* **C27** 35

Rosenbluth M N and Rosenbluth A W 1955 *J. Chem. Phys.* **23** 356

Wall F T 1954 *J. Chem. Phys.* **22** 1036

Wall F T, Windwer S and Gans P J 1963 *Methods in computational physics* (New York: Academic Press) vol. 1 pp. 217

Wall F T, Mandel F and Chin J C 1976 *J. Chem. Phys.* **63** 4592