New Monte Carlo algorithms for interacting self-avoiding walks

1997 J. Phys. A: Math. Gen. 30 1457

(http://iopscience.iop.org/0305-4470/30/5/014)

J. Phys. A: Math. Gen. **30** (1997) 1457–1469. Printed in the UK PII: S0305-4470(97)74266-5

New Monte Carlo algorithms for interacting self-avoiding walks

P P Nidras and R Brak†

Department of Mathematics, The University of Melbourne, Parkville, Victoria 3052, Australia

Received 8 May 1996

Abstract. In this paper we present two new Monte Carlo algorithms for simulating interacting self-avoiding walks. These algorithms are built on the Berretti–Sokal and reptation algorithms for simulating non-interacting self-avoiding walks respectively by adding a Metropolis step. We enhance each algorithm by introducing special local moves. In addition, the reptation algorithm is further enhanced by running the simulations as a multiple Markov chain. Each of these algorithms are studied here through an analysis of their autocorrelation properties. In the case of the Berretti–Sokal algorithm an increase in efficiency by a factor of around 40 is achieved for non-interacting self-avoiding walks.

1. Introduction

There have recently been several Monte Carlo studies on the interacting self-avoiding walk (ISAW) and related models [1–5]. In this paper we will present two new Monte Carlo algorithms. Both algorithms are dynamic Monte Carlo algorithms which generate a highly correlated sequence of ISAWs from a realization of a Markov chain with a given unique limit distribution.

The first algorithm we study generates self-avoiding walks (SAWs) in the grand canonical, or fixed fugacity, ensemble and was introduced by Berretti and Sokal a decade ago [6]. It is now generally referred to as the Berretti–Sokal (B–S), or sometimes 'slithering tortoise' algorithm. The second algorithm has been around for more than a quarter of a century and goes under the name of the 'reptation' or 'slithering snake' algorithm. It was first introduced by Kron *et al* [7] and later Wall and Mandel [8] independently introduced their own version. The algorithm generates SAWs of fixed length, i.e. in the canonical ensemble. In its original form the algorithm is *not* ergodic but the addition of a combination of local and/or bilocal moves rectifies this flaw [9].

The efficiency of dynamic Monte Carlo algorithms is usually determined by the *integrated* autocorrelation time. This quantity can be estimated for every observable. It determines how many Monte Carlo steps are needed to produce an effectively independent configuration. The *exponential* autocorrelation time can also be determined for each observable. More importantly, we can define the exponential autocorrelation time, τ_{exp}, of the actual Markov chain as the relaxation time of the slowest mode of the system. This quantity measures the rate of convergence of the Markov chain to its equilibrium distribution from an arbitrary initial distribution. In practice it tells us how many Monte Carlo moves

† E-mail address: ppn,brak@mundoe.maths.mu.oz.au

0305-4470/97/051457+13$19.50

we need to execute before sampling from the Markov chain in order to avoid any bias in the data from the initial state.

The exponential autocorrelation time plays an important role in algorithms employing global moves such as the pivot algorithm where the slowest mode is an order of n times longer than the faster modes which arise from global observables (n is the length of the ISAW). Since the algorithms that we are concerned with here employ only local and bilocal moves, they do not have such a problem.

In section 2 we describe the B–S algorithm and a new local move which dramatically improves its efficiency. We then give a heuristic argument to explain the resulting increased efficiency. We then describe the addition of a Metropolis move which enables us to simulate the effect of interactions at finite temperatures. We also discuss some programming techniques which minimize the extra CPU time that is needed to execute this new move. We repeat this procedure in section 3 for the reptation algorithm. In section 4 we estimate the autocorrelation times of both algorithms for a variety of parameter values and compare these with the other versions of the algorithms.

In this paper we only report on the relative efficiencies of the various algorithms. The determination of the various properties of the ISAWs will be reported in a future publication [10]. In particular we verify the new scaling form for the canonical partition function for ISAWs in the low temperature phase which was proposed by Owczarek *et al* [11].

2. The Berretti–Sokal–Metropolis algorithm

The B–S algorithm generates SAWs in a grand canonical ensemble at fixed monomer fugacity z with one endpoint fixed at the origin and the other endpoint free. Each n-step walk has a probability of $z^n/G(z)$ of occurring in the ensemble, where

$$G(z) = \sum_{n=0}^{\infty} c_n z^n \tag{1}$$

is the grand partition function and c_n is the number of n-step SAWs.

There are two possible moves that can be attempted—the addition of a single bond onto the end of the current walk in one of q possible directions (where q is the coordination number of the lattice), or the deletion of the last bond of the walk. An attempt to append a bond onto the end of the walk is successful if the self-avoidance constraint is not violated. If it is violated and the walk intersects itself, then that move is rejected and the old configuration is counted again in the sample (this is known as a null transition). Attempting to delete a bond from the end of the walk is always successful unless the current walk is the empty walk, i.e. $n = 0$. If this is the case, then a null transition occurs and the empty walk is counted once more in the sample.

The relative probabilities of attempting to add or delete a bond from the walk are as follows:

$$\Pr(\text{attempting to add any bond}) = \frac{qz}{1+qz} \tag{2}$$

$$\Pr(\text{attempting to delete a bond}) = \frac{1}{1+qz}. \tag{3}$$

These probabilities are chosen to satisfy the detailed-balance condition for the grand canonical ensemble at monomer fugacity z, i.e.

$$\pi(\psi)\, p(\psi \to \psi') = \pi(\psi')\, p(\psi' \to \psi). \tag{4}$$

Here the transition probability, $p(\psi \to \psi')$, is the probability of obtaining the walk ψ' from the walk ψ after one Monte Carlo step. The probability distribution $\pi(\psi)$ is the equilibrium distribution of the Markov chain and is given by

$$\pi(\psi) = \frac{z^{|\psi|}}{G(z)} \chi_{\mathrm{SAW}}(\psi) \tag{5}$$

where $|\psi|$ denotes the length of the walk and $\chi_{\mathrm{SAW}}(\psi)$ is given by

$$\chi_{\mathrm{SAW}}(\psi) = \begin{cases} 1 & \text{if } \psi \text{ is a SAW} \\ 0 & \text{if } \psi \text{ is not a SAW .} \end{cases} \tag{6}$$

2.1. Simulating ISAWs

When simulating ISAWs, we introduce a Boltzmann factor $\omega = \mathrm{e}^{\beta\epsilon}$ where $\beta = 1/k_B T$ and $\epsilon \geqslant 0$ is the contact energy, i.e. the amount of energy associated with each pair of nearest-neighbour vertices of the walk (not connected by an edge of the walk). Each contact in a configuration now contributes a weight of ω so that an ISAW will appear in the ensemble with probability

$$\pi(\psi) = \frac{z^{|\psi|}\omega^{m(\psi)}}{G(z,\omega)} \chi_{\mathrm{SAW}}(\psi) \tag{7}$$

where $m(\psi)$ is the number of contacts in the configuration ψ, $G(z,\omega)$ is the grand partition function for ISAWs,

$$G(z,\omega) = \sum_{n=0}^{\infty} z^n Z_n(\omega) \tag{8}$$

where $Z_n(\omega)$ is the canonical partition function

$$Z_n(\omega) = \sum_{m=0}^{\infty} c_n^m \omega^m \tag{9}$$

and c_n^m is the number of configurations of n-step walks with m nearest-neighbour contacts.

In order to correctly weight each configuration appearing in the sample, we need (7) to be the equilibrium distribution of the Markov chain. This can be achieved by using the standard Metropolis step [12] after any successful B–S move. To do this we calculate the energy produced by a Monte Carlo move which is $\Delta m = m(t+1) - m(t)$, where $m(t)$ is the number of nearest-neighbour contacts after the tth Monte Carlo step. This change in energy gives us the probability of accepting the trial move as follows. We let $p = \min(1, \omega^{\Delta m})$. If $p = 1$ then the trial move is automatically accepted. If $p < 1$ we choose a (pseudo) random number uniformly distributed between 0 and 1 and compare it with p. If the number is smaller than p, then the trial move is accepted; otherwise it is rejected resulting in a null transition. Notice that $p < 1$ corresponds to $\Delta m < 0$ which is possible only when we attempt to delete a bond from the walk. Thus, any successful attempt at adding a bond to the end of the walk is always accepted. We will refer to a B–S move combined with a Metropolis step as the Berretti–Sokal–Metropolis (B–S–M) algorithm.

2.2. Enhanced B–S–M algorithm

A significant improvement in the efficiency of this algorithm can be obtained by the following alteration. Instead of appending a single bond onto the end of the walk, we append another SAW of length Δn. Similarly, the deletion of a single bond at the end of

the walk is now replaced by the deletion of the last Δn bonds of the walk. This is similar to the idea of strides due to Wall *et al* [13]. In order to avoid any bias, the particular mini-walk that is appended to the current walk has to be chosen uniformly from all walks of length Δn. Since the number of walks grows exponentially with increasing n, in practice we are restricted in our choice of Δn by the amount of computer memory. As an example of the memory requirements, choosing $\Delta n = 17$ for two-dimensional walks requires approximately 50 MB of memory.

Since the lengths of the walks generated by the algorithm are multiples of Δn, the generating function for the ensemble of walks is changed from (1) to

$$G(z) = \sum_{k=0}^{\infty} z^{k\Delta n} c_{k\Delta n}. \tag{10}$$

The transition probabilities given in (3) also have to be changed as follows:

$$\Pr(\text{attempting to add } \Delta n \text{ bonds}) = \frac{c_{\Delta n} z^{\Delta n}}{1 + c_{\Delta n} z^{\Delta n}} \tag{11}$$

$$\Pr(\text{attempting to delete } \Delta n \text{ bonds}) = \frac{1}{1 + c_{\Delta n} z^{\Delta n}}. \tag{12}$$

The extension required for the above move to simulate ISAWs is straightforward—just apply the Metropolis step as mentioned above.

We now consider the efficiency of the algorithm for SAWs with this additional move. First we note the heuristic argument provided by Berretti and Sokal which suggests that the integrated autocorrelation time τ should be $\mathcal{O}(\langle n \rangle^2)$. This is in fact correct to leading order if ordinary random walks (i.e. with intersections allowed) are simulated with the algorithm (see [6, appendix A]); this is a lower bound for τ. The argument assumes that the length of the walks being simulated will execute a random walk on the integers with inward drift. Thus, on average it will take $\langle n \rangle^2$ Monte Carlo steps for the empty walk to be reached upon which a completely independent configuration will be generated since all memory of the past is erased. In a later paper [14] an upper bound was obtained so that $\mathcal{O}(\langle n \rangle^2) \leqslant \tau \leqslant \mathcal{O}(\langle n \rangle^{\gamma+1})$, where γ is a critical exponent associated with the entropy of SAWs and is greater than 1. Here we argue that the reduction in τ ought to be $\mathcal{O}(\Delta n^2)$, at least for ordinary random walks, when using the improvement mentioned above. To see this simply replace $\langle n \rangle$ in the original argument of Berretti and Sokal by $\langle n \rangle / \Delta n$. In section 4 we will provide some evidence to show that the reduction in τ more or less follows this trend for SAWs. Therefore, this local move has the effect of reducing the length of the walk by a factor of Δn. Of course the consequence is an increase in the CPU time needed to execute a Monte Carlo step, as we shall see below.

For ISAWs the Δn dependence of τ is rather more complicated. We find that the algorithm no longer necessarily becomes more efficient with increasing Δn. Instead, there is now some optimal value of Δn, above which the efficiency of the algorithm decreases. Furthermore, this optimal value of Δn decreases as the temperature is lowered. Thus, the move is less effective for lower temperatures. This phenomenon is due to two effects, the first being that as the walk collapses it is increasingly difficult to append many bonds onto the end of the walk without violating self-avoidance. Secondly, the chance of the Metropolis move being successful rapidly decreases with decreasing temperature, (i.e. increasing ω)—$p_{\text{accept}} = 1/\omega^{\Delta m}$.

There is no particular reason for choosing to append/delete only SAWs of fixed length Δn when using the algorithm. We therefore tried to further improve the efficiency of the algorithm by allowing Δn to vary. This is implemented by deciding on the maximum value

that Δn can take, call it k. A weight is then prescribed to every walk of length up to k. We determined empirically that improved results can be achieved by choosing either

$$w_i = \frac{i}{\sum_{i=1}^{k} i} \tag{13}$$

or

$$w_i = \frac{i^2}{\sum_{i=1}^{k} i^2}. \tag{14}$$

The modified transition probabilities for this alteration are

$$\Pr(\text{attempting to add } i \text{ bonds}) = \frac{c_i z^i w_i}{1 + \sum_{i=1}^{k} c_i z^i w_i} \tag{15}$$

$$\Pr(\text{attempting to delete } i \text{ bonds}) = \frac{w_i}{1 + \sum_{i=1}^{k} c_i z^i w_i}. \tag{16}$$

For a certain range of temperatures this technique provided a gain of between 20% and 50% in the efficiency of the algorithm over the fixed Δn results.

2.3. Programming techniques

The efficiency of a Monte Carlo algorithm is not solely determined by its autocorrelation time. The average amount of CPU time per Monte Carlo step is the other factor determining the algorithm's efficiency. Since both algorithms that we consider here only employ local/bilocal moves, the CPU time is $\mathcal{O}(1)$. However, the relevant question in our case is: how much does the move suggested above increase the CPU time per Monte Carlo step? The increase is linear in Δn but by using some careful programming techniques, we are able to reduce the proportionality constant.

As pointed out by Berretti and Sokal [6], an efficient way to implement the B–S algorithm in two dimensions is to use a bitmap and a linear linked list. However, a bitmap is not practical in three dimensions since it uses too much memory. This situation would normally require the use of a relatively slow hash table; however, a clever data structure known as a sliding bitmap was introduced by Berretti and Sokal [15] which is about twice as fast as a hash table. This data structure can be employed solely because the algorithm uses only local moves. We found that when using a bitmap, it is expensive to repeatedly calculate the addresses of lattice points. A better approach is to store the addresses of the points of the walk in a linear list and then use these to calculate the addresses of their neighbours. This approach only requires the use of inexpensive addition and subtraction operations. In fact we use this same technique for the hash table which is required when the reptation algorithm is implemented in three dimensions.

When implementing the algorithm for ISAWs an efficient method is required to deal with the problem of calculating Δm for each Monte Carlo step. Since bonds can only be added or deleted from one end of the walk, we found that it was never necessary to calculate Δm when a bond was being deleted. All that is needed is to calculate and store the value of Δm when the ith bond is added onto the end of the walk in an array. When the ith bond is eventually deleted, Δm will simply be the negative of the ith entry in the array. There are no short cuts for calculating Δm when adding a bond onto the walk, but by using the technique described above for calculating the addresses of the neighbours, we minimized the time it takes to do this.

3. The reptation–Metropolis algorithm

The reptation algorithm generates SAWs at fixed n with equal weight, i.e. in a canonical ensemble. It is also very simple to generate ISAWs with this algorithm by including the Metropolis move mentioned previously. Reptation is very similar to the B–S algorithm in that its basic move is to delete one bond at the tail of the walk and simultaneously append another bond at the head of the walk, thus conserving n. This algorithm was known to be non-ergodic by its inventors. In order to make it ergodic, we must add a pair of local/bilocal moves which allow the algorithm to extricate itself from otherwise frozen configurations. A list of these moves is given in [16]. We chose to use the end-kink and kink-end moves in our simulations.

In order to satisfy detailed balance, the end-kink and kink-end moves have to occur with the correct relative probabilities. The ratio of these probabilities should be the ratio of the number of two-step self-avoiding configurations to the number of kink configurations. For example, in two dimensions this ratio is 9:2 (assuming immediate reversals are not attempted) thus

$$\Pr(\text{attempting end-kink}) = \tfrac{2}{9}\Pr(\text{attempting kink-end}). \tag{17}$$

There is no relationship between the probability of attempting a reptation move and the probability of attempting an end-kink or kink-end move. We can therefore tune these probabilities to maximize the efficiency of the algorithm. We will refer to the reptation algorithm with a Metropolis move and ergodic extensions as the reptation–Metropolis (R–M) algorithm.

To improve the R–M algorithm we simply delete the last Δn bonds from the tail of the walk and simultaneously add a mini-walk of length Δn to the head of the walk. 'Simultaneously' here means that the mini-walk added onto the head of the walk is allowed to occupy sites that were previously occupied by the last Δn sites of the walk. The results in employing this move were mixed, as can be seen in section 4. A significant improvement to the overall efficiency of the algorithm was achieved when simulating ordinary SAWs. However, the simulation of ISAWs, especially in the collapsed phase, yielded only marginal gains (40–60%) in the overall efficiency of the algorithm.

The problem of simultaneously adding and deleting bonds from both sides of the walk make the implementation of this algorithm significantly more difficult than the B–S–M algorithm. The added problems of the end-kink and kink-end moves compound this difficulty resulting in a much slower program. Calculating Δm is also expensive since the short cut available for B–S–M is not applicable here. This results in a stronger linear dependence on CPU time with increasing Δn meaning that any improvement in τ is significantly dampened by an increase in CPU time. It is this very problem that leads to the slight gains in efficiency for ISAWs. We were partly able to compensate for these problems by employing the method of multiple Markov chains.

3.1. Multiple Markov chains

We will briefly outline this method here, for further details and a more comprehensive explanation of why the algorithm helps see [3]. This technique consists of running a set of Markov chains in parallel, where the state space of the Markov chains are ISAWs in a canonical ensemble. The idea is to simulate each chain at a different temperature and to regularly attempt to swap two neighbouring chains, i.e. to cool one chain and heat up the other one. In order to perform this swapping, the chains must be ordered in

increasing temperature so that neighbouring chains will be simulated at similar temperatures. The probability with which the swapping move is accepted is chosen to satisfy detailed balance. This probability also ensures that swapping two chains will not throw them out of equilibrium—otherwise the algorithm would be useless. Thus the probability of swapping two chains is reasonably high only when there is a sufficient overlap between the distributions of the neighbouring chains. In practice this means that the temperature difference between neighbouring chains has to be fairly small.

To see why this method reduces the autocorrelation time, consider simulating an ISAW at some temperature† ω. The configuration of the ISAW will change as it normally does for a single Markov chain until a swap move is successful between itself and a neighbouring chain. One of the chains will then spend some time at a higher temperature where equilibriation is rapid, resulting in a different configuration in a shorter time. Since the Markov chain will still be in equilibrium, we can sample from it normally. These swaps should occur often enough to ensure that they have an effect, i.e. at least several attempts should be made during one 'typical' autocorrelation time. However, they should also be sparse enough to allow a noticeable change in the configurations between swaps. The exact frequency of these swaps was determined empirically for the particular set of temperatures being simulated.

3.2. Efficiency of the R–M algorithm

The argument for the efficiency of the reptation algorithm is similar to the argument for the B–S algorithm. After approximately n^2 Monte Carlo moves, the slithering motions of the algorithm will have carried it forward a distance of n steps. Thus, all of the original bonds in the walk will have been replaced by new ones and we will therefore have a new configuration. Once more we would expect this argument to be exact for ordinary random walks although unlike B–S, we are not aware of any proof of this. To quantify the improvement in the autocorrelation time with our bilocal move, simply replace n by $n/\Delta n$ as before.

Most of the programming techniques mentioned above for the B–S algorithm are also applicable for this algorithm. However in three dimensions a hash table is necessary since a bitmap requires too much memory and a sliding bitmap cannot handle bilocal moves. The hash function chosen should not be too complicated so that the addresses of the nearest neighbours of a site can be calculated from the address of the site. Unfortunately the trick used for the B–S algorithm when calculating Δm is not applicable here since bonds can be appended to either end of the walk. The calculation of Δm whenever a reptation move is applied is actually fairly cumbersome and this significantly slows the speed of the program. This is more so in three dimensions where a hash table must be used.

4. Autocorrelation times

In this section we will present the comparison of the algorithms mentioned above with and without the above enhancements. It is first important to note the method we used to estimate the autocorrelation times for the various observables. The quantity we actually estimate is the integrated autocorrelation time. This is done by first estimating the *normalized* autocorrelation function $\rho_{AA}(s) = C_{AA}(s)/C_{AA}(0)$, where $C_{AA}(s) \equiv \langle A_t A_{t+s}\rangle - \mu_A^2$, for some observable A. The natural estimator of $C_{AA}(s)$ for a time series with N samples of

† Even though ω is not the actual temperature (it is a dimensionless parameter that *controls* the temperature) we will refer to it as such for convenience.

some observable A is given by

$$\hat{C}_{AA}(s) = \frac{1}{N-|s|} \sum_{t=1}^{N-|s|} (A_t - \bar{A})(A_{t+|s|} - \bar{A}). \tag{18}$$

Theoretically $\tau_{\mathrm{int},A}$ is the area under $\rho_{AA}(s)$, so we can estimate it by summing the area under $\hat{\rho}_{AA}(s) = \hat{C}_{AA}(s)/\hat{C}_{AA}(0)$. This is fine for small s, however as s gets large, $\rho_{AA}(s)$ typically decays exponentially to zero and the estimates of $\rho_{AA}(s)$ for large s are mostly noise. The question then arises as to where the sum should be cut off. In [17], Madras and Sokal suggest

$$\hat{\tau}^{(M)}_{\mathrm{int},A} = 0.5 + \sum_{k=1}^{M} \rho_{AA}(k) \tag{19}$$

where $M = 10\hat{\tau}^{(M)}_{\mathrm{int},A}$. The factor of 10 is chosen arbitrarily and is there to ensure that contributions are made from several $\tau_{\mathrm{int},A}$'s apart.

Finally, please note that all error bars appearing in this section are 95% confidence intervals.

4.1. The B–S–M algorithm

We start off by demonstrating the gains in the autocorrelation time for the B–S–M algorithm. The trial runs were performed on the square lattice at three temperatures and were chosen to illustrate the advantages and limitations of the new local move. The results are summarized in three tables which show the integrated autocorrelation time for the observable n. All the simulations were performed on a Dec Alpha 250 4/266. The tables show the average time taken to execute a Monte Carlo step in microseconds, t_{CPU}, the time, in seconds, needed to produce an independent configuration, $t_{\mathrm{IND}} = 2\tau_{\mathrm{int},n} t_{\mathrm{CPU}}$ and the gain in efficiency E, where E is defined as

$$E(\Delta n) = \frac{t_{\mathrm{IND}}(\Delta n = 1)}{t_{\mathrm{IND}}(\Delta n)}. \tag{20}$$

Table 1 shows the results obtained when simulating ordinary SAWs, i.e. $\omega = 1$. This is where the new local move works best since fewer null transitions occur at this temperature. The best result is obtained by choosing $\Delta n = 16$ (we did not go beyond $\Delta n = 16$ due to memory constraints) and this increases the efficiency of the algorithm by a factor of 38.8.

Table 1. Autocorrelation times, $\tau_{\mathrm{int},n}$, for SAWs with $\langle n \rangle \approx 165$ on the square lattice for a variety of Δn values using the B–S–M algorithm

Δn	Monte Carlo steps	$\tau_{\mathrm{int},n}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
1	2×10^{10}	$110\,000 \pm 3300$	1.05	0.231	1.0 ± 0.03
2	3×10^{9}	$31\,600 \pm 1400$	1.28	0.081	2.9 ± 0.2
4	5×10^{8}	$7\,760 \pm 390$	1.93	0.030	7.7 ± 0.4
6	5×10^{8}	$3\,900 \pm 140$	2.40	0.019	12.3 ± 0.6
8	2×10^{8}	$2\,240 \pm 100$	2.87	0.013	18.0 ± 1.0
10	2×10^{8}	$1\,540 \pm 60$	3.30	0.010	22.7 ± 1.1
12	1×10^{8}	$1\,100 \pm 50$	3.87	8.5×10^{-3}	27.1 ± 1.5
14	1×10^{8}	820 ± 30	4.25	7.0×10^{-3}	33.1 ± 1.6
16	1×10^{8}	620 ± 20	4.80	6.0×10^{-3}	38.8 ± 1.7

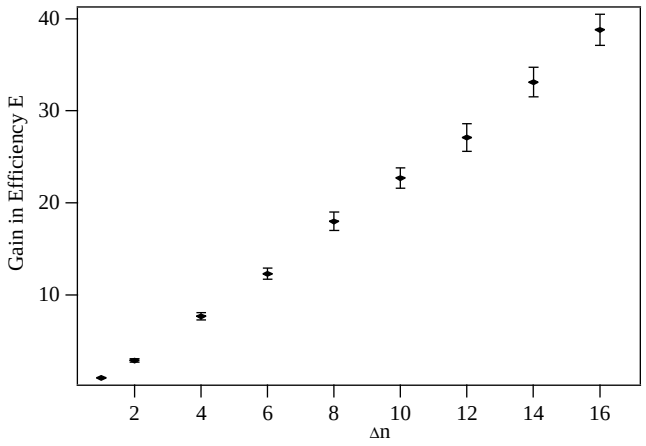

Figure 1. E as a function of Δn for the enhanced B–S algorithm.

Table 2. Autocorrelation times, $\tau_{\mathrm{int},n}$, at $\omega = 1.6$ with $\langle n \rangle \approx 165$ on the square lattice for a variety of Δn values using the B–S–M algorithm.

Δn	Monte Carlo steps	$\tau_{\mathrm{int},n}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
1	1×10^{10}	$120\,000 \pm 5300$	1.16	0.278	1.0 ± 0.04
2	3×10^{9}	$36\,400 \pm 1700$	1.66	0.121	2.3 ± 0.2
4	2×10^{9}	$14\,100 \pm 500$	2.10	0.059	4.7 ± 0.3
6	1×10^{9}	$9\,700 \pm 300$	2.45	0.048	5.9 ± 0.3
8	1×10^{9}	$8\,800 \pm 200$	2.64	0.046	6.0 ± 0.3
10	1×10^{9}	$8\,500 \pm 200$	2.81	0.048	5.8 ± 0.3
12	1×10^{9}	$9\,400 \pm 300$	2.90	0.055	5.1 ± 0.3
14	1×10^{9}	$11\,600 \pm 500$	3.05	0.071	3.9 ± 0.2

The behaviour of $\tau_{\mathrm{int},n}$ is shown by performing a log–log plot of $\tau_{\mathrm{int},n}$ versus Δn. The data formed a straight line whose slope was estimated by the linear least squares method. The slope was found to be -1.86 ± 0.02, significantly away from the random walk result of -2. We also performed fits for t_{CPU} and E as functions of Δn. The behaviour of t_{CPU} was linear and the regression equation was found to be $t_{\mathrm{CPU}} = 0.25\Delta n + 0.86$.

Figure 1 shows a plot of E versus Δn. For small Δn it is expected that $E \sim \Delta n^{1.86}$, whilst for large Δn it is expected that $E \sim \Delta n^{0.86}$. However, for the range of Δn in these simulations the behaviour of E is intermediate between these two asymptotic regions and thus neither asymptotic behaviour is evident in the figure.

Table 2 shows the results obtained for ISAWs at $\omega = 1.6$. An important point to note in table 2 is that $\tau_{\mathrm{int},n}$ is not monotonically decreasing with Δn. The best result occurs for $\Delta n = 8$ where we see an improvement by a factor of 6.0. This occurs when the acceptance fraction of a Monte Carlo step outweighs the larger conformational change obtained for increasing Δn. Unlike the SAW result, the plot of t_{CPU} versus Δn is not linear since many attempts to append mini-walks violate self-avoidance and hence the expensive operation of calculating Δm is not performed as often.

Table 3 shows the efficiency of the algorithm in the collapsed phase of the model where $\omega = 2.05$. The trends that were present in table 2 are magnified in table 3. We see that the

Table 3. Autocorrelation times, $\tau_{\mathrm{int},n}$, at $\omega = 2.05$ with $\langle n \rangle \approx 200$ on the square lattice for a variety of Δn values using the B–S–M algorithm.

Δn	Monte Carlo steps	$\tau_{\mathrm{int},n}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
1	8×10^{10}	$1\,100\,000 \pm 100\,000$	1.16	2.55	1.0 ± 0.1
2	2.5×10^{10}	$460\,000 \pm 50\,000$	1.55	1.43	1.8 ± 0.3
4	2×10^{10}	$410\,000 \pm 50\,000$	1.77	1.45	1.8 ± 0.3
6	2.5×10^{10}	$770\,000 \pm 110\,000$	1.76	2.71	0.9 ± 0.2

Table 4. Autocorrelation times, $\tau_{\mathrm{int},R_n^2}$, for SAWs with $n = 500$ on the square lattice for a variety of Δn values using the R–M algorithm

Δn	Monte Carlo steps	$\tau_{\mathrm{int},R_n^2}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
1	5×10^9	$56\,700 \pm 2500$	1.77	0.200	1 ± 0.04
2	2×10^9	$19\,800 \pm 800$	2.32	0.092	2.2 ± 0.1
4	1×10^9	$6\,600 \pm 200$	2.88	0.038	5.3 ± 0.3
6	5×10^8	$3\,400 \pm 100$	3.65	0.025	8.1 ± 0.4
8	2×10^8	$2\,000 \pm 80$	4.27	0.017	11.7 ± 0.7
10	2×10^8	$1\,500 \pm 50$	4.89	0.016	13.7 ± 0.8
12	2×10^8	$1\,150 \pm 40$	5.42	0.012	16.1 ± 0.9
14	2×10^8	910 ± 30	5.96	0.011	18.5 ± 1.0
16	2×10^8	780 ± 20	6.56	0.010	19.6 ± 1.0

optimal value now occurs at $\Delta n = 2$ with a gain in efficiency of only 1.8. Another striking feature in this table is the magnitude of the autocorrelation times. For $\omega = 1.6$, about 20 independent configurations could be generated in 1 s whereas now more than 1 s is needed to generate an independent configuration with a slightly larger average length. This fact severely limits our sample size and results in significantly poorer statistics in the collapsed regime.

4.2. The reptation–Metropolis algorithm

We performed trial runs at three temperatures on the square lattice and at two different temperatures on the simple cubic lattice for the R–M algorithm. We also used two multiple Markov on both the square and cubic lattices, both employing 12 temperatures. In both cases one of the multiple Markov chains used $\Delta n = 1$ and the other used $\Delta n > 1$. This allowed us to distinguish between the effect of using a multiple Markov chain, and using $\Delta n > 1$. The results for these trial runs are summarized in three tables.

Table 4 shows the results obtained when simulating ordinary SAWs with the reptation algorithm for different Δn values. The results are qualitatively the same as for the B–S algorithm, i.e. increasing Δn increases the efficiency of the algorithm. We performed a log–log plot for the autocorrelation time for the squared end-to-end distance, $\tau_{\mathrm{int},R_n^2}$ versus Δn to check our argument for the behaviour of $\tau_{\mathrm{int},R_n^2}$. A straight line fit to the data yielded a slope of -1.56 ± 0.03 which was worse than the result obtained for the B–S algorithm.

Figure 2 shows a plot of E against Δn. We expect $E \sim \Delta n^{1.56}$ for small Δn and $E \sim \Delta n^{0.56}$ for large Δn. However, as for the B–S algorithm, the range of Δn used in the simulations are between these two asymptotic regimes. Thus, none of the behaviour

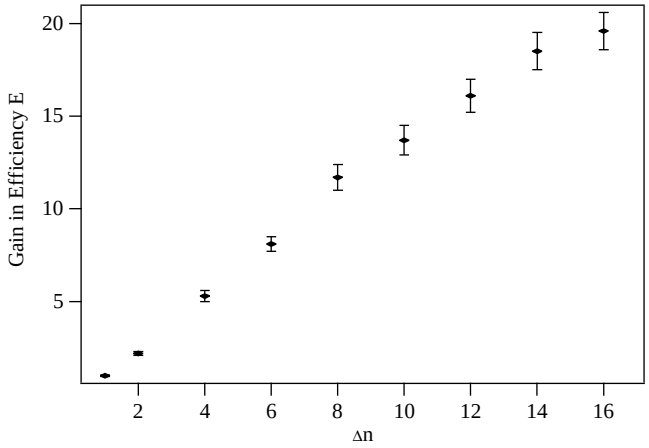

Figure 2. E as a function of Δn for the enhanced reptation algorithm.

Table 5. Autocorrelation times, $\tau_{\mathrm{int},m}$, for ISAWs with $n = 500$ on the square lattice using the R–M algorithm with multiple Markov chains.

ω	Δn	Monte Carlo steps	$\tau_{\mathrm{int},m}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
No multiple Markov chains						
1.8	1	5×10^9	$55\,700 \pm 2400$	2.36	0.262	1.0 ± 0.04
2.1	1	1×10^{10}	$101\,000 \pm 4000$	2.21	0.446	1.0 ± 0.04
1.8	6	2×10^9	$18\,900 \pm 700$	4.76	0.180	1.5 ± 0.1
2.1	5	5×10^9	$64\,000 \pm 6000$	3.17	0.406	1.1 ± 0.1
Multiple Markov chains						
1.8	1	1.5×10^9	$27\,400 \pm 1500$	3.07	0.168	1.6 ± 0.1
2.1	1	1.5×10^9	$24\,900 \pm 1300$	2.87	0.143	3.1 ± 0.2
1.8	6	1×10^9	$13\,800 \pm 700$	6.18	0.171	1.5 ± 0.2
2.1	5	1×10^9	$18\,500 \pm 1000$	4.12	0.152	2.9 ± 0.3

described above for E as a function of Δn is evident in the plot.

In carrying out the simulations involving multiple Markov chains, we followed the suggestions made in [3] for deciding on the number of chains to be used and on their temperatures. During the trial runs, we kept track of the number of successful swaps between all pairs of neighbouring chains. We also kept track of the fraction of time each chain spent at each temperature. In our final runs, these two diagnostics indicated that the chains were mixing well and that the frequency of successful swapping between neighbouring chains was satisfactory.

Table 5 shows the results for simulating ISAWs in two dimensions with and without multiple Markov chains. We simulated at 12 temperatures, namely $\omega = 1.4$, 1.5, 1.6, 1.7, 1.8, 1.9, 1.95, 2.0, 2.05, 2.1, 2.15 and 2.2, however, we only present the results for $\omega = 1.8$ and 2.1. As is apparent from the efficiency column, using $\Delta n > 1$ does not provide a great increase in performance, except for the case $\omega = 1.8$ with no multiple Markov chains. The use of multiple Markov chains make the algorithm roughly three times as efficient in the low temperature phase. We found that there is a CPU overhead of approximately 30% when

Table 6. Autocorrelation times, $\tau_{\mathrm{int},m}$, for ISAWs with $n = 500$ on the cubic lattice using the R–M algorithm with multiple Markov chains.

ω	Δn	Monte Carlo steps	$\tau_{\mathrm{int},m}$	t_{CPU} (μs)	t_{IND} (s)	Efficiency E
No multiple Markov chains						
1.3	1	3×10^9	$33\,000 \pm 1400$	10.4	0.686	1.0 ± 0.04
1.45	1	6×10^9	$66\,000 \pm 2800$	8.3	1.096	1.0 ± 0.04
1.3	6	5×10^8	$4\,300 \pm 200$	20.4	0.175	3.9 ± 0.3
1.45	6	3×10^9	$22\,000 \pm 800$	11.6	0.510	2.1 ± 0.1
Multiple Markov chains						
1.3	1	1.05×10^9	$16\,200 \pm 800$	13.5	0.437	1.6 ± 0.1
1.45	1	1.05×10^9	$22\,400 \pm 1300$	10.8	0.484	2.3 ± 0.2
1.3	6	5×10^8	$3\,800 \pm 150$	26.5	0.201	3.4 ± 0.3
1.45	6	5×10^8	$8\,450 \pm 450$	15.1	0.255	4.3 ± 0.4

using multiple Markov chains. This rather large overhead arises from the need to replace single variables with arrays. Using $\Delta n > 1$ together with multiple Markov chains in two dimensions does not seem to increase the performance of the algorithm.

Table 6 shows the results for using the algorithm on ISAWs in three dimensions. The multiple Markov chains employed 12 temperatures: $\omega = 1.1$, 1.15, 1.2, 1.25, 1.3, 1.35, 1.375, 1.4, 1.425, 1.45, 1.475 and 1.5. The results are better than those observed in two dimensions. However, this is not surprising since the self-avoidance constraint is not as severe in three dimensions. We also see that (unlike two dimensions) the use of multiple Markov chains together with $\Delta n > 1$ achieves an increase in efficiency in the low temperature phase. The larger CPU times per Monte Carlo step in three dimensions are due to the use of hashing rather than using a bitmap. This is partly compensated for by the smaller autocorrelation times that were observed around the effective critical point (for $n = 500$, the effective critical point occurs at $\omega \approx 1.4$).

5. Conclusion

We have added a Metropolis move as well as a new local move to the B–S and reptation algorithms to simulate ISAWs in two and three dimensions. The use of the local move for simulating ordinary SAWs greatly increased the efficiency—by an order of magnitude—of both algorithms. There is also a substantial gain in efficiency even when interactions are introduced and simulations are carried out in the low temperature phase or at the critical point. To further improve the performance of the reptation algorithm at low temperatures, we introduced a multiple Markov chain. This also increased the efficiency in both two and three dimensions.

Acknowledgments

We would like to thank A J Guttmann and A Owczarek for carefully reading the manuscript. One of us (RB) would like to thank the Australian Research Council for financial assistance and PN would like to thank the School of Graduate Studies at the University of Melbourne for an Australian Postgraduate Award.

References

[1] Grassberger P and Hegger R 1995 *J. Physique* I **5** 597
[2] Grassberger P and Hegger R 1995 *J. Chem. Phys.* **102** 6881
[3] Tesi M C, Janse van Rensburg E J, Orlandini E and Whittington S G 1996 *J. Stat. Phys.* **82** 155
[4] Chang I and Meirovitch H 1993 *Phys. Rev.* E **48** 3656
[5] Li B, Madras N and Sokal A D 1995 *J. Stat. Phys.* **80** 661
[6] Berretti A and Sokal A D 1985 *J. Stat. Phys.* **40** 485
[7] Kron A K, Ptitsyn O B, Skvortsov A M and Fedorov A K 1967 *Molek. Biol.* **1** 576 (Engl. transl. 1967 *Molec. Biol.* **1** 487)
[8] Wall F T and Mandel F 1975 *J. Chem. Phys.* **63** 4592
[9] Reiter J 1990 *Macromolecules* **23** 3811
[10] Nidras P P *J. Phys. A: Math. Gen.* to appear
[11] Owczarek A L, Prellberg T and Brak R 1993 *Phys. Rev. Lett.* **70** 951
[12] Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H and Teller E J 1953 *J. Chem. Phys.* **21** 1087
[13] Wall F T, Rubin R J and Isaacson L M 1957 *J. Chem. Phys.* **27** 186
[14] Sokal A D and Thomas L E 1989 *J. Stat. Phys.* **54** 797
[15] Berretti A and Sokal A D 1990 *Comput. Phys. Commun.* **58** 1
[16] Sokal A D 1995 *Monte Carlo and Molecular Dynamics Simulations in Polymer Science* ed K Binder (New York: Oxford University Press) pp 47–124
[17] Madras N and Sokal A D 1987 *J. Stat. Phys.* **50** 109