MOLECULAR PHYSICS, 10 JUNE 2003, VOL. 101, NO. 11, 1675–1682

Taylor & Francis
Taylor & Francis Group

Dynamic pruned-enriched Rosenbluth method

NICOLAS COMBE[1], THIJS J. H. VLUGT[2],
PIETER REIN TEN WOLDE[1] and DAAN FRENKEL[1]*

[1]FOM Institute for Atomic and Molecular Physics, Kruislaan,
407, 1098 SJ Amsterdam, The Netherlands
[2]Physical Chemistry of Interfaces, P.O. Box 80.051,
3508 TB Utrecht, The Netherlands

(*Received 17 October 2002; revised version accepted 21 January 2003*)

Recently, Grassberger [1997, *Phys. Rev.* E, **56**, 3682] has presented a new algorithm ('PERM') for simulating flexible polymer chains. This algorithm has been shown to have a good efficiency and has been used in a wide class of systems. A drawback of this algorithm is that it is static: it is therefore not suited for Markov-chain Monte Carlo simulations. Here, we present a dynamic generalization of the PERM algorithm. For a specific example, we compare the efficiency of DPERM to that of other Monte Carlo algorithms. In the case studied, we find that DPERM is only marginally more efficient. However, this result may depend on the details of the implementation.

1. Introduction

Numerical simulations of polymers are almost as old as computer simulation itself. However, whereas the Monte Carlo algorithm for simulating simple atomic and molecular systems has not changed since its very inception [1], polymer simulations remain a technical challenge driving the development of novel Monte Carlo algorithms. The main problem with polymer simulations is that it is not easy to devise an algorithm that will efficiently sample the space of possible polymer conformations. The problem is that the number of possible conformations is astronomically large. For instance, for polymers living on a simple cubic lattice, the number [2] of allowed conformations scales approximately as 4.7^n, where n is the number of monomers in the chain. It would clearly be desirable to achieve large conformational changes in a single Monte Carlo move. However, for most existing MC schemes, the obtained conformations are not very relevant for the calculations of average quantities, in particular for long polymers (see e.g. [3]).

A very early Monte Carlo scheme to generate polymer conformations is the one due to Rosenbluth and Rosenbluth (RR) [4]. In the RR method, the sampling of polymer conformations is biased in order to improve the efficiency of the algorithm. The bias is corrected for by introducing a conformation-dependent weight factor such that the weighted average over all polymer conformations will converge towards the correct Boltzmann average. While the Rosenbluth method is much more efficient than an algorithm that would generate polymer conformations at random, it still becomes inefficient when applied to long chains [5]. Grassberger [6] has suggested adding two ingredients to the RR algorithm to improve its efficiency: 'pruning' and 'enrichment'. The basic rationale behind pruning is that it is not useful to spend much computer time on the generation of a conformation that will hardly contribute to the weighted average. Therefore, it is advantageous to discard ('prune') such irrelevant conformations at an early stage. The idea behind enrichment is to make multiple copies of partially grown chains that have a large statistical weight [6, 7] and to continue growing these potentially relevant chains. The algorithm that combines these two features is called the pruned-enriched Rosenbluth method (PERM). The examples presented by Grassberger and co-workers [8, 9] indicate that the PERM approach can be very useful for estimating the thermal equilibrium properties of long polymers. Moreover, the method can be used to search for the lowest-energy conformation ('native state') of simple lattice proteins.

The main limitation of both the RR method and the PERM algorithm is that they are 'static' Monte Carlo schemes. In a static scheme, a large number of configurations are created independently from each other: one could understand it as picking points in phase space independently from each other (see figure 1 (*a*)). In contrast, in a 'dynamic' scheme, the system performs a walk through phase space. At each step, a new point in phase space is chosen and this trial move is then accepted or rejected depending on the weight of *both* the

* Author for correspondence. e-mail: frenkel@amolf.nl

Molecular Physics ISSN 0026–8976 print/ISSN 1362–3028 online
http://www.tandf.co.uk/journals
DOI: 10.1080/0026897031000094461

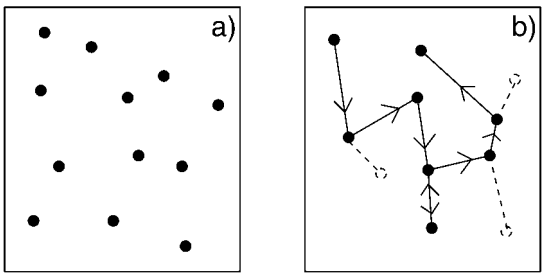


Figure 1. Schematic illustration in space phase of (*a*) a static Monte Carlo method and (*b*) a dynamic Monte Carlo method. In a static scheme, the points in phase space are picked independently from each other, while in a dynamic scheme, one performs an importance-weighted random walk (see text). The solid line denotes the walk between the accepted points, while the dashed line denotes the rejected trial moves.

new and the old configuration (see figure 1 (*b*)). In essence, this method is a Markov-chain Monte Carlo scheme.

The static scheme can simulate single polymer chains very efficiently, but it does become problematic when studying systems consisting of many polymer chains: at each step, one would have to simultaneously generate the conformations of all the chains in the system. On the other hand, in a dynamic scheme, one can conveniently choose a new point in phase space by only changing one chain at each step of the algorithm. An additional advantage is that such a dynamic scheme can be easily incorporated for other ensembles like the Gibbs ensemble or grand-canonical ensemble, giving the opportunity to compute phase equilibria efficiently.

As the RR method and the PERM method are examples of a static scheme, they cannot handle many chains efficiently. However, in the case of the RR method there exists a dynamic generalization: the configurational-bias Monte Carlo (CBMC) [3]. In the CBMC algorithm, the (Rosenbluth) weight of the individual chains determines the probability of accepting or rejecting a new polymer conformation generated by the RR method.

The impressive results that Grassberger and co-workers have reported for the static PERM algorithm, inspired us to generalize the method to a *dynamic* MC scheme (DPERM). The purpose of this paper is to transform the original PERM algorithm into a dynamical scheme (DPERM) and to investigate whether the concept of PERM can improve existing dynamic algorithms such as CBMC. In § 2, we describe the algorithm. In § 3, we apply it to a simple toy model of proteins due to Lau and Dill [10]. In § 4, we discuss ways to select optimal values for the free parameters in the algorithm. However, we find that, for the examples that we studied, DPERM does not significantly outperform existing methods such as CBMC. While this is, of course, disappointing, the flexibility of DPERM makes it likely that there will be cases where it will be the method of choice.

2. Algorithm

Both PERM and DPERM algorithms are based on the Rosenbluth scheme to generate the chains. We therefore briefly recall the essential steps of that scheme. For convenience, we only present the algorithm for lattice models. A description of the off-lattice case can be found in [3].

In the RR algorithm, a conformation of a chain of length l is constructed as follows.

(1) The first monomer is inserted at a random lattice position (n). We define its Rosenbluth weight as $w_1 = \exp(-\beta u^{(1)}(n))$, where $u^{(1)}(n)$ is the energy of the monomer.

(2) For subsequent segments, we consider all possible orientations of the next segment. The energy of the jth trial position of the ith monomer of the chain is denoted by $u^{(i)}(j)$. We select one of these positions with a probability:

$$p^{(i)}(n) = \frac{\exp\left[-\beta u^{(i)}(n)\right]}{w_i}, \tag{1}$$

where $w_i = \sum_{j=1}^{\alpha} \exp[-\beta u^{(i)}(j)]$ and α is the number of trial positions. The energy $u^{(i)}(j)$ takes into account only the interactions of monomer i with previous monomers in the chain so that the total energy of the chain is: $U(n) = \sum_{i=1}^{l} u^{(i)}(n)$.

(3) Step 2 is repeated until the end of the chain is reached.

The Rosenbluth weight of the chain is defined as: $W(\text{chain}) = \prod_{i=1}^{l} w_i$.

This algorithm generates a given chain with a probability $\exp(-\beta U(n))/W(\text{chain})$, where $U(n)$ is the energy of the chain. One can then calculate the thermodynamic average $\langle A \rangle$ (following the Boltzmann distribution) of an observable A by

$$\langle A \rangle = \frac{\sum_{\text{chain}} A * W(\text{chain})}{\sum_{\text{chain}} W(\text{chain})}. \tag{2}$$

The PERM algorithm uses the same algorithm to generate the chains except that now pruning and enrichment are added. These ingredients are implemented as follows. At any step of the creation of a chain, if the partial Rosenbluth weight $W(j) = \prod_{i=1}^{j} w_i$ of a configuration is below a lower threshold $W^{<}(j)$, there is a probability of $\frac{1}{2}$ to terminate the generation of this

conformation. If the conformation survives this pruning step, its Rosenbluth weight is doubled $W^*(j) = 2 * W(j)$. Enrichment occurs when the partial Rosenbluth weight of a conformation $W(j) = \prod_{i=1}^{j} w_i$ exceeds an upper threshold $W^>(j)$. In that case, k copies of the partial chain are generated, each with a weight $W^*(j) = W(j)/k$. All these copies subsequently grow independently (subject to further pruning and enrichment). This procedure is then repeated many times. Average properties can be computed using the re-weighted Rosenbluth weight $W^*(j)$ of all chains that were grown to completion:

$$\langle A \rangle = \frac{\sum_j A * W^*(j)}{\sum_j W^*(j)}. \tag{3}$$

The value of the average is independent of the choice of the upper and lower thresholds if the sum is infinite. However, the rate of convergence of the average could strongly depend on these thresholds.

The DPERM algorithm is the dynamic generalization of the PERM algorithm. As in the CBMC algorithm, we bias the acceptance of trial conformations to recover a correct Boltzmann sampling of chain conformations.

Thus, starting from an old configuration, we create a trial conformation and calculate the probability to generate it. Starting from the condition for detailed balance, we then derive the expression for the probability to accept or reject a new trial conformation. The generation of the chains as such follows the PERM scheme. Below, we describe the method for lattice polymers. However, as in the case for CBMC [3], all steps generalize directly to off-lattice polymers

As we use the Rosenbluth method to generate chains, the probability to grow a particular conformation is

$$P_{\text{gen}}(\text{chain}) = \prod_{i=1}^{l} \frac{\exp\left[-\beta u^{(i)}(n)\right]}{w_i}. \tag{4}$$

In addition, every time the re-weighted Rosenbluth partial weight $W^*(j)$ of the chain drops below the lower threshold $W^<(j)$, the chain has a probability 1/2 of being deleted.† Let us assume that this happens m times. Then, the total probability to generate a *particular* conformation is

$$P_{\text{gen}}(\text{chain}) = \frac{1}{2^m} \prod_{i=1}^{l} \frac{\exp[-\beta u^{(i)}(n)]}{w_i} \tag{5}$$

† We have chosen a probability of 0.5 to prune, but one can modify easily this value.

and the re-weighted Rosenbluth weight of such a chain would be:

$$W^*(\text{chain\_new}) = 2^m * W(\text{chain\_new}) \tag{6}$$

$$\text{with} \quad W(\text{chain\_new}) = \prod_{i=1}^{l} w_i. \tag{7}$$

Whenever the Rosenbluth partial weight exceeds the upper threshold, k copies of the chain are created with the Rosenbluth weight $W^*(j) = W(j)/k$, which leads to the creation of a set of chains: this is a deterministic procedure. At every stage during the growth of the chain, others chains will branch off. The probability to grow the entire family of chains that is generated in one DPERM move can be written as

$$P_{\text{gen}}(\text{chain\_new}) * P_{\text{gen}}(\text{rest\_new}), \tag{8}$$

where $P_{\text{gen}}(\text{rest\_new})$ describes the product of the probabilities involved in generating all the other pieces of chains that branch off from the main chain. If we now call p the number of times the Rosenbluth weight exceeds the upper threshold during the generation of the *given* trial configuration, the probability to generate this *particular* chain is

$$P_{\text{gen}}(\text{chain\_new}) = k^p \prod_{i=1}^{l} \frac{\exp\left[-\beta u^{(i)}(n)\right]}{w_i} \tag{9}$$

and its re-weighted Rosenbluth weight is

$$W^*(\text{chain\_new}) = \frac{1}{k^p} W(\text{chain\_new}). \tag{10}$$

Here k is the number of copies that are created each time the Rosenbluth weight exceeds the upper threshold. In equation (9), the first term of the right-hand side describes the usual probability to generate a given chain following the Rosenbluth method. The factor k^p comes from the fact that the new chain could be any of the chains in the set so that the probability to generate a given chain is multiplied by this term. We also deduce equation (10) from the fact that, each time we make some copies, the Rosenbluth weight is divided by k.

If we now also take into account the possibility that the chain can be pruned, then equation (9) becomes

$$\begin{aligned} P_{\text{gen}}(\text{chain\_new}) &= \frac{k^p}{2^m} \prod_{i=1}^{l} \frac{\exp[-\beta u^{(i)}(n)]}{w_i} \\ &= \frac{k^p}{2^m} \frac{\exp[-U(n)]}{W(\text{chain\_new})}. \end{aligned} \tag{11}$$

Thus equation (10) becomes

$$W^*(\text{chain\_new}) = \frac{2^m}{k^p} W(\text{chain\_new}). \tag{12}$$

Note that equation (11) and equation (12) respectively reduce to equation (5) and equation (6) in the absence of enrichment ($p = 0$) and to equation (9) and equation (10) in the absence of pruning ($m = 0$).

We now choose to select the new trial chain from the set of chains created by the DPERM move with a probability given by

$$P_{\text{choose\_new}} = W^*(\text{chain\_new})/W_{\text{total}}(\text{new}), \tag{13}$$

where $W^*(\text{chain\_new})$ is the re-weighted Rosenbluth weight mentioned in equation (12) and W_{total} is the sum of all such weights

$$W_{\text{total}}(\text{new}) = \sum_{\text{set}} W^*_{\text{chain}}. \tag{14}$$

Equation (13) implies that we are most likely to choose the best chain (the one with the largest re-weighted Rosenbluth weight) of the set as the next Monte Carlo trial conformation.

Assuming that we start from an old configuration denoted by the subscript 'old', we generate a new configuration following the scheme described above and we accept this move with the following acceptance rule:

$$\text{acc}(\text{old} \to \text{new}) = \min\left(1, \frac{W_{\text{total}}(\text{new})}{W_{\text{total}}(\text{old})}\right). \tag{15}$$

To calculate $W_{\text{total}}(\text{old})$, one has to 'retrace' the old chain: the chain is first clear and reconstructed following the procedure described above to determine its weight. This is exactly analogous to what is done in the configurational-bias Monte Carlo scheme.

A proof of this scheme is obtained using the super-detailed balance condition [3]: detailed balance is obeyed if the probability flow $\kappa([\text{old}, \text{set}_{\text{old}}] \to [\text{new}, \text{set}_{\text{new}}])$ from the old configuration *with its set of chains* to new one *with its set* is equal to the reverse flow $\kappa([\text{new}, \text{set}_{\text{new}}] \to [\text{old}, \text{set}_{\text{old}}])$. One can calculate both flows:

$$\begin{aligned}\kappa([\text{old}, \text{set}_{\text{old}}] \to [\text{new}, \text{set}_{\text{new}}]) = {} & \\ & P_{\text{Boltzmann}}(\text{old}) * P_{\text{gen}}(\text{chain\_new}) \\ & * P_{\text{gen}}(\text{rest\_new}) * P_{\text{gen}}(\text{rest\_old}) \\ & * P_{\text{choose\_new}} * \text{acc}(\text{old} \to \text{new}),\end{aligned} \tag{16}$$

where $P_{\text{Boltzmann}}(\text{old})$ is the probability to find the system in the *old* state following the Boltzmann distribution.

Writing the super-detailed balance condition:

$$\begin{aligned}\kappa([\text{old}, \text{set}_{\text{old}}] \to [\text{new}, \text{set}_{\text{new}}]) = {} & \\ & \kappa([\text{new}, \text{set}_{\text{new}}] \to [\text{old}, \text{set}_{\text{old}}]).\end{aligned} \tag{17}$$

$P_{\text{gen}}(\text{rest\_new})$ and $P_{\text{gen}}(\text{rest\_old})$ appear on both sides of equation (17), and therefore these terms drop. Using equations (11) and (13), one can then deduce:

$$\frac{\text{acc}(\text{old} \to \text{new})}{\text{acc}(\text{new} \to \text{old})} = \frac{W_{\text{total}}(\text{new})}{W_{\text{total}}(\text{old})}, \tag{18}$$

which is fulfilled by our criteria of equation (15).

The algorithm described above allows us to perform dynamic Monte Carlo simulations using both enrichment and pruning. In the original static PERM algorithm of Grassberger, the upper and lower thresholds, as well as the number of copies, could be adjusted on the fly. This flexibility was useful to avoid the generation of very large sets of chains, especially at low temperature where the variations in partial weights are huge. As DPERM is a Markov-chain Monte Carlo algorithm, changing the threshold on the fly would break detailed balance. Hence, the number of copies, the upper and lower thresholds should all be fixed before the beginning of the simulation. To select suitable values for these parameters, it is useful to perform a short CBMC simulation before performing the DPERM simulation. In § 4 we discuss how best to choose these parameters.

In the next part, we apply the DPERM method to the simulation of a toy model of proteins. The main aim of this example is that the DPERM method reproduces the results obtained by other dynamic MC schemes, such as CBMC.

3. Toy model of proteins

We apply our algorithm to a toy model of proteins, namely the HP model [10–13]. The folding of proteins is believed to be essentially due to hydrophobic interactions. In the HP model, proteins are modelled as a linear chain of n amino acids. Each amino acid can be of two types: hydrophobic (H) or polar (P). A conformation is represented by a self-avoiding walk on a three-dimensional cubic lattice. Moreover, hydrophobic amino acids that are neighbours on the lattice, but not adjacent along the sequence attract each other with a binding energy $\varepsilon_{\text{HH}} = \varepsilon < 0$. We assume there is no interaction between any other couple of amino acids: $\varepsilon_{\text{HP}} = 0$ and $\varepsilon_{\text{PP}} = 0$. The properties of this model system only depend on the dimensionless parameter $\varepsilon^* = \varepsilon/k_{\text{b}}T$ where k_{b} is the Boltzmann constant and T the temperature. We use a chain made of 48 amino acids and use the following sequence:

$$\text{H}_6\text{PH}_2\text{P}_6\text{HP}_2\text{H}_4\text{PHPH}_5\text{P}_4\text{HP}_3\text{H}_2\text{P}_6\text{H}_2.$$

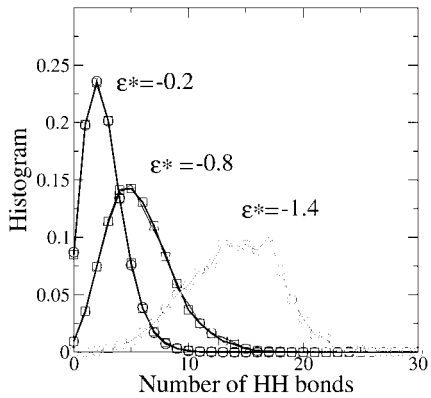


Figure 2. Histogram of the HH bonds for the HP model calculated with both DPERM (square) and CBMC (circle) algorithms for different values of ε^*. For $\varepsilon^* = -0.2$, the pruning rate is about 10% and the enriching rate about 7%. For $\varepsilon^* = -0.9$, the pruning rate is about 17% and the enriching rate about 7%. For $\varepsilon^* = -1.4$, the pruning rate is about 0.2% and the enriching rate about 5%.

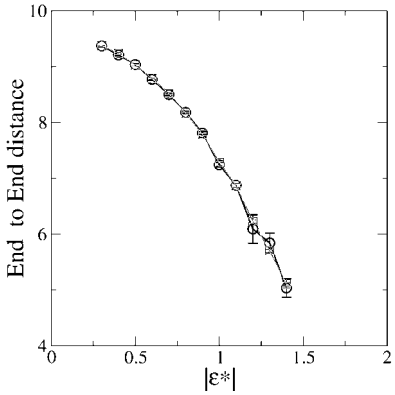


Figure 3. Average end-to-end distance calculated as a function of ε^* for both CBMC (circle) and PERM (square) algorithms. The error bars are mentioned but are smaller than the symbols at high temperature.

For a small value of $|\varepsilon^*|$, only a few HH bonds are found and the chain is in a coil state. When increasing the value of $|\varepsilon^*|$, the average number of HH bonds increases and we observe a globular state. For a given value of ε^*, one can calculate the probability of having a given number of HH bonds. To check the results of the DPERM algorithm, we compare the histogram of the HH bonds obtained with the DPERM algorithm with the one generated using a CBMC algorithm, see figure 2. We have checked that histograms do not depend on the choice of the lower or upper thresholds—however, the acceptance rate does.

Figure 2 illustrates that the results obtained with the DPERM algorithm are, apart from statistical errors, identical to those obtained using the CBMC algorithm. We have not studied this model at even lower temperatures because it is becoming increasingly difficult to obtain good statistics. Moreover, the fluctuations in the Rosenbluth weight become so large that the size of the sets created during successive Monte Carlo steps also fluctuates wildly—in some cases, we ended up with sets containing more than a thousand chains. We have also calculated the average end-to-end distance of the chain as a function of ε^*: figure 3 shows these results.

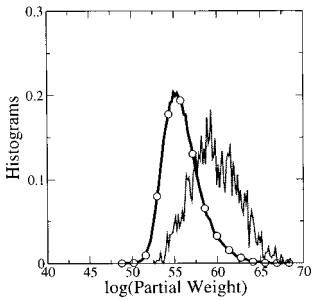
Both figures 2 and 3 show that the DPERM algorithm yields the same results as CBMC. In the next section, we discuss how to optimize the efficiency of the algorithm.

4. Choice of the thresholds and efficiency

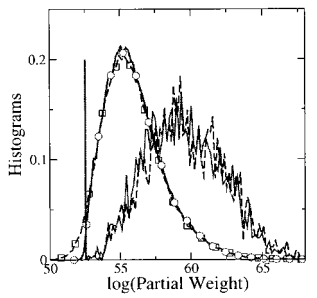
The choice of the different thresholds strongly affects the efficiency of the algorithm. Indeed, if the lower thresholds are too high, then almost every created chain will be pruned, and much CPU time will be wasted on the generation of chains that do not survive anyway. On the other hand, if the upper thresholds are too low, most chains that are generated will be enriched and the average size of the set of chains created at each Monte Carlo move will become far too large, again wasting a lot of CPU time.

In the description of the PERM algorithm, Grassberger pointed out that it is desirable to have pruning and enrichment more or less in balance. For the DPERM algorithm, this implies that it would be optimal if, on average, a DPERM trial move generates a single chain.

Pruning is useful if only those chains that are deleted would anyway have stood little chance of being accepted as a trial conformation. A good way to determine the pruning threshold is to perform a short CBMC simulation in which we construct a histogram of the partial weight of all chains that have been grown to a given length m. Separately, we can collect a histogram of the partial Rosenbluth weights at length m of only those chains that were accepted at the end of the trial move. The pruning threshold should be chosen such that potentially successful chains will not be deleted. Figure 4 (*a*) shows an example of these histograms for $m = 30$. These histograms were obtained from CBMC simulations with a value of $\varepsilon^* = -1.7$. The fact that the two histograms are different illustrates the fact that it makes sense to perform pruning: there is a clear correlation between the partial weight of a chain at $m = 30$ and its chance of being accepted at the end of the growth process. Moreover, the separation between the two histograms becomes more pronounced with decreasing temperature and with increasing position in the chain. This allows us to estimate how to apply pruning in such a way that the overall acceptance rate of the Monte Carlo simulation is not negatively affected. In practice, we fix the pruning threshold at such that all



a)



b)

Figure 4. Normalized histogram of the partial weight at the position 30 of the chain. $\varepsilon^* = -1.7$. In (*a*) the histograms are obtained from a CBMC simulation. The curve with circles represents the histogram of the partial weight of created chains, and the solid line, the one for the accepted chains. In (*b*) we use a lower threshold indicated by the vertical solid line. The solid line with circles and the dashed line with squares respectively represent the histograms of the created chains using DPERM and CBMC. The two noisy dashed and solid curves that are almost superimposed respectively represent the histograms of the accepted chains using CBMC and DPERM. The error bars on these histograms could be obtained knowing that their statistics follow a Poisson distribution.

'good' chains survive. Figure 4 (*b*) shows a comparison between histograms using CBMC and DPERM with the pruning threshold fixed as described above. This simulation was done in the absence of enrichment. The effect of pruning is clearly seen and in this case does not affect the acceptance rate of the Monte Carlo simulation. Looking at the average position in the chain where the pruning occurs, we find that it occurs at about half of the chain in our case of a 48 monomer chain. Additionally this pruning occurs with a rate of order $\rho_{\text{pruning}} = 10\%$ for this value of ε^*: ρ_{pruning} is the number of pruned chains divided by the total number of chains we have tried to create. So one can easily calculate the CPU time gained by pruning. We define the relative efficiency ζ of DPERM compared to CBMC as the ratio of the CPU time spent by a CBMC algorithm to generate a set of N_{accept} chains with the Boltzmann distribution (i.e. to have N_{accept} chains accepted in the Monte Carlo simulation) divided by the CPU time spent by the DPERM to generate the same ensemble. Calling τ the average CPU time spent to add one monomer in the chain, a CBMC algorithm spends a CPU time:

$$\tau_{\text{CBMC}} = N_{\text{trial}} * l * \tau, \tag{19}$$

where N_{trial} is the total number of trials that have been performed and l is the length of the chain. In the case of DPERM, the CPU time spent is

$$\tau_{\text{PERM}} = N_{\text{trial}}(1 - \rho_{\text{pruning}})l\tau - N_{\text{trial}}\rho_{\text{pruning}}l_{\text{cut}}\tau, \tag{20}$$

where l_{cut} is the average position where chains are cut, and we use the same value of N_{trial} because the definition of the lower threshold does not affect the acceptance rate $\rho_{\text{accept}} = N_{\text{accept}}/N_{\text{trial}}$. So the efficiency ζ can be written as

$$\zeta = \frac{1}{1 - \rho_{\text{pruning}}[1 - (l_{\text{cut}}/l)]}. \tag{21}$$

From the above analysis it follows that ζ is never less than one: pruning can only speed up the simulation. But unfortunately, its effect is not very large. In the present case (assuming $\rho_{\text{pruning}} \approx 10\%$ and $l_{\text{cut}}/l \approx 1/2$), we estimate $\zeta \approx 1.05$. Such an improvement in efficiency is hardly significant. From equation (21), increasing the pruning rate would increase the efficiency, but one has to remember that this equation is only valid if pruning does not affect the acceptance rate. One could argue that increasing a little bit the pruning thresholds would allow us to prune a lot of chains without affecting too much the acceptance rate (see figure 4 (*b*)): still, this would not lead to a significant increase in the efficiency, since even with a pruning rate of about 20%, the efficiency would still be $\leqslant 1.1$.

The situation cannot be improved by enrichment of chains with a partial weight in excess of a certain threshold value. Grassberger demonstrated that enrichment is a useful device for finding the native state. However, in the case of DPERM, it is very time consuming yet does not significantly improve the acceptance rate: little is gained by enriching 'good' chains as these are anyway likely to be accepted in the importance sampling process. A similar case happens for the parallel version of the CBMC algorithm [14] in which multiple chains are grown simultaneously and only one is chosen with a certain probability.

The crucial difference between the dynamic and static PERM algorithms is that, in the latter scheme, all the chains which are created by enrichment are used in the evaluation of thermodynamic quantities. Therefore, the average CPU time to generate one chain decreases with the number of copies made. In contrast, in the DPERM algorithm, only one chain of the set will be

kept for a trial Monte Carlo move. Nevertheless, since we are likely to choose the best chain of the set for this trial move, one could expect to improve significantly the Monte Carlo acceptance rate by enriching.

Defining computational gain in the same way as we did above, we can estimate the effect of enrichment on efficiency. The CPU time spent by a CBMC simulation is

$$\tau_{\mathrm{CBMC}} = N_{\mathrm{trial}}^{\mathrm{cbmc}} * l * \tau. \tag{22}$$

With the DPERM algorithm, the CPU time spent is

$$\tau_{\mathrm{CBMC}} = N_{\mathrm{trial}}^{\mathrm{perm}} * l * \tau + N_{\mathrm{trial}}^{\mathrm{perm}} \rho_{\mathrm{enrich}} k (l - l_{\mathrm{enrich}}) \tau, \tag{23}$$

where ρ_{enrich} is the enrichment rate, l_{enrich} is the average position where enrichment occurs and k is the average size of sets that are generated. The efficiency can then be written as

$$\zeta = \frac{N_{\mathrm{trial}}^{\mathrm{cbmc}}}{N_{\mathrm{trial}}^{\mathrm{perm}}} \frac{1}{1 + \rho_{\mathrm{enrich}} k [1 + (l_{\mathrm{enrich}}/l)]} \tag{24}$$

$$= \frac{\rho_{\mathrm{accept}}^{\mathrm{perm}}}{\rho_{\mathrm{accept}}^{\mathrm{cbmc}}} \frac{1}{1 + \rho_{\mathrm{enrich}} k [1 + (l_{\mathrm{enrich}}/l)]}. \tag{25}$$

Since the enrichment should, on average, balance pruning, the product $\rho_{\mathrm{enrich}} * k$ should be of the same order as the pruning rate ρ_{pruning}. In the present case, that means that $\rho_{\mathrm{enrich}} * k \approx 10\%$ and moreover $l_{\mathrm{enrich}}/l \geqslant 0.5$. For such small enrichment rates, we never found a value for the ratio $\rho_{\mathrm{accept}}^{\mathrm{perm}}/\rho_{\mathrm{accept}}^{\mathrm{cbmc}}$ which was significantly higher than 1. A very optimistic estimate of the efficiency would lead to a value of 1.1. In practice, using enrichment, we have never obtained an efficiency higher than one.

In summary, in the examples of the DPERM scheme that we studied, we found that enrichment does not increase the computational efficiency.

5. Conclusion

Motivated by the efficiency of the PERM algorithm, we have modified this algorithm into a dynamic Markov-chain Monte Carlo algorithm.

This new method can be applied everywhere algorithms like CBMC apply: we have applied it to linear chains in this paper, but it could also be applied to polymers with more complex architecture. For the cases that we studied, this algorithm does not present a significant improvement in efficiency compared to existing dynamic algorithms, such as CBMC and we do not see any reason why this conclusion would be different in the case of branched polymers. However, it should be stressed that there is considerable freedom in choosing the criteria for pruning and enrichment. In particular, the Rosenbluth weight may not be the best quantity to use as a pruning criterion. Other quantities that are more strongly correlated with the contribution of a particular chain conformation to the final Boltzmann average, may yield better pruning criteria.

Also, as we mention above, since it is useless to enrich chains with a high Rosenbluth weight, one can imagine to enrich only a small window of the histograms presented in figure 4. However, it then becomes difficult to define a strategy for finding the optimal upper thresholds.

This article is dedicated to Dominique Levesque whose seminal contributions to many aspects of computer simulations have been a driving force behind many developments in that field. This research has been supported by a Marie Curie Fellowship of the European Community program 'Improving Human Research Potential and the socio-economic Knowledge Base' under contract number HPMF-CT-2001-01212. Disclaimer: the authors are solely responsible for information communicated and the European Commission is not responsible for any views or result expressed. The work of the FOM Institute is part of the research program of FOM and is made possible by financial support from the Netherlands organization for Scientific Research (NWO).

Appendix: Pseudo-code summary

We present here a pseudo-code summary for readers who would implement the DPERM algorithm.

(1) Generate a trial set of conformations using the PERM scheme: the weight of that set is given by $W_{\mathrm{total}}(\mathrm{new}) = \sum_{\mathrm{set}} W^*_{\mathrm{chain}}(\mathrm{new})$, where $W^*_{\mathrm{chain}}(\mathrm{new})$ is the weight of each chain.
(2) Choose one of these conformations for a trial move with the probability $P_{\mathrm{choose}}(\mathrm{new}) = W^*_{\mathrm{chain}}(\mathrm{new})/W_{\mathrm{total}}(\mathrm{new})$.
(3) 'Retrace' the old conformation and compute the weight of the corresponding set of conformations: $W_{\mathrm{total}}(\mathrm{old}) = \sum_{\mathrm{set}} W^*_{\mathrm{chain}}(\mathrm{old})$.
(4) Accept the trial move with the probability

$$\mathrm{acc}(\mathrm{old} \to \mathrm{new}) = \min\left(1, \frac{W_{\mathrm{total}}(\mathrm{new})}{W_{\mathrm{total}}(\mathrm{old})}\right). \tag{A 1}$$

The trial set of conformations of a chain of size l is generated using the PERM scheme.

Make a stack where the set of chains will be recorded. The first monomer is inserted at a random lattice position. The weight of this partial chain is $W^*(1) = \exp(-\beta u^{(1)})$, where $u^{(1)}$ is the energy of the

monomer. The stack has only one element: this chain of 1 monomer.

(1) Choose a position for the next monomer i of the chain on the top of the stack using the Rosenbluth scheme. The partial weight at position i of the chain is given by $W^*_{\text{partial}}(i) = w_i * W^*_{\text{partial}}(i-1)$, where w_i has been defined in § 2. If $i = l$ then remove this chain from the stack and record its weight. If the stack is not empty, restart from the partial chain on the top of it and repeat step 1. If it is empty, end the procedure.

(2) If $W^*_{\text{partial}}(i) < W^<(i)$. With a probability 1/2, choose one of the following steps.

   (a) Assign a weight $W^* = 0$ to the chain and forgive its generation. Remove it from the stack.

   Check the stack, if it is empty, end the procedure; if not, restart from the partial chain on the top of the stack and repeat step 1.

   (b) Multiply the weight of that chain by 2. $W^*_{\text{partial}}(i) = 2 * W^*_{\text{partial}}(i)$. Repeat step 1.

(3) If $W^*_{\text{partial}}(i) > W^<(i)$. Make k copies of this partial chain and put them onto the stack. Assign to each copies the weight $W^*_{\text{partial}}(i) = W^*_{\text{partial}}(i)/k$. Continue using the partial chain on the top of the stack and repeat step 1.

Similarly, to determine the weight of the old configuration we use the following steps.

(1) One of the chains is selected at random. This chain will be denoted by o.

(2) Compute the weight using the Rosenbluth technique exactly like in the CBMC algorithm. However, every time $W^*_{\text{partial}}(i) < W^<(i)$, multiply the weight by two $W^*_{\text{partial}}(i) = 2 * W^*_{\text{partial}}(i)$.

   Every time the weight $W^*_{\text{partial}}(i) > W^<(i)$, make $k-1$ copies and put them in a stack. Each of these copies as well as the chain o have a weight $W^*_{\text{partial}}(i) = W^*_{\text{partial}}(i)/k$.

(3) Starting from the chain on the top of the stack, restart from step 1 of the preceding scheme to produce the set of conformation using PERM.

The weight factor associated with the chain o and its set of conformations is then given by

$$W_{\text{total}}(\text{old}) = \sum_{\text{set}} W^*_{\text{chain}}(\text{old}). \qquad \text{(A 2)}$$

References

[1] Metropolis, N., Rosenbluth, A. W., Rosenbluth, N. M., Teller, A. N., and Teller, E., 1953, *J. chem. Phys.*, **21,** 1087.

[2] Des Cloizeaux, J., and Jannink, G., 1990, *Polymers in Solution* (Oxford: Clarendon Press).

[3] Frenkel, D., and Smit, B., 2002, *Understanding Molecular Simulation*, 2nd Edn (London: Academic Press).

[4] Rosenbluth, M. N., and Rosenbluth, A. W., 1955, *J. chem. Phys.*, **23,** 356.

[5] Batoulis, J., and Kremer, K., 1988, *J. Phys.* A: *Math. Gen.*, **21,** 127.

[6] Grassberger, P., 1997, *Phys. Rev.* E, **56,** 3682.

[7] Bastolla, U., Frauenkron, H., Gerstner, E., GrassBerger, P., and Nadler, W., 1998, *Proteins: Struc. Func. Gen.*, **32,** 52.

[8] Frauenkron, H., Bastolla, U., Gerstner, E., GrassBerger, P., and Nadler, W., 1998, *Phys. Rev. Lett.*, **80,** 3149.

[9] Bastolla, U., and Grassberger, P., 1987, *J. stat. Phys.*, **89,** 1061.

[10] Lau, K. F., and Dill, K. A., 1989, *Macromolecules*, **22,** 3986.

[11] Fields, G. B., Alonso, D. O. V., Stiger, D., and Dill, K. A., 1992, *J. phys. Chem.*, **96,** 3974.

[12] Chan, H. S., and Dil, K. A., 1994, *J. chem. Phys.*, **100,** 9238.

[13] Fan, K., Wang, J., and Wang, W., 2001, *Phys. Rev.* E, **64,** 041907/1.

[14] Esselink, K., Loyens, L. D. J. C., and Smit, B., 1995, *Phys. Rev.* E, **51,** 1560.