Biased Monte Carlo Methods

D. Frenkel

FOM Institute for Atomic and Molecular Physics, Amsterdam, NL

Abstract. Polymer simulations make extensive use of biased Monte Carlo schemes. In this paper, I describe a subset of polymer-simulation algorithms that aim to increase the sampling efficiency by biasing the selection of trial moves. Algorithms that belong to this category are the Configurational Bias MC method (CBMC), Dynamical Pruned Enriched Rosenbluth sampling (DPERM) and Recoil-Growth (RG) sampling.

INTRODUCTION

The original Metropolis Monte Carlo scheme [1] was designed to perform single-particle trial moves. For most simulations, such moves are perfectly adequate. However, in some cases it is more efficient to perform moves in which the coordinates of many particles are changed. A case in point is the sampling of polymer conformations. The conventional Metropolis algorithm is ill suited for polymer simulations because the natural dynamics of polymers is dominated by topological constraints (chains cannot cross). Hence, any algorithm that mimics the real motion of macromolecules will suffer from the same problem. For this reason, many algorithms have been proposed to speed up the Monte Carlo sampling of polymer conformations (see *e.g.* ref. [2]). The Configurational-Bias Monte Carlo (CBMC) method is a dynamic MC scheme that makes it possible to achieve large conformational changes in a single trial move that affects a large number of monomeric units [3, 4, 5, 6].

CBMC

The CBMC method is based on the Rosenbluth sampling scheme [8, 3, 4] for lattice systems. In this scheme, the molecular conformation is built up step-by-step, in such a way that, at every stage, the next monomeric unit is preferentially added in a direction that has a large Boltzmann weight. This increases the probability of generating a trial conformation that has no hard-core overlaps. As explained below, the probability of acceptance of the trial conformation is given by the ratio of the 'Rosenbluth weights' of the new and the old conformations. Whereas the original Rosenbluth scheme was devised for polymers on a lattice, the CBMC scheme will also work for chain molecules in continuous space. The advantage of the CBMC algorithm over many of the other, popular algorithms is that it can be used in cases where particle-insertion and particle-removal trial moves are essential. This is the case in Grand-Canonical and Gibbs-

Ensemble simulations. In addition, the CBMC can be used in the simulation of grafted chains and ring polymers.

Detailed balance

Before explaining the CBMC scheme, it is useful to recall the general recipe to construct a Monte Carlo algorithm. It is advisable (although not strictly obligatory [9])to start from the condition of detailed balance:

$$P_o \times P_{gen}(o \rightarrow n) \times P_{acc}(o \rightarrow n) = P_n \times P_{gen}(n \rightarrow o) \times P_{acc}(n \rightarrow o)\,, \tag{1}$$

where P_o (P_n) is the Boltzmann weight of the old (new) conformation, P_{gen} denotes the *a priori* probability to generate the trial move from o to n, and P_{acc} is the probability that this trial move will be accepted. From Eqn. 1 it follows that

$$\frac{P_{acc}(o \rightarrow n)}{P_{acc}(n \rightarrow o)} = \exp(-\beta \Delta U) \frac{P_{gen}(n \rightarrow o)}{P_{gen}(o \rightarrow n)}\,, \tag{2}$$

where $\exp(-\beta \Delta U)$ is the ratio of the Boltzmann weights of the new and old conformations. If we use the Metropolis rule to decide on the acceptance of MC trial moves, then Eqn 2 implies

$$P_{acc}(o \rightarrow n) = \min\left(1, \exp(-\beta \Delta U) \frac{P_{gen}(n \rightarrow o)}{P_{gen}(o \rightarrow n)}\right)\,. \tag{3}$$

Ideally, by biasing the probability to generate a trial conformation in the right way, we could make the term on the right-hand side of Eqn. 3 always equal to unity. In that case, every trial move will be accepted. This ideal situation can be reached in rare cases [7] Configurational bias Monte Carlo does not achieve this ideal situation. However it does lead to enhanced acceptance probability of trial moves that involve large conformational changes.

In CBMC, chain configurations are generated by successive insertion of the bonded segments of the chain. When the positions of the segments are chosen at random, it is very likely, that one of the segments will overlap with another particle in the fluid, which results in rejection of the trial move. The Rosenbluth sampling scheme increases the insertion probability by looking one step ahead. On lattices, the availability (i.e. the Boltzmann factor) of all sites adjacent to the previous segment can be tested. In continuous space, there are in principle an infinite number of positions that should be tested (e.g. in the case of a chain molecule with rigid bonds, all points on the surface of a sphere with a radius equal to the bond length). Of course, it is not feasible to scan an infinite number of possibilities. Surprisingly, it turns out that it is possible to construct a correct Monte Carlo scheme for off-lattice models in which only a finite number of trial segments (k), is selected either at random or, more generally, drawn from the distribution of bond-lengths and bond-angles of the 'ideal' chain molecule.

During a CBMC trial move, a polymer conformation is generated segment-by-segment. At every step, k trial segments are generated (k is, in principle arbitrary and

can be chosen to optimize computational efficiency). One of these segments, say i, is selected with a probability

$$P_i = \frac{\exp(-\beta u_i)}{\sum_{j=1}^{k} \exp(-\beta u_j)}$$

where u_j is the change in the potential energy of the system that would result of this particular trial segment was added to the polymer. The probability to generate a complete conformation Γ_{new} consisting of ℓ segments is then

$$P(\Gamma)_{new} = \prod_{n=1}^{\ell} \frac{\exp(-\beta u_i(n))}{\sum_{j=1}^{k} \exp(-\beta u_j(n))}$$

To keep the equations simple, we only consider the expression for one of the ℓ segments.

$$P_{gen}(\{j\}) = d\Gamma_j P_{id}(\Gamma_j) \left[\prod_{j' \neq j}^{k} d\Gamma_{j'} P_{id}(\Gamma_{j'}) \right] \frac{\exp(-\beta u_{ext}(j))}{\sum_{j'=1}^{k} \exp(-\beta u_{ext}(j'))} \tag{4}$$

In order to compute the acceptance probability of this move, we have to consider what happens in the reverse move. Then we start from conformation j and generate a set of k trial directions that includes i. When computing the acceptance probability of the forward move, we have to impose detailed balance. However, detailed balance in this case means not just that in equilibrium the number of moves from i to j is equal to the number of reverse move, but even that the rates are equal *for any given set of trial directions for the forward and reverse moves*. This condition we call 'super-detailed balance'. Super-detailed balance implies that we can only decide on the acceptance of the forward move if we also generate a set of $k-1$ trial directions around the old conformation i. We denote the probability to generate this set of $k-1$ trial orientations by $P_{rest}(\{i\})$, where the sub-script 'rest' indicates that this is the set of orientations around, but excluding, i. This allows us to compute the ratio $w_j^{(t)}/w_i^{(o)}$ of the Rosenbluth weights for forward and reverse moves:

$$w_j^{(t)} = \frac{\exp(-\beta u_{ext}^{(t)}(j) + \sum_{j' \neq j}^{k-1} \exp(-\beta u_{ext}^{(t)}(j'))}{k}$$

and

$$w_i^{(o)} = \frac{\exp(-\beta u_{ext}^{(o)}(i) + \sum_{i' \neq i}^{k-1} \exp(-\beta u_{ext}^{(o)}(i'))}{k}$$

The superscript (t) and (o) distinguish the **t**rial conformation from the **o**ld conformation. The acceptance probability is determined by the ratio $x \equiv w_j^{(t)}/w_i^{(o)}$ (actually, for a molecule of ℓ segments, we should compute a product of such factors). Let us assume that $w_j^{(t)} < w_i^{(o)}$. In that case, $P_{acc}(i \to j) = x$ while $P_{acc}(j \to i) = 1$. Next, let us check whether detailed balance is satisfied. To do so, we write down the explicit expressions for K_{ij} and K_{ji}.

$$K_{ij} = N_i P_{gen}(\{j\}) P_{rest}(\{i\}) w_j^{(t)}/w_i^{(o)}$$

and

$$K_{ji} = N_j P_{gen}(\{i\}) P_{rest}(\{j\}) 1$$

In addition, we use the fact that

$$P_{gen}(\{j\}) P_{rest}(\{i\}) w_j^{(t)} \sim N_j P_{rest}(\{j\}) P_{rest}(\{i\})$$

and

$$P_{gen}(\{i\}) P_{rest}(\{j\}) w_j^{(t)} \sim N_i P_{rest}(\{i\}) P_{rest}(\{j\})$$

In then follows immediately that

$$\begin{aligned} K_{ij} w_i^{(o)} &= N_i P_{gen}(\{j\}) P_{rest}(\{i\}) w_j^{(t)} \qquad (5) \\ &= \text{constant} \times N_i N_j P_{rest}(\{i\}) P_{rest}(\{j\}) \\ &= N_j P_{gen}(\{i\}) P_{rest}(\{j\}) w_i^{(o)} \\ &= K_{ji} w_i^{(o)} \qquad (6) \end{aligned}$$

Hence, K_{ij} is indeed equal to K_{ji}. Note that, in this derivation, the number of trial directions, k, was arbitrary. The procedure sketched above is valid for a complete regrowth of the chain, but it is also possible to regrow only part of a chain, i.e. to cut a chain at a (randomly chosen) point and regrow the cut part of the chain either at the same site or at the other end of the molecule. Clearly, if only one segment is regrown and only one trial direction is used, CBMC reduces to the reptation algorithm (at least, for linear homo-polymers).

We still have to consider the choice for the number of trial directions at the $i-th$ regrowth step, k_i. Too many trial directions increase the cost of a simulation cycle, but too few trial directions lower the acceptance rate, and increase the simulation length. There exist simple guidelines that allow us to select k_i for every segment such that it optimizes the efficiency of the simulation [11].

BEYOND CBMC

The CBMC method has several drawbacks. First of all, as the scheme looks ahead only one step at a time, it is likely to end up in "dead-alleys". Secondly, for long chain molecules, the Rosenbluth weight of the trial conformation tends to become quite small. Hence, much of the computational effort may be wasted on "doomed" configurations.

DPERM

Grassberger and co-workers [12] have suggested adding two ingredients to the Rosenbluth scheme 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 have a low Rosenbluth weight. 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 [12, 13], 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 [14, 15] indicate that the PERM approach can be very useful to estimate the thermal equilibrium properties of long polymers. The main limitation of both the original Rosenbluth method and the PERM algorithm is that they are "static" Monte Carlo schemes. Such schemes can simulate single polymer chains very efficiently, but are less suited to simulate 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.

Just as CBMC is the "dynamic" version of Rosenbluth sampling. Similarly, one can construct a dynamic version of the PERM algorithm: DPERM [16].

The static PERM algorithm uses the Rosenbluth 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 $W(j) = \Pi_{i=1}^{j} w_i$ of a configuration is below a lower threshold $W^{<}(j)$, there is a probability of 50% 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 $W(j) = \Pi_{i=1}^{j} w_i$ 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).

The DPERM algorithm is the dynamic generalization of the PERM algorithm. As in the CBMC algorithm, we bias the acceptance of trials 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. As we use the Rosenbluth method to generate chains, the probability to grow a particular conformation is :

$$P_{gen}(chain) = \Pi_{i=1}^{l} \frac{e^{-\beta u^{(i)}(n)}}{w_i} \tag{7}$$

Whenever the re-weighted Rosenbluth partial weight $W^{*}(j)$ of the chain drops below the lower threshold $W^{<}(j)$, the chain has a 50% probability of being deleted. Let us assume that this happens m times. Then, the total probability to generate a *particular* conformation is :

$$P_{gen}(chain) = \frac{1}{2^m} \Pi_{i=1}^{l} \frac{e^{-\beta u^{(i)}(n)}}{w_i} \tag{8}$$

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

$$W^{*}(\text{chain\_new}) \;=\; 2^m * W(\text{chain\_new}) \tag{9}$$

$$\text{with } W(\text{chain\_new}) = \Pi_{i=1}^{l} w_i \tag{10}$$

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 :

$$P_{gen}(\text{chain\_new}) * P_{gen}(\text{rest\_new}) \tag{11}$$

Where $P_{gen}(\text{rest\_new})$ 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 :

$$P_{gen}(\text{chain\_new}) = k^p \Pi_{i=1}^{l} \frac{e^{-\beta u^{(i)}(n)}}{w_i} \tag{12}$$

and its re-weighted Rosenbluth weight is :

$$W^*(\text{chain\_new}) = \frac{1}{k^p} W(\text{chain\_new}) \tag{13}$$

Here k is the number of copies that are created each time the Rosenbluth weight exceeds the upper threshold. In Eq. 12, 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. And we deduce Eq. 13 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 Eq. 12 becomes :

$$\begin{aligned} P_{gen}(\text{chain\_new}) &= \frac{k^p}{2^m} \Pi_{i=1}^{l} \frac{e^{-\beta u^{(i)}(n)}}{w_i} \\ &= \frac{k^p}{2^m} \frac{e^{-U(n)}}{W(\text{chain\_new})} \end{aligned} \tag{14}$$

And Eq. 13 becomes :

$$W^*(\text{chain\_new}) = \frac{2^m}{k^p} W(\text{chain\_new}) \tag{15}$$

Note that Eq. 14 and Eq. 15 respectively reduce to Eq. 8 and Eq. 9 in the absence of enrichment ($p = 0$) and to Eq. 12 and Eq. 13 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 :

$$P_{\text{choose\_new}} = W^*(\text{chain\_new})/W_{total}(\text{new}) \tag{16}$$

where $W^*(\text{chain\_new})$ is the re-weighted Rosenbluth weight mentioned in Eq. 15 and W_{total} is the sum of all such weights

$$W_{total}(\text{new}) = \sum_{set} W^*_{\text{chain}} \tag{17}$$

Eq. 16 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 :

$$acc(old \rightarrow new) = min(1, \frac{W_{total}(new)}{W_{total}(old)}) \tag{18}$$

To calculate $W_{total}(old)$, 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.

The demonstration that this scheme satisfies detailed balance is given in ref. [16]

This algorithm contains even more free parameters than CBMC. We choose them to optimize computational efficiency. In practice, this usually means that we wish to generate "enough" potentially successful trial chains, but not too many. Typically, the number of chains that survive at the end of a trial move should be $\mathcal{O}(1)$.

Recoil-growth

The recoil growth (RG) scheme is a dynamic Monte Carlo algorithm that was also developed with the dead-alley problem in mind [17, 18]. The algorithm is related to earlier static MC schemes due to Meirovitch [19] and Alexandrowicz and Wilding [20]. The basic strategy of the method is that it allows us to escape from a trap by "recoiling back" a few monomers and retrying the growth process using another trial orientation. In contrast, the CBMC scheme looks only one step ahead. Once a trial orientation has been selected, we cannot "deselect" it, even if it turns out to lead into a dead alley. The recoil growth scheme looks several monomers ahead to see whether traps are to be expected before a monomer is irrevocably added to the trial conformation. In this way we can alleviate (but not remove) the dead-alley problem. In principle, one could also do something similar with CBMC by adding a sequence of l monomers per step. However, as there are k possible directions for every monomer, this would involve computing k^l energies per group. Even though many of these trial monomers do not lead to acceptable conformations, we would still have to compute all interaction energies.

The RG algorithm is best explained by considering a totally impractical, but conceptually simple schemer that has the same effect. We place the first monomer at a random position. Next, we generate k trial positions for the second monomer. From each of these trial positions, we generate k trial positions for the third monomer. At this stage, we have generated k^2 "trimer" chains. We continue in the same manner until we have grown k^{l-1} chains of length l. Obviously, most of the conformations thus generated have a vanishing Boltzmann factor and are, therefore, irrelevant. However, some may have a reasonable Boltzmann weight and it is these conformations that we should like to find. To simplify this search, we introduce a concept that plays an important role in the RG algorithm: we shall distinguish between trial directions that are "open" and those that are "closed." To decide whether a given trial direction, say b, for monomer j is open, we compute its energy $u_j(b)$. The probability that trial position b is open is given by

$$p_j^{\text{open}}(b) = \min(1, \exp[-\beta u_j(b)]), \tag{19}$$

For hard-core interactions, the decision whether a trial direction is open or closed is unambiguous, as $p_j^{\text{open}}(b)$ is either zero or one. For continuous interactions we compare $p_j^{\text{open}}(b)$ with a random number between 0 and 1. If the random number is less than $p_j^{\text{open}}(b)$, the direction is open; otherwise it is closed. We now have a tree with k^{l-1} branches but many of these branches are "dead," in the sense that they emerge from a "closed" monomer. Clearly, there is little point in exploring the remainder of a branch if it does not correspond to an "open" direction. This is where the RG algorithm comes in. Rather than generating a host of useless conformations, it generates them "on the fly." In addition, the algorithm uses a cheap test to check if a given branch will "die" within a specified number of steps (this number is denoted by l_{max}). The algorithm then randomly chooses among the available open branches. As we have only looked a distance l_{max} ahead, it may still happen that we have picked a branch that is doomed. But the probability of ending up in such a dead alley is much lower than that in the CBMC scheme.

In practice, the recoil growth algorithm consists of two steps. The first step is to grow a new chain conformation using only "open" directions. The next step is to compute the weights of the new and the old conformations.

The following steps are involved in the generation of a new conformation:

1. The first monomer of a chain is placed at a random position. The energy of this monomer is calculated (u_1). The probability that this position is "open" is given by Eqn. 19. If the position is closed we cannot continue growing the chain and we reject the trial conformation. If the first position is open, we continue with the next step.
2. A trial position b_{i+1} for monomer $i+1$ is generated starting from monomer i. We compute the energy of this trial monomer $u_{i+1}(b)$ and, using Eqn. 19, we decide whether this position is open or closed. If this direction is closed, we try another trial position, up to a maximum of k trial orientations. As soon as we find an open position we continue with step 3.
   If not a single open trial position is found, we make a recoil step. The chain retracts one step to monomer $i-1$ (if this monomer exists), and the unused directions (if

any) from step 2, for $i-1$, are explored. If all directions at level $i-1$ are exhausted, we attempt to recoil to $i-2$. The chain is allowed to recoil a total of $l_{\max}$ steps, i.e., down to length $i-l_{\max}+1$.
If, at the maximum recoil length, all trial directions are closed, the trial conformation is discarded.

3. We have now found an "open" trial position for monomer $i+1$. At this point monomer $i-l_{\max}$ is permanently added in the new conformation; i.e., a recoil step will not reach this monomer anymore.
4. Steps 2 and 3 are repeated until the entire chain has been grown.

In the naive version of the algorithm sketched above, we can consider the above steps as a procedure for searching for an open branch on the existing tree. However, the RG procedure does this by generating the absolute minimum of trial directions compatible with the chosen recoil distance $l_{\max}$.

Once we have successfully generated a trial conformation, we have to decide on its acceptance. To this end, we have to compute the weights, $W(n)$ and $W(o)$, of the new and the old conformations, respectively. This part of the algorithm is more expensive. However, we only carry it out once we know for sure that we have successfully generated a trial conformation. In contrast, in CBMC it may happen that we spend much of our time computing the weight factor for a conformation that terminates in a dead alley.

In the RG scheme, the following algorithm is used to compute the weight of the new conformation:

1. Consider that we are at monomer position i (initially, of course, $i=1$). In the previous stage of the algorithm, we have already found that at least one trial direction is available (namely, the one that is included in our new conformation). In addition, we may have found that a certain number of directions (say k_c) are closed—these are the ones that we tried but that died within $l_{\max}$ steps. We still have to test the remaining $k_{\text{rest}} \equiv k-1-k_c$ directions. We randomly generate k_{rest} trial positions for monomer $i+1$ and use the recoil growth algorithm to test whether at least one "feeler" of length $l_{\max}$ can be grown in this direction grown (unless $i+l_{\max} > l$; in that case we only continue until we have reached the end of the chain). Note that, again, we do *not* explore all possible branches. We only check if there is at least *one* open branch of length $l_{\max}$ in each of the k_{rest} directions. If this is the case, we call that direction "available." We denote the total number of available directions (including the one that corresponds to the direction that we had found in the first stage of the algorithm) by m_i. In the next section we shall derive that monomer i contributes a factor $w_i(n)$ to the weight of the chain, where $w_i(n)$ is given by
$$w_i(n) = \frac{m_i(n)}{p_i^{\text{open}}(n)}$$
and $p_i^{\text{open}}(n)$ is given by Eqn. 19.
2. Repeat the previous step for all i from 1 to $l-1$. The expression for the partial weight of the final monomer seems ambiguous, as $m_l(n)$ is not defined. An easy (and correct) solution is to choose $m_l(n)=1$.

3. Next compute the weight for the entire chain:

$$W(n) = \prod_{i=1}^{\ell} w_i(n) = \prod_{i=1}^{\ell} \frac{m_i(n)}{p_i^{\text{open}}(n)}. \tag{20}$$

For the calculation of the weight of the old conformation, we use almost the same procedure. The difference is that, for the old conformation, we have to generate $k-1$ additional directions for every monomer i. The weight is again related to the total number of directions that start from monomer i and that are "available," i.e., that contain at least one open feeler of length $l_{\max}$:

$$W(o) = \prod_{i=1}^{\ell} w_i(o) = \prod_{i=1}^{\ell} \frac{m_i(o)}{p_i^{\text{open}}(o)}.$$

Finally, the new conformation is accepted with a probability:

$$\text{acc}(o \to n) = \min(1, \exp[-\beta U(n)]W(n)/\exp[-\beta U(o)]W(o)), \tag{21}$$

where $U(n)$ and $U(o)$ are the energies of the new and old conformations, respectively. It can easily be demonstrated (see [17, 18]) that this scheme generates a Boltzmann distribution of conformations.

CONCLUSIONS

A comparison between CBMC and the RG algorithm was made by Consta *et al.* [18], who studied the behavior of Lennard-Jones chains in solution. The simulations showed that for relatively short chains ($\ell = 10$) at a density of $\rho = 0.2$, the recoil growth scheme was a factor of 1.5 faster than CBMC. For higher densities $\rho = 0.4$ and longer chains $N = 40$ the gain could be as large as a factor 25. This illustrates the fact that the recoil scheme is still efficient, under conditions where CBMC is likely to fail. For still higher densities or still longer chains, the relative advantage of RG would be even larger. However, the bad news is that, under those conditions, *both* schemes become very inefficient. A similar comparison has been made between CBMC and DPERM [16]. For the cases studied, DPERM was no worse than CBMC, but also not much better. However, the fact that pruning can be performed on any chain property (i.e. not necessarily the Rosenbluth weight), may make the method attractive in special cases.

The basic idea behind both DPERM and RG is that these algorithms aim to avoid investing computational effort in the generation of trial moves that are, in the end, rejected. Houdayer [21] has proposed an algorithm that aims to achieve the same. In this algorithm, the trial move is a so-called "wormhole" move, where a polymer grows in one part of the simulation box while it shrinks at its original location. The growth-shrinkage process is carried our using a reptation-like algorithm. This algorithm has the advantage that, even if trial moves to the new state are rejected, the "old" state has also changed. This speeds up relaxation. In addition, the computing effort for the wormhole scheme appears to scale favorably with polymer size for long polymers (namely as

N^n [22], rather than as $\exp(cN)$). However, for short chains, the existing schemes are almost certainly more efficient (in ref. [21], the comparison with CBMC is made for a particularly inefficient CBMC-parameter choice). For longer chains, the wormhole scheme really should win. However, in that regime, all schemes are extremely costly. Nevertheless, as pointed out in ref. [21], a combination of the various algorithms would probably be more efficient than any one of them alone.

ACKNOWLEDGMENTS

The work of the FOM Institute is part of the research programme of FOM, and is made possible by financial support from NWO.

REFERENCES

1. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A.H.Teller, and E.Teller, J.Chem. Phys. **21**, 1087 (1953).
2. Monte Carlo and Molecular Dynamics Simulations in Polymer Science Edited by K. Binder, OUP, Oxford, 1995.
3. J. Harris and S. A. Rice, J. Chem. Phys. **88**, 1298 (1988).
4. J. I. Siepmann and D. Frenkel, Mol.Phys. **75**, 59 (1992).
5. D. Frenkel, G. C. A. M. Mooij, and B. Smit, J.Phys. Condensed Matter **4**, 3053 (1992).
6. J. J. de Pablo, M. Laso, and Suter U. W, J. Chem. Phys. **96**, 2394 (1992).
7. R. H Swendsen and J. S. Wang, Phys. Rev. Lett. **58**, 86 (1987).
8. M. N. Rosenbluth and A. W. Rosenbluth, J.Chem. Phys. **23**, 356 (1955).
9. V.I. Manousiouthakis and M.W. Deem, J. Chem. Phys., **110**,2753(1999).
10. G. C. A. M. Mooij, D. Frenkel, and B. Smit, J. Phys. Condensed Matter **4**, L255 (1992).
11. G. C. A. M. Mooij and D.Frenkel, Molecular Simulation, **17**,41(1996).
12. P. Grassberger, Phys. Rev. E **56**, 3682 (1997).
13. U. Bastolla, H. Frauenkorn, E. Gerstner, P. Grassberger and W. Nadler, Proteins: Struc. Func. Gen. **32**,52 (1998).
14. H. Frauenkorn, U. Bastolla, E. Gerstner, P. Grassberger and W. Nadler, Phys. Rev. Lett. **80**,3149 (1998).
15. U. Bastolla and P. Grassberger, J. Stat. Phys. **89**,1061 (1997).
16. N. Combe, P.R. ten Wolde, T.J.H. Vlugt, Mol. Phys. (in press)
17. S.Consta, T. J. H. Vlugt, J.Wichers Hoeth, B.Smit and D.Frenkel, Mol. Phys. **97**,1243(1999)
18. S. Consta, N. B. Wilding, D. Frenkel and Z. Alexandrowicz, J. Chem. Phys. **110**,3220(1999)
19. H. Meirovitch, J. Chem. Phys., **89**,2514,(1988).
20. Z. Alexandrowicz and N.B. Wilding, J. Chem. Phys., **109**,5622(1998).
21. J. Houdayer, J. Chem. Phys. **116**, 1783(2002)
22. Houdayer [21] suggests that the computing time per accepted move should scale as N^2. However, as the problem is asymptotically that of a one-dimensional random walk with absorbing boundaries, it is more plausible that, for very long chains, the computing time per accepted move should scale as N^3.