

The number c_n is generally difficult to determine, even for modest values of n . In two dimensions $c_0 = 1$, $c_1 = 4$, $c_2 = 12$, $c_3 = 36$ while $c_4 = 100$, and c_n is known to $n = 71$ [24]. In three dimensions $c_0 = 1$, $c_1 = 6$, $c_2 = 30$ and $c_3 = 150$ while c_n is known to $n = 30$ [4]. It is the case that

$$(1.1) \quad d^n \leq c_n \leq 2d(2d-1)^{n-1},$$

so that c_n grows exponentially with n .

Determining c_n is generally a formidable numerical challenge, and only sophisticated ideas and programming, and improvements in computing power following Moore's law, made possible the determination of c_n in references [24] and [4].

A few more facts are known about c_n . For example, $c_{n+1} \geq c_n$, so that c_n is monotonic for all values of n [35]. By cutting a self-avoiding walk of length $n+m$ in its vertex v_n , two subwalks of lengths n and m are obtained. The number of choices of a walk of length $n+m$ is c_{n+m} , but the number of resulting pairs of subwalks is at most $c_n c_m$. Thus, c_n is a submultiplicative function on \mathbb{N} : $c_{n+m} \leq c_n c_m$. Together with the bounds in equation (1.1), this implies that the limit

$$(1.2) \quad \mu = \lim_{n \rightarrow \infty} [c_n]^{1/n}$$

exists and $d \leq \mu \leq (2d-1)$ [16, 13, 15].

Kesten's pattern theorem [27, 28] shows that the limit

$$(1.3) \quad \lim_{n \rightarrow \infty} \frac{c_{n+2}}{c_n} = \mu^2$$

exists, but it is not known that the limit $\lim_{n \rightarrow \infty} [c_{n+1}/c_n] = \mu$ exists.

Lattice self-avoiding walks which return to the origin are *lattice polygons*. Such polygons have a root vertex at the origin, but normally one counts polygons up to equivalences under translations in the lattice. This removes the root, and p_n is defined as the number of (unrooted) polygons of length n steps in the lattice \mathbb{Z}^d . For example, $p_4 = 1$, $p_6 = 2$, $p_8 = 7$ in the square lattice, and series data exist for p_n for all $n \leq 110$ [26, 23].

It is a theorem [14] that the limit

$$(1.4) \quad \mu = \lim_{n \rightarrow \infty} [p_{2n}]^{1/2n}$$

exists (it is taken through even integers in bipartite lattices such as the hypercubic lattice, since $p_n = 0$ for odd values of n in such lattices). The value of the limit is equal to μ , the growth constant of walks which is defined in equation (1.2) (see reference [14]). It is known that the limit

$$(1.5) \quad \lim_{n \rightarrow \infty} \frac{p_{n+2}}{p_n} = \mu^2$$

exists, a result which is due to Kesten [27, 28]; see reference [31] for a simpler proof.

1.1. Scaling of c_n . Scaling arguments presume that c_n exhibits scaling: At the most basic level, the *susceptibility of the self-avoiding walk* $\chi(t)$ defined by

$$(1.6) \quad \chi(t) = \sum_{n=0}^{\infty} c_n t^n$$

and has a singularity at $t = t_c = 1/\mu$ which to leading order should be given by the singular part of $\chi(t)$ which is assumed to diverge as an inverse fractional power of

$|t - t_c|$:

$$(1.7) \quad \chi(t) \sim |t - t_c|^{-\gamma}.$$

This shows that a reasonable assumption for leading order behaviour of c_n is

$$(1.8) \quad c_n \sim A n^{\gamma-1} \mu^n.$$

In other words, $[c_n/\mu^n] \sim A n^{\gamma-1}$, and this scaling should be universal (and independent of the lattice).

In high dimensions there may be enough space for the self-avoiding condition to be local and of no consequence in the scaling of the walk. In that case one expects $\gamma = 1$. This is the case in $d > 4$ dimensions (see for example [31]). We say that the *entropic exponent* γ has mean field value $\gamma = 1$ (this is also the random walk value, since the number of random walks of length n is $(2d)^n$), and the mean field value can be computed from a free field theory in high dimensions. See references [17, 18].

Dimensions $d = 4$ is the upper critical dimension of this model. In 4-dimensional lattices, $\gamma = 1$, but equation (1.8) is modified by a logarithmic correction to $c_n \sim A |\log n|^{1/4} \mu^n$.

The calculation of μ and γ from data collected on c_n is a major motivation for enumerating c_n . In addition to the exact computation of c_n , Monte Carlo methods have been used to determine μ and γ . These efforts are not nearly as accurate as exact enumeration studies, but they do allow us to verify series enumeration results independently.

1.2. Computing μ and γ . The growth constant μ can be estimated from the numerical values of c_n . In low dimensions such series data give the best estimates for μ : In \mathbb{Z}^2 it is known that

$$(1.9) \quad \mu = 2.63815856 \pm 0.00000003.$$

as determined in reference [24], see references [25, 12] for additional results.

Monte Carlo estimates of μ and γ can be obtained from grand canonical simulations of self-avoiding walks. These simulations sample walks from an invariant limiting distribution related to $\chi(t)$: The probability to sample a state of length n is given by $(c_n t^n)/\chi(t)$. By examining the distribution of walks obtained, μ and γ can be estimated. This was in particular done with the Beretti-Sokal algorithm [1] to obtain $\mu = 2.63820 \pm 0.00034$ and $\gamma = 1.352 \pm 0.031$ in the square lattice. See also reference [33] and generalisation of this algorithm in reference [34]. In reference [33] it is reported that

$$(1.10) \quad \mu = 2.638164 \pm 0.000014.$$

Less precise estimates for μ are available in the cubic lattice \mathbb{Z}^3 , see reference [4] where it is estimated that

$$(1.11) \quad \mu = 4.684043 \pm 0.000012$$

from series data on walks obtained by the lace expansion.

Series analysis for polygons [26] gives μ in two dimensions to very high precision:

$$(1.12) \quad \mu = 2.63815853034 \pm 0.00000000010.$$

Reference [23] can be consulted for a slight improvement of this estimate.

Collecting atmospheric statistics on polygons with canonical Monte Carlo simulations showed that

$$(1.13) \quad \mu = 4.68398 \pm 0.00016$$

in the cubic lattice [21]. Reference [19] reports an (unpublished) estimate $\mu = 4.683907 \pm 0.000022$ due to A.J. Guttmann.

In two and three dimensions $\gamma > 1$. In two dimensions conformal field theory gives the exact value $\gamma = 43/32$ [5] while numerical simulations by Monte Carlo algorithms were used to estimate that $\gamma = 1.1575(6)$ in three dimensions [30].

The best estimates for γ are due to exact enumeration data in two dimensions and Monte Carlo simulations in three dimensions:

$$(1.14) \quad \gamma = \begin{cases} 1.343745 \pm 0.000015, & \text{if } d = 2, [24]; \\ 1.1575 \pm 0.0006, & \text{if } d = 3, [2]. \end{cases}$$

See reference [4] for more on results in $d = 3$, and the very accurate estimate $\gamma = 1.1573 \pm 0.00032$ in three dimensions was obtained in reference [20] by using the PERM algorithm on the Domb-Joyce model.

In this paper we review algorithms and general ideas underlying the approximate enumeration of self-avoiding walks. The invention of the Rosenbluth algorithm in 1955 [38] provided the first means for estimating c_n . This algorithm was generalised to PERM [11] and later to GARM [37]. A further generalisation of GARM to GAS [22] is recent, and we will consider each of these implementations briefly to demonstrate the approximation enumeration of self-avoiding walks. Many of the ideas underlying these algorithms are quite general, and is applicable to other objects, such as finite groups [7] and other lattice objects such as self-avoiding polygons [37].

2. APPROXIMATE COUNTING OF SELF-AVOIDING WALKS BY ROSENBLUTH SAMPLING

Rosenbluth sampling of self-avoiding walks generates walks s_n of weights W_n such that the mean value of the weights $\langle W_n \rangle$ is a direct estimate of c_n .

The implementation is as follows. Put $W_0 = 1$ and s_0 is the trivial walk of length zero starting in the origin. If s_n and W_n are known, then determine the number of open lattice sites which are nearest neighbour to the endpoint of s_n ; let σ_n be this number. With uniform probability choose one of these open lattice sites and append it to s_n to obtain s_{n+1} , and update the weight $W_{n+1} = \sigma_n W_n$. Repeat this process to grow a walk of a desired length. A realisation of a walk by the algorithm is illustrated in figure 2.

There is a possibility that $\sigma_n = 0$ when the endpoint of the walk has no unoccupied lattice sites adjacent for the next step. In that case we say that the walk is trapped, and the attempt to grow a walk is stopped. The algorithm will continue by discarding the failed attempt by assigning weight zero to all subsequent walks in this attempt. It will then attempt to grow a new walk starting again with the trivial walk.

If N started walks are grown to a target length n , then some number $M \leq N$ is completed, giving a sample of M walks s each with an associated weigh W_s given

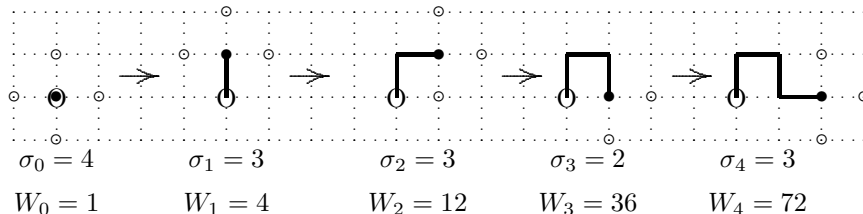


FIGURE 2. Rosenbluth sampling of walks. The algorithm starts with the trivial walk at the origin on the left, and then finds unoccupied nearest neighbour vertices (denoted by \circ 's) to the endpoint of the walk. These are added to the walk while the weight W_n is updated as shown.

by

$$(2.1) \quad W(s) = \prod_{i=0}^{n-1} \sigma_i(s).$$

Started walks which were trapped should still be considered part of the ensemble of grown walks, but with weight equal to zero. This is important for analysing the data produced by the algorithm.

The probability that a given walk s of weight $W(s)$ and length n is grown by the algorithm is given by

$$(2.2) \quad \text{Pr}(s) = \prod_{i=0}^{n-1} \left[\frac{1}{\sigma_i(s)} \right] = \frac{1}{W(s)}.$$

The mean value of the weights of walks of length n is

$$(2.3) \quad \langle W \rangle_n = \sum_{s \in S_n} \text{Pr}(s) W(s) = \sum_{s \in S_n} 1 = c_n,$$

where S_n is the collection of all self-avoiding walks of length n . In other words, by determining the average weights of a set of walks grown by the algorithm, we get an estimate of c_n . In particular, if N walks were started, then the average weight

$$(2.4) \quad \overline{W}_n = \frac{1}{N} \sum_{i=1}^N W(s_i) \rightarrow \langle W \rangle_n = c_n$$

as $N \rightarrow \infty$, and where walks that were lost to attrition have been assigned zero weight.

Typically the algorithm is implemented by two parameters: the first is the number of starts N , and the length of each grown walk n . Each started walk is not guaranteed to be completed to target length n ; the attempt may fail if a trapped conformation with $\sigma_m = 0$ is encountered. For short walks this is not a serious problem, but for longer walks this attrition makes the algorithm ineffective. In the square lattice this attrition of started walks is mild for walks of lengths less than about 50 steps, but becomes a serious problem for walks of lengths 100 steps or longer.

The weights W_n of started walks also disperse over many orders of magnitude as the lengths of the walks increase. Eventually, the entire generated sample of walks is dominated statistically by a few walks with very large weights, and the

quality of data deteriorates. In the square lattice this becomes a problem for values of n approaching about 100 edges. Together with the attrition of started walks, the dispersion of weights makes efficient sampling of longer walks impractical if not impossible. In figure 3 the attrition of started walks in the Rosenbluth algorithm is illustrated.

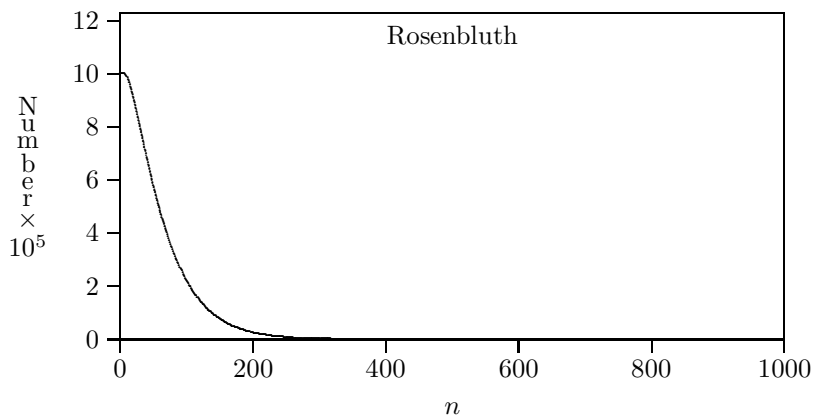


FIGURE 3. Attrition of started walks in the Rosenbluth algorithm in the two dimensional square lattice. A million started walks have almost all been discarded by attrition after 200 steps. Only about 20% of the started walks survived to 100 steps. In addition, the dispersion of weights increases rapidly with n , making the algorithm inefficient for walks longer than about 100 steps.

Estimates of c_n by the Rosenbluth method in the square lattice is given in table 1. A million started walks were generated, and the average weights were computed at each value of n . For small values of n the results are good, but deteriorates as more walks are lost to attrition, and as the weights disperse with increasing values of n . Over the range of values of n displayed in this table, the Rosenbluth approximation is good, and in figure 3 we note that more than 50% of started walks survives to lengths of $n = 70$. The quality of the estimates decreases for $n > 100$, as attrition of the walks, and the dispersion of weights, take their toll.

3. PRUNED ENRICHED ROSENBLUTH SAMPLING (PERM)

The two basic flaws in Rosenbluth sampling are (1) the attrition of trapped walks, and (2) the increasing dispersion of weights with increasing length in the surviving walks. The increasing dispersion eventually leads to the situation that a few walks with very large weights dominate the entire set of sampled walks. These challenges can be overcome simultaneously by the addition of pruning and enriching steps to the algorithm [11].

PERM adapts Rosenbluth sampling in such a way that both flaws are simultaneously addressed. Walks with large weights are enriched (see reference [39]) in the simulation, at the same time having their individual weights reduced. Walks with low weights are pruned. These measures reduce the dispersion of the weights, and

by enrichment increase the number of longer walks in the simulation. The addition of enrichment and pruning steps in the Rosenbluth algorithm gives PERM, an acronym for “Pruned and Enriched Rosenbluth Method”.

Suppose that the weight of a walk s of length n sampled by the Rosenbluth method is given by

$$(3.1) \quad W(s) = \prod_{i=1}^{n-1} \sigma_i.$$

Introduce a cut-off weight or threshold T_n at length n , and if $W(s) > T_n$, then enrich the sample by adding a copy of s to it, while at the same time reducing the weight $W(s)$ by a factor of two. In other words, there are now two copies of s , but each with a weight of $W(s)/2$. While this enrichment does not affect the sample

n	c_n	Rosenbluth $\langle W_n \rangle$	flatPERM $\langle W_n \rangle$
0	1	1	1
1	4	4	4
2	12	12	12
3	36	36.0023	35.9999
4	100	100.005	99.9670
5	284	283.969	284.024
6	780	780.187	780.085
7	2172	2173.69	2172.49
8	5916	5918.86	5921.01
9	16268	16275.1	16279.4
10	44100	44098.6	44131.9
11	120292	120344	120370
12	324932	325039	325293
13	881500	881624	882273
14	2374444	2.37473×10^6	2.37759×10^6
15	6416596	6.41677×10^6	$6.42589e \times 10^6$
16	17245332	1.72457×10^7	1.72694×10^7
17	46466676	4.64554×10^7	4.65682×10^7
18	124658732	1.24646×10^8	1.24896×10^8
19	335116620	3.35045×10^8	3.35598×10^8
20	897697164	8.97597×10^8	8.98656×10^8
25	123481354908	1.23661×10^{11}	1.23552×10^{11}
30	16741957935348	1.67595×10^{13}	1.67569×10^{13}
35	2252534077759844	2.2564×10^{15}	2.25608×10^{15}
40	300798249248474268	3.01374×10^{17}	3.01558×10^{17}
45	39992704986620915140	4.00832×10^{19}	4.01102×10^{19}
50	5292794668724837206644	5.30379×10^{21}	5.31268×10^{21}
55	698501700277581954674604	6.99913×10^{23}	7.01539×10^{23}
60	91895836025056214634047716	9.2439×10^{25}	9.21359×10^{25}
65	12066271136346725726547810652	1.21868×10^{28}	1.20728×10^{28}
70	1580784678250571882017480243636	1.58563×10^{30}	1.58013×10^{30}
71	4190893020903935054619120005916	4.20159×10^{30}	4.18935×10^{30}

TABLE 1. Approximate Enumeration (Rosenbluth and flatPERM).

average at n , continued growth of the walks from this value of n will result in the copies growing along different trajectories, and their enrichment has the effect of reducing the dispersion of the weights by replacing large weights with reduced sized weights.

The problem of walks with small weights is dealt with by introducing a lower threshold t_n at length n . If a walk s is grown by the Rosenbluth algorithm to length n and of weight $W(s) < t_n$, then the walk is killed (its growth is stopped) with a probability $1/q$ where q is a parameter of the algorithm. If the walk is not killed (with probability $1 - 1/q$), then its weight is increased by a factor of q . Normally, one could put $q \approx 2$.

Sample averages are computed as before. If N is the number of started walks, then sample averages are computed over the entire sample of pruned and enriched conformations, where the weights of pruned or trapped walks are equal to zero.

The algorithm is implemented similar to the Rosenbluth algorithm by tracking started walks s by length n and weight W_n . The algorithm is best implemented with a recursive routine calling itself [11] to complete started and enriched walks to a preset length.

The thresholds t_n and T_n can be changed during the simulation: normally the initial values are $t_n = 0$ and T_n some large constant, and the ratio of these are eventually set to that $T_n/t_n \approx 10$.

Suppose that a collection of N started walks s_j of length n were sampled. Then the average weight is

$$(3.2) \quad \langle W_n \rangle = \frac{1}{N} \sum_{j=1}^N W(s_j) \rightarrow c_n.$$

The thresholds t_n and T_n are chosen by

$$(3.3) \quad t_n = c \langle W_n \rangle, \quad \text{and} \quad T_n = C \langle W_n \rangle$$

where one puts $C/c = 10$. This implementation reduces the dispersion of the weights of the walks to about one order of magnitude, by either pruning walks with relative small weights (including trapped walks), or enriching walks with large weights in the sample. The enrichment process increases the number of completed walks, reducing attrition of the underlying Rosenbluth dynamics significantly - these factors are a significant advance in the approximate enumeration of self-avoiding walks, see reference [11] for more details.

PERM is best implemented in a slightly different incarnation called flatPERM, where enrichment and pruning are tuned to produce a flat histogram of sampled states over the lengths of the walks.

3.1. Flat-histogram PERM (flatPERM). The average weight $\langle W_n \rangle$ of walks of length n in the Rosenbluth algorithm is an estimate of c_n . If the variance of this estimate can be reduced, then one should be able to compute improved estimates of c_n . A main source of the increased uncertainties in estimates of c_n with increasing n is the dispersion of the weights of completed walks. PERM reduces this dispersion by pruning and enrichment, while it also increases the number of completed walks to give a larger collection of sampled walks.

A further refinement in PERM can be achieved by continually favouring walks of large weights in the enrichment process. That is, partially grown walks with weights larger than the average are enriched in PERM at every step. This can

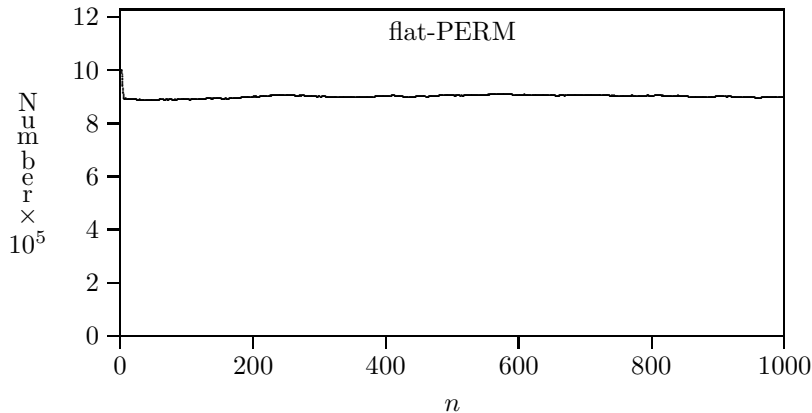


FIGURE 4. Attrition of started walks in the flatPERM algorithm. A virtually constant number of walks is obtained after initial attrition due to thermalization of the algorithm. In this simulations a million walks were started, and the number of surviving walks at each value of n is plotted above. These results should be compared to the Rosenbluth data in figure 3. The nearly constant value of the number of completed walks after an initial decrease for small values of n shows that flatPERM is highly effective in self-tuning to produce a flat histogram over the lengths of completed walks.

be achieved as follows: Let $\langle W_n \rangle_m$ be the estimate of c_n after m walks have been sampled. If the m -th walk has weight W_m , then compute the ratio

$$(3.4) \quad r = \frac{W_m}{\langle W_n \rangle_m},$$

and observe that W_m is also included in the calculation of $\langle W_n \rangle_m$.

If $r > 1$ then the N -th walk has a larger than expected weight, and it should be enriched. If r is also larger than the number of possible ways of extending the current walk ($r > \sigma_n$) then put $c = \min\{[r], \sigma_n\}$. The walk is then enriched in the sample by making c copies of it, and then by reducing its weight by a factor of c .

On the other hand, if $r < 1$, then the m -th walk has weight smaller than expected, and it can be pruned with probability r . If it is kept (not pruned), then its weight is increased by the factor $1/r$.

Observe that the pruning and enriching is done *after* the current walk is included in computing the average $\langle W_n \rangle_m$. The estimate $\langle W_n \rangle_m$ for c_n is initially very wrong, but it improves with increasing m . In this implementation the number of states (walks) at each length n is roughly constant - this gives a flat histogram of the number of walks sampled at each value of n .

Implementation of flatPERM produces a roughly constant number of walks at each value of n : for each walk pruned on average, one walk is enriched. This gives a significant improvement over Rosenbluth sampling, where attrition makes the sampling of long walks very difficult. In figure 4 the attrition of walks in flatPERM sampling is measured for a million started walks in the square lattice, with maximum possible length set to 1000. In this simulation the attrition is less than 10% even for $n = 1000$ and the histogram is more or less flat over the entire

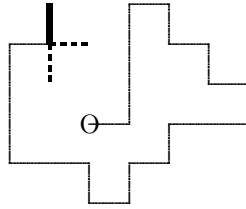


FIGURE 6. The two dashed edges compose the neutral endpoint atmosphere of this square lattice walk. The bold final step of the walk may be changed to anyone of the two dashed neutral atmospheric edges to change the conformation of the walk. In this example the size of the neutral endpoint atmosphere is $a_0^e = 2$.

The last edge of the walk in figure 5 (denoted by the arrow-head) can be deleted to create a self-avoiding of length reduced by one edge. This is the *negative endpoint atmosphere* of the walk. The size of the negative endpoint atmosphere will be indicated by a_-^e . The trivial walk of length zero has $a_-^e = 0$, but every other walk of length one or bigger than one has $a_-^e = 1$.

A *neutral endpoint atmosphere* can also be defined. In figure 6 a definition is given: The bold edge is the final step in the walk, and it can be changed to any one of the two alternative positions indicated by the dashed line segments. These dashed line segments compose the neutral endpoint atmosphere of the walk. We denote the size of the neutral endpoint atmosphere by a_0^e .

Generalised Atmospheres: A more general definition of atmospheres (the *generalised atmospheres*) are illustrated in figures 7 and 8.

The *positive generalised atmosphere* of a walk is defined as follows: Consider the set of edges which can be inserted into a walk to increase its length by one while maintaining self-avoidance. Two such edges are illustrated in figure 7. These are positive generalised atmospheric edges, and they are part of the positive generalised atmosphere of a walk.

The size of the positive generalised atmosphere is denoted by a_+^g . If s is the walk on the left in Figure 7, then $a_+^g(s) = 15$. The positive generalised atmosphere of the trivial walk in the square lattice has size $a_+^g = 4$.

A *negative generalised atmosphere* for a given self-avoiding walk can be defined as in Figure 8. By contracting one of the bold edges, a self-avoiding walk of length reduced by one is obtained. The set of edges which can be contracted in this way is the negative generalised atmosphere of the walk. The walk in Figure 8 has size of its negative generalised atmosphere $a_-^g = 3$. All walks, with the exception of the trivial walk of length zero, has its final edge also a negative generalised atmospheric edge. In other words, $a_-^g(s) \geq 1$ if s is a walk of length at least one.

Neutral generalised atmospheres can also be defined in one of several ways [22], and other definitions for atmospheres may be used, see for example references [37, 21].

4.2. Atmospheric Moves in Self-Avoiding Walks. Positive, neutral and negative atmospheres induces *atmospheric moves* by adding, rearranging, or subtracting

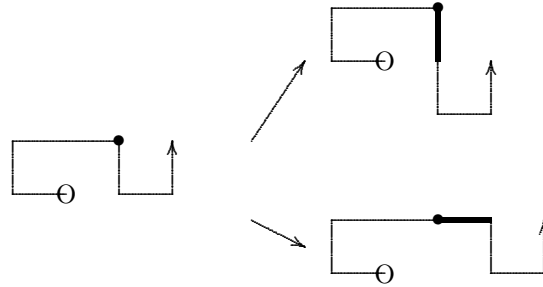


FIGURE 7. The positive generalised atmosphere of a square lattice walk. There are two ways in which an edge can be inserted in the walk at the vertex \bullet on the left. The outcomes are illustrated on the right, with the inserted edge indicated by bold edges. The two bold edges are part of the positive generalised atmosphere of the walk on the left. The collection of edges which can be added to the walk in this way is the positive generalised atmosphere of size a_+^g . The walk on the left has $a_+^g = 15$.

edges from a walk. Such atmospheric moves are already illustrated in Figures 5, 6, 7 and 8.

If atmospheric moves are induced by endpoint atmospheres, then we call them *endpoint atmospheric moves*. Observe that several Monte Carlo algorithms are obtained by implementing endpoint atmospheric moves as elementary moves, including the Rosenbluth and PERM algorithms in sections 2 and 3, and also the Berretti-Sokal algorithm [1]. The pivot algorithm for walks [32] is an implementation of a neutral atmospheric move defined in terms of pivots.

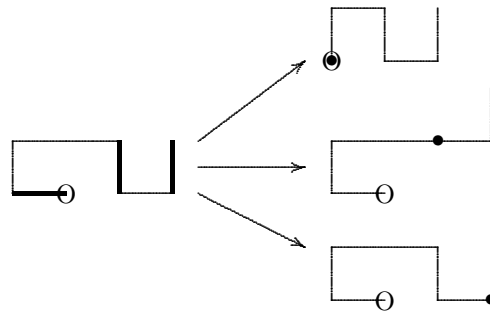


FIGURE 8. The negative generalised atmosphere of a walk in the square lattice. By deleting and contracting one of the three bold edges in the walk on the left, the walks on the right are obtained. All the edges which can be deleted and contracted in this way compose the negative generalised atmosphere of size a_-^g . The walk on the left has $a_-^g = 3$.

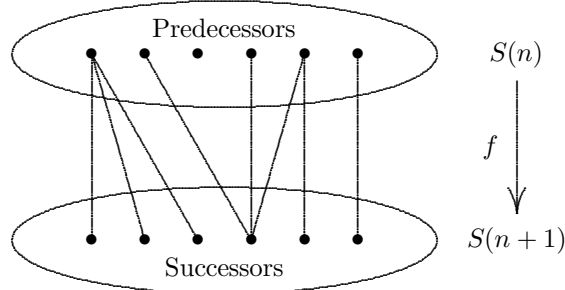


FIGURE 9. Generalised positive atmospheric moves in the GARM algorithm sets up a map $f : S(n) \rightarrow S(n + 1)$ which maps walks of length n (predecessors) in the set $S(n)$ to walks of length $n + 1$ (successors) in the set $S(n + 1)$. f may be represented as graph as illustrated above. Some states in $S(n)$ has no successors; these are trapped conformations.

Observe that any positive atmospheric move defined above (whether general or endpoint) can be reversed in which case it is a negative atmospheric move. Similar, a negative atmospheric move is reversible in terms of a positive atmospheric move. Further, given an arbitrary walk, it is possible to reduce it to the trivial walk by a sequence of negative atmospheric moves. That is, the trivial state communicates with any given state s along a sequence of positive atmospheric moves.

Define $S(n)$ to be the set of walks of length n . Then a positive atmospheric move defines a map $f : S(n) \rightarrow S(n + 1)$ as illustrated in figure 9. An implementation of generalised atmospheric moves defined in Figures 7 and 8 sets up a sampling scheme by iterating the positive atmospheric moves as schematically illustrated in figure 10: If positive endpoint atmospheres are used, then we obtain Rosenbluth sampling. Using more generalised atmospheres, such as illustrated in figure 7, gives a version of GARM. Other definitions of positive and negative atmospheres will give alternative implementations of GARM, provided that every state communicates with the trivial state along a sequence of positive atmospheric moves.

The edges in figure 9 are *linkages* between $S(n)$ and $S(n + 1)$. Consider a walk $s \in S(n)$ together with its associated atmospheres of sizes $a_+(s)$, $a_0(s)$ and $a_-(s)$ (these may be any kind of suitable set of atmospheres). Then the total number of linkages between $S(n)$ and $S(n + 1)$ is given by

$$(4.1) \quad \# \text{ Linkages} = \sum_{s \in S(n)} a_+(s) = \sum_{s \in S(n+1)} a_-(s).$$

Diving this by $c_n = \sum_{s \in S(n)} 1$ shows that the mean positive and negative atmospheres are related by

$$(4.2) \quad \langle a_+ \rangle_n = \frac{1}{c_n} \sum_{s \in S(n)} a_+(s) = \frac{c_{n+1}}{c_n} \frac{1}{c_{n+1}} \sum_{s \in S(n+1)} a_-(s) = \frac{c_{n+1}}{c_n} \langle a_- \rangle_{n+1}$$

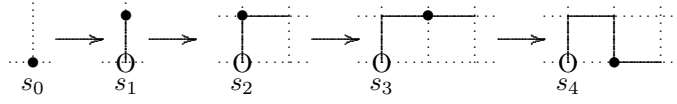


FIGURE 10. Sampling generalised positive atmospheres in GARM. Starting from the trivial walk, at each iteration a vertex \bullet is chosen and a positive atmospheric edge is inserted to generate the next state in the sequence. The atmospheric move is at each step chosen uniformly from those available.

where $\langle \cdot \rangle_n$ is the mean value computed over all walks of length n . Thus,

$$(4.3) \quad \frac{\langle a_+ \rangle_n}{\langle a_- \rangle_{n+1}} = \frac{c_{n+1}}{c_n}.$$

In other words, approximations to the ratio $[c_{n+1}/c_n]$ can be computed by sampling walks in the canonical ensemble (fixed length ensemble) using for example the pivot algorithm. This is one particular method for approximate enumeration that follows from the atmospheric statistics, see references [36, 21] for more details and applications.

4.3. Generalised Atmospheric Sampling of Walks (GARM). GARM is an implementation of the generalised positive atmospheric moves in figure 7 in a Rosenbluth style sampling scheme. The algorithm generates walks and their associated weights along a sequence ϕ and use a counting formula to approximately enumerate walks. We require (1) that the trivial state (walk with length zero) communicates with s along a sequence of positive atmospheric moves, (2) that every positive atmospheric move is reversible by a corresponding negative atmospheric move, and vice versa, and (3) that every neutral atmospheric move is reversible by a corresponding neutral atmospheric move.

GARM is implemented as follows: Let s_0 be the trivial walk of length zero. Select a positive atmospheric move from those available and execute it to obtain s_1 . Once s_n has been constructed, find s_{n+1} by selecting a positive atmospheric move from the positive atmosphere of s_n . At each step, the atmospheric move is selected uniformly from those available. This implementation shows that the sequence $\phi = \langle s_0, s_1, \dots, s_n, s_{n+1}, \dots, s_N \rangle$ is generated by repeatedly applying the map f in figure 9. We say that ϕ is a sequence of length $|\phi| = N + 1$ of states $s_n \in S(n)$ realised by positive atmospheric moves. In figure 10 we illustrate the sampling along ϕ by GARM.

Since the state s_j is obtained from s_{j-1} by uniformly selecting a randomly chosen positive atmospheric move, the conditional probability that s_j follows s_{j-1} is

$$(4.4) \quad \Pr(s_j | s_{j-1}) = \frac{1}{a_+(s_{j-1})}.$$

The probability to sample a given sequence ϕ is therefore

$$(4.5) \quad \Pr(\phi) = \prod_{j=1}^{|\phi|-1} \Pr(s_j | s_{j-1}) = \prod_{j=1}^{|\phi|-1} \left[\frac{1}{a_+(s_{j-1})} \right].$$

The final state of the sequence ϕ is a specific state $s_N \equiv \tau$. The probability that ϕ terminates in the state τ is given by

$$(4.6) \quad \Pr(\tau) = \sum_{\phi \rightarrow \tau} \prod_{j=1}^{|\phi|-1} \left[\frac{1}{a_+(s_{j-1})} \right]$$

where the sum is over all sequences ϕ which starts at the trivial state s_0 of one vertex in the origin, and terminates in the final state which is τ .

We assign a weight $W(\phi)$ to each realised sequence by

$$(4.7) \quad W(\phi) = \prod_{j=1}^{|\phi|-1} \left[\frac{a_+(s_{j-1})}{a_-(s_j)} \right].$$

Each factor in this product is the ratio of the positive atmosphere of the *current state*, divided by the negative atmosphere of the *next state*. This choice of the weight gives the following lemma:

Lemma 4.1. *The average $\langle W(\phi) \rangle$ of the weight of sequences that end in the state τ is unity:*

$$\langle W(\phi) \rangle = \sum_{\phi \rightarrow \tau} W(\phi) \Pr(\phi) = \sum_{\phi \rightarrow \tau} \prod_{j=1}^{|\phi|-1} \left[\frac{1}{a_-(s_j)} \right] = 1.$$

Proof: Consider the average:

$$(4.8) \quad \langle W(\phi) \rangle = \sum_{\phi \rightarrow \tau} \prod_{j=1}^{|\phi|-1} \left[\frac{a_+(s_{j-1})}{a_-(s_j)} \right] \left[\frac{1}{a_+(s_{j-1})} \right] = \sum_{\phi \rightarrow \tau} \prod_{j=1}^{|\phi|-1} \left[\frac{1}{a_-(s_j)} \right].$$

The last term can now be interpreted as the probability that the walk τ is reduced to the trivial walk s_0 composed of a single vertex by executing negative atmospheric moves. This probability is equal to one, since s_0 communicates with the state τ via sequences of positive atmospheric moves which may be reversed into sequences of negative atmospheric moves starting in τ and necessarily terminating in the trivial walk. \square

By summing $\langle W(\phi) \rangle$ over all last states τ which are walks of length n , one obtains the counting formula

$$(4.9) \quad \sum_{|\tau|=n} \langle W(\phi) \rangle = \sum_{|\tau|=n} \sum_{\phi \rightarrow \tau} \prod_{j=1}^{|\phi|-1} \left[\frac{1}{a_-(s_j)} \right] = c_n$$

where c_n is the number of walks of length n . That is, by determining the average of the weights in a given implementation GARM the numbers c_n can be estimated by computing average weights.

Variances of the computed weights $\langle W_n \rangle$ in a GARM simulation do not increase as quickly as in the Rosenbluth algorithm, but they do tend to increase with the length of the walks being sampled. As in the case of PERM, one may introduce enrichment and pruning moves to reduce the variance. GARM with pruning and enrichment proceeds, as for PERM, by tracking the weights of a given sequence. If this weight grows too small, then the sequence can be pruned, and if the weight grows too large, then it can be enriched.

If the implementation of GARM is with endpoint atmospheric moves, then GARM reduces to Rosenbluth sampling. More generally, if GARM is implemented with enrichment and pruning moves, then it generalises PERM and reduces to PERM if it is implemented with endpoint atmospheric moves.

In table 2 the results of a simulation using GARM are compared to exact enumeration data from reference [24]. In this case, GARM sampled one million started sequences of length $n = 72$ to compute average weights as in equation (4.9).

The implementation of GARM allows for the possibility of trapped conformations with zero positive atmospheres, but this rarely occurs in the case of generalised atmospheres. The result is that trapped conformations are encountered only rarely and attrition is not the serious problem it is in Rosenbluth sampling.

n	c_n	GARM $\langle W_n \rangle$	flatGARM $\langle W_n \rangle$
0	1	1	1
1	4	4	4
2	12	12	12
3	36	36.0012	36.0049
4	100	100.0143	100.020
5	284	284.054	284.084
6	780	780.448	779.772
7	2172	2174.17	2172.42
8	5916	5919.34	5916.98
9	16268	16279.7	16263.7
10	44100	44121.1	44087.1
11	120292	1.20385×10^5	1.20297×10^5
12	324932	3.25203×10^5	3.24837×10^5
13	881500	8.82099×10^5	8.81427×10^5
14	2374444	2.37534×10^6	2.37490×10^6
15	6416596	6.41833×10^6	6.41986×10^6
16	17245332	1.72509×10^7	1.72540×10^7
17	46466676	4.64541×10^7	4.65168×10^7
18	124658732	1.24643×10^8	1.24788×10^8
19	335116620	3.35027×10^8	3.35552×10^8
20	897697164	8.96830×10^8	8.98939×10^8
25	123481354908	1.23346×10^{11}	1.23682×10^{11}
30	16741957935348	1.66939×10^{13}	1.67753×10^{13}
35	2252534077759844	2.23868×10^{15}	2.25720×10^{15}
40	300798249248474268	2.98010×10^{17}	3.01309×10^{17}
45	39992704986620915140	3.94580×10^{19}	4.00419×10^{19}
50	5292794668724837206644	5.21079×10^{21}	5.30212×10^{21}
55	698501700277581954674604	6.90023×10^{23}	6.99201×10^{23}
60	91895836025056214634047716	9.06769×10^{25}	9.20095×10^{25}
65	12066271136346725726547810652	1.18578×10^{28}	1.20736×10^{28}
70	1580784678250571882017480243636	1.54860×10^{30}	1.58136×10^{30}
71	4190893020903935054619120005916	4.08355×10^{30}	4.19364×10^{30}

TABLE 2. Approximate Enumeration with GARM and flatGARM.

The dispersion of the weights of realised sequences are controlled similarly to the implementation of PERM, and by using enrichment and pruning. The calculation of generalised atmospheres for walks of length n is $O(n)$; it follows that the computational effort in generating a single sequence with final walk of length n is $O(n^2)$. This shows that GARM slows down significantly with increasing length of the sampled walks and realised sequences. Improving this may be possible using an implementation similar to the pivot algorithm implemented by Clisby [3].

GARM can also be implemented by adding neutral atmospheric moves. In a given iteration the algorithm chooses uniformly from the possible positive and neutral atmospheric moves available to it. The probability of realising a sequence ϕ is

$$(4.10) \quad \Pr(\phi) = \prod_{k=1}^{|\phi|-1} \frac{1}{a_+(s_{k-1}) + a_0(s_{k-1})}.$$

The corresponding weight of the sequence is

$$(4.11) \quad W(\phi) = \prod_{k=1}^{|\phi|-1} \frac{a_+(s_{k-1}) + a_0(s_{k-1})}{a_-(s_k) + a_0(s_k)}.$$

The average weight of all sequences ending in a conformation of size n is c_n ; the proof of this is similar to the above.

4.4. Flat Histogram Generalised Atmospheric Rosenbluth Method (flat-GARM). The use of pruning and enrichment moves in GARM naturally leads to a flat histogram implementation similar to flatPERM. This is the flatGARM algorithm [37].

The pruning and enrichment of states in flatGARM proceeds by the implementation of the same steps used for flatPERM. If the j -th state s_j in a sequence ϕ generated by flatGARM has weight $W(s_j)$, then the implementation proceeds by the calculation of the parameter r

$$(4.12) \quad r = \frac{W(s_j)}{\langle W_j \rangle}$$

where $\langle W_j \rangle$ is the average of the weights of the j -th state in all sequences generated thus far (including the current sequence). Calculating $W(s_j)$ to determine $\langle W_j \rangle$ requires the negative atmosphere of the state s_{j+1} , which have not been constructed yet. We overcome this issue by extrapolating to the negative atmosphere of the next state from the current state. For generalised atmospheres it is not unreasonable to suppose that $a_-(s_{j+1}) = a_-(s_j) + 1$, and for endpoint atmospheres, one has exactly $a_-(s_{j+1}) = a_-(s_j)$ unless s_j is the trivial state. These estimate for generalised atmospheres works well in actual simulations.

Once the parameter r has been computed, pruning and enrichment is implemented as follows: If $r < 1$, then retain the current sequence with probability r and update the weight $W(s_j) \rightarrow W(s_j)/r$. Otherwise, prune the sequence (with probability $1 - r$). If $r > 1$ then enrich the sequence: Compute $c = \lceil r \rceil$ with probability $r - \lfloor r \rfloor$, and $c = \lfloor r \rfloor$ otherwise. Make c copies of the sequence, each with weight $W(s_j)/c$ and continue growing from each, recursively pruning and enriching at each step.

This implementation of flatGARM introduces some initial attrition of started sequences due to pruning, but the parameter r is designed such that it produces a

flat histogram: the pruned sequences are eventually replaced by enriched sequences. A major advantage over PERM and flatPERM is that correlations are suppressed in the enrichment process. While endpoint atmospheric moves in PERM or flatPERM leave the walk unchanged up to the enrichment point, generalised atmospheric moves quickly updates edges even along the early part of the walk, so that correlations between enriched copies soon become statistically of lesser significance.

In table 2 the results of an approximate enumeration with flatGARM is given for a million started sequences of length up to 72 and compared to results from a GARM simulation with the same number of started walks. The data show that flatGARM outperform GARM for longer walks. The CPU-time for these simulations is the same to within a few percent.

5. GENERALISED ATMOSPHERIC SAMPLING (GAS)

The implementation of GARM suggests that an even more general implementation of atmospheric moves, which includes negative atmospheric moves, should be possible. This gives rise to Generalised Atmospheric Sampling (GAS), in which negative atmospheric moves are added to the mix of possible moves in GARM.

Let s be a self-avoiding walk together with its associated atmospheric statistics $a_+(s)$, $a_0(s)$ and $a_-(s)$ of positive, neutral and negative atmospheres together with their corresponding atmospheric moves. As a starting point we require that any two walks communicate along at least one sequence of atmospheric moves (that is, the moves are irreducible on the set $S = \cup_n S(n)$ of all self-avoiding walks where $S(n)$ is the set of self-avoiding walks of length n). We also require that every positive atmospheric move is reversible by a corresponding negative atmospheric move, and vice versa, and that every neutral atmospheric move is reversible by a corresponding neutral atmospheric move.

The atmospheric moves define linkages in S as in figure 11. Let $f : S \rightarrow S$ be a map which maps the states in S to S such that $(s, s') \in f$ if s is a predecessor of s' .

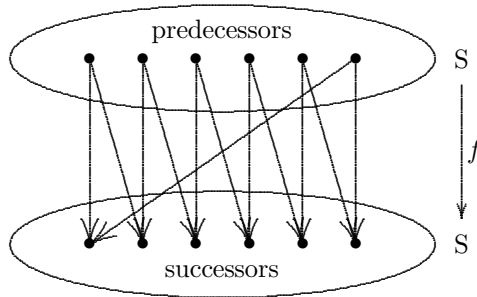


FIGURE 11. Generalised atmospheric moves in the GAS algorithm defines a map $f : S \rightarrow S$ which maps predecessors in S to successors in S . We represent f by the digraph illustrated above. The outdegree of any predecessor vertex in S is equal to the indegree of any successor vertex. This is because every atmospheric move is assumed to be reversible.

Then f may be represented as a digraph with vertex set two copies of S and arcs from vertices in a copy S (the predecessor vertices) to vertices in the second copy of S (the successor vertices). We illustrate this in figure 11. Observe that S is an infinite set of walks of arbitrary length.

Successor states are obtained by applying a move to a predecessor state, and the number of arcs incident with it is equal to the number of atmospheric moves on other states which creates it. Since all the atmospheric moves are reversible, the indegree of any successor vertex s is equal to the outdegree of the same predecessor vertex s . (Note that since each atmospheric move is reversible, s is both a predecessor and a successor of s' (and vice versa)).

Let $s_0 \in S$ be an initial state in the algorithm. Successors of s_0 are states s_1 which can be reached from s_0 by implementing a (positive, neutral or negative) atmospheric move. Once s_1 has been selected as the next state, then s_2 can be selected from amongst the successors of s_1 . Generally, s_{j+1} is selected from amongst the successors of s_j , but necessarily with uniform probability. This process builds a sequence $\phi = s_0 s_1 s_2 \dots s_j \dots$ of states by repeated compositions of the map f defined in figure 11 and above.

The *level* of a state s is its position in the sequence $\phi = s_0 s_1 s_2 \dots s_j \dots$. For example, s_0 has level 0, and s_j has level j . The recursive sampling of states from the successors of the current state is the basic operation of the GAS-algorithm. When the j -th state is sampled, the algorithm is said to *sample in level j* .

The state space S of the GAS-algorithm is the collection of all self-avoiding walks from the origin, an infinite set. The algorithm may be modified so that S is a finite set: Define all walks in S of length n_{max} to have zero positive atmospheres. That is, if the state s_j is sampled by GAS and it is a walk of length n_{max} , then its positive atmosphere is zero. Define $S_*(n_{max}) = \cup_{n=0}^{n_{max}} S(n)$ to be the collection of walks from the origin of maximum length n_{max} , then GAS will sample from the atmospheres of walks in $S_*(n_{max})$ along a Markov Chain.

We must still assign weights to sequences in GAS-sampling: Suppose that state s_j in level j in GAS-sampling has been realised. Introduce the parameter β (possibly dependent on the length of state s_j) and perform an atmospheric move with probabilities

$$(5.1) \quad P_+ = \Pr(\text{positive atmospheric move}) = \frac{\beta a_+(s_j)}{a_-(s_j) + a_0(s_j) + \beta a_+(s_j)};$$

$$(5.2) \quad P_0 = \Pr(\text{neutral atmospheric move}) = \frac{a_0(s_j)}{a_-(s_j) + a_0(s_j) + \beta a_+(s_j)};$$

$$(5.3) \quad P_- = \Pr(\text{negative atmospheric move}) = \frac{a_-(s_j)}{a_-(s_j) + a_0(s_j) + \beta a_+(s_j)},$$

which are normalised to sum up to unity.

The weight of a sequence ϕ is determined as follows. Define ℓ_j to be length (number of edges) in state s_j (which is a walk in S or in $S_*(n_{max})$). Define $|\phi|$ to be the number of levels in a sequence realised by GAS.

If GAS realises a sequence ϕ with $|\phi|$ levels, then the weight of ϕ is

$$(5.4) \quad W(\phi) = \left[\prod_{j=0}^{|\phi|-1} \left[\frac{a_-(s_j) + a_0(s_j) + \beta a_+(s_j)}{a_-(s_{j+1}) + a_0(s_{j+1}) + \beta a_+(s_{j+1})} \right] \right] \prod_{j=0}^{|\phi|-1} \beta^{\sigma(s_j, s_{j+1})}.$$

The function $\sigma(s_j, s_{j+1})$ is defined by

$$(5.5) \quad \sigma(s_j, s_{j+1}) = \begin{cases} -1, & \text{if } s_{j+1} \text{ follows } s_j \text{ through } a_+; \\ +1, & \text{if } s_{j+1} \text{ follows } s_j \text{ through } a_-; \\ 0, & \text{if } s_{j+1} \text{ follows } s_j \text{ through } a_0; \end{cases}$$

Let $P(\phi)$ be the number of positive atmospheric moves in ϕ , and $N(\phi)$ to be the number of negative atmospheric moves in ϕ . Then $P(\phi) - N(\phi)$ is equal to ℓ_L , the length of the final state s_L in ϕ .

With these definitions the product above telescopes down to the much simplified expression:

$$(5.6) \quad W(\phi) = \left[\frac{a_-(s_0) + a_0(s_0) + \beta a_+(s_0)}{a_-(s_L) + a_0(s_L) + \beta a_+(s_L)} \right] \beta^{N(\phi) - P(\phi)}.$$

The probability of realising a particular sequence ϕ is given by

$$(5.7) \quad P(\phi) = \left[\prod_{j=0}^{|\phi|-1} \left[\frac{1}{a_-(s_j) + a_0(s_j) + \beta a_+(s_j)} \right] \right] \beta^{P(\phi)}.$$

The expected value of the weight over all sequences terminating in the state τ is then given by

$$\langle W(\phi) \rangle_\tau = \sum_{\phi: s_0 \rightarrow \tau} \left[\prod_{j=0}^{|\phi|-1} \left[\frac{1}{a_-(s_{j+1}) + a_0(s_{j+1}) + \beta a_+(s_{j+1})} \right] \right] \beta^{N(\phi)},$$

where the summation over $\phi: s_0 \rightarrow \tau$ is over all sequences starting from the trivial state s_0 and ending in state τ .

Reverse each step along ϕ in the above to obtain the sequence $\phi': \tau \rightarrow s_0$. Along ϕ' each positive atmospheric step corresponds to a negative atmospheric step along ϕ , and each negative atmospheric step corresponds to a positive atmospheric step. Thus $N(\phi) = P(\phi')$ and $P(\phi) = N(\phi')$.

Thus, the expression for $\langle W(\phi) \rangle_\tau$ can be written in terms of ϕ' as

$$\langle W(\phi) \rangle_\tau = \sum_{\phi': \tau \rightarrow s_0} \left[\prod_{j=0}^{|\phi'|-1} \left[\frac{1}{a_-(s'_{j+1}) + a_0(s'_{j+1}) + \beta a_+(s'_{j+1})} \right] \right] \beta^{P(\phi')},$$

where ϕ' is the sequence $s'_0 s'_1 \dots s'_j \dots s'_N = s_N s_{N-1} \dots s_{N-j} \dots s_0$. The summand in the above is the probability that a sequence ϕ' starting in the state τ will terminate in the trivial state s_0 if atmospheric steps are given with probabilities

$$(5.8) \quad P^+ = \frac{\beta a_+(s'_{j+1})}{a_-(s'_{j+1}) + a_0(s'_{j+1}) + \beta a_+(s'_{j+1})},$$

$$(5.9) \quad P^0 = \frac{a_0(s'_{j+1})}{a_-(s'_{j+1}) + a_0(s'_{j+1}) + \beta a_+(s'_{j+1})},$$

$$(5.10) \quad P^- = \frac{a_-(s'_{j+1})}{a_-(s'_{j+1}) + a_0(s'_{j+1}) + \beta a_+(s'_{j+1})}.$$

If the set of atmospheric moves is irreducible, and if the mean probabilities of positive and negative atmospheric moves satisfy

$$(5.11) \quad \langle P^+ \rangle_\ell \leq \langle P^- \rangle_\ell$$

over the entire range of lengths of walks in the chain, then this probability (that the chain terminates at the trivial state s_0) is bigger than zero, since the atmospheric moves are reversible and the entire process is a (biased) random walk on the integers which is an ergodic Markov Chain.

The above is in particular true if β satisfies

$$(5.12) \quad \beta \leq \frac{\langle a_1^g \rangle_\ell}{\langle a_+^g \rangle_\ell}$$

for all values of ℓ , or whenever

$$(5.13) \quad \beta \leq \inf_\ell \left[\frac{\langle a_-^g \rangle_\ell}{\langle a_+^g \rangle_\ell} \right].$$

The mean value of the weight of the sequence ϕ ending in state τ is given by

$$(5.14) \quad \langle W(\phi) \rangle_\tau = P(s_0|\tau)$$

where $P(s_0|\tau)$ is the conditional probability that the (backwards) chain will terminate in state s_0 , given that it started in state τ . Summing this over all walks τ of length n shows that

$$(5.15) \quad \sum_{|\tau|=n} \langle W(\phi) \rangle_\tau = c_n \langle P(s_0|\tau) \rangle_n$$

where c_n is the number of walks of length n , and where $\langle P(s_0|\tau) \rangle_n$ is the average conditional probability that sequences starting in a state τ of length n will terminate in s_0 stepping with backwards probabilities P^+ , P^0 and P^- .

Similarly, for walks σ of length m it follows that

$$(5.16) \quad \sum_{|\sigma|=m} \langle W(\phi) \rangle_\sigma = c_m \langle P(s_0|\sigma) \rangle_m.$$

In the case that $S_*(n_{max})$ is our state space, then the GAS algorithm samples along a recurrent and ergodic Markov Chain in a finite state space. This implies that the mean conditional probabilities $\langle P(s_0|\tau) \rangle_n$ and $\langle P(s_0|\sigma) \rangle_m$ become independent of n and m and has a non-zero asymptotic value. This is so for any finite value of β .

In the case that S is our (infinite) state space of all walks from the origin, then the GAS algorithm samples along a recurrent and ergodic Markov Chain in an infinite state space provided that β satisfies equation (5.13). In this event the mean conditional probabilities $\langle P(s_0|\tau) \rangle_n$ and $\langle P(s_0|\sigma) \rangle_m$ are asymptotically independent of n and m and is strictly positive.

Hence, by taking the ratios of (5.15) and (5.16) these factors cancel and one finds that

$$(5.17) \quad \frac{c_n}{c_m} = \frac{\sum_{|\tau|=n} \langle W(\phi) \rangle_\tau}{\sum_{|\sigma|=m} \langle W(\phi) \rangle_\sigma}.$$

In particular, if $m = 0$, then $c_0 = 1$ and

$$(5.18) \quad c_n = \frac{\sum_{|\tau|=n} \langle W(\phi) \rangle_\tau}{N_0},$$

since the weight of a sequence terminating in the trivial state s_0 is one, and where N_0 is the number of visits of the sequence ϕ to the state s_0 .

In a simulation one collects the (total accumulated) weights $\sum_{|\tau|=j} \langle W(\phi) \rangle_\tau$ for walks of length j along sequences ϕ . Ratios of these accumulated weights are

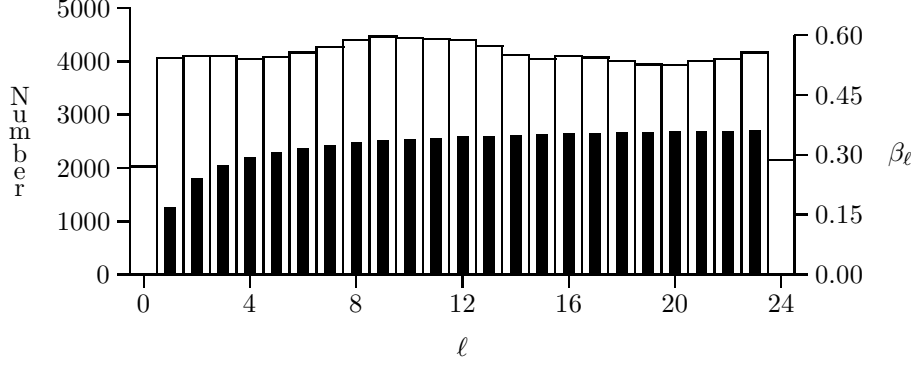


FIGURE 12. Flat histogram sampling in the flatGAS-algorithm. The solid bars give the values of β_ℓ as a function of ℓ on the right hand side scale. $\beta_0 = 1$ by definition and is left away, since the probability of stepping from the walk of length 0 to a walk of length 1 is $1/2d$, independent of β_0 . Since $n_{max} = 24$, the positive atmospheres of walks of length 24 is put equal to zero, and we also put $\beta_{24} = 0$. The open bars display the number of times a walk of length ℓ was observed in the simulation; the scale is on the left hand side. The histogram is reasonably flat over the entire range of ℓ .

estimates of ratios of c_n as in equations (5.17) and (5.18). Realising N independent sequences ϕ gives independent estimates of c_n/c_m , from which one may extract estimates of c_n .

5.1. Flat Histogram Generalised Atmospheric Sampling (flatGAS). A flat histogram (over the lengths of the walks) implementation of GAS is possible. Consider the interval $[0, n_{max}]$, and our aim is to sample states in $S_*(n_{max}) = \cup_{n=0}^{n_{max}} S(n)$ such that the algorithm samples roughly the same number or times on states of length $m \in [0, n_{max}]$. Define $a_+(s) = 0$ if the length of s is $\ell_s = n_{max}$, and implement GAS to realise a sequence $\phi = \langle s_0, s_1, \dots, s_j, \dots \rangle$ of states of lengths ℓ_j in $[0, n_{max}]$.

Flat histogram sampling in GAS can be performed by changing the transition probabilities in equations (5.1), (5.2) and (5.3) by making β dependent on ℓ_j , the length of the state s_j . A hint for the kind of modification needed is given by equation (5.13). Choose β_ℓ such that equation (5.11) becomes

$$(5.19) \quad \langle P^+ \rangle_\ell = \langle P^- \rangle_\ell$$

and replace β by β_ℓ in equations (5.1), (5.2) and (5.3). In particular, for each value of $\ell \in [0, n_{max}]$,

$$(5.20) \quad \beta_\ell = \frac{\langle a_-^g \rangle_\ell}{\langle a_+^g \rangle_\ell}.$$

This implies that that

$$(5.21) \quad P_+ = \Pr(\text{positive atmospheric move}) = \frac{\beta_{\ell_j} a_+^g(s_j)}{a_-^g(s_j) + a_0^g(s_j) + \beta_{\ell_j} a_+^g(s_j)};$$

$$(5.22) \quad P_0 = \Pr(\text{neutral atmospheric move}) = \frac{a_0^g(s_j)}{a_-^g(s_j) + a_0^g(s_j) + \beta_{\ell_j} a_+^g(s_j)};$$

$$(5.23) \quad P_- = \Pr(\text{negative atmospheric move}) = \frac{a_-^g(s_j)}{a_-^g(s_j) + a_0^g(s_j) + \beta_{\ell_j} a_+^g(s_j)},$$

with the result that $\langle P^+ \rangle_\ell = \langle P^- \rangle_\ell$ and $\langle P_+ \rangle_\ell = \langle P_- \rangle_\ell$ for each $\ell \in (0, n_{max})$. That is, the probability of a positive atmospheric move is equal to the probability of a negative atmospheric move on average locally for each length between 0 and n_{max} . With these choices for the β_j , the algorithm perform on average a random walk on the integers n which should give a uniform distribution on $(0, n_{max})$ and visits the states of length 0 and n_{max} half as frequently as the states of lengths in $(0, n_{max})$.

In figure 12 the flat histogram of a simulation of flat histogram GAS (flatGAS) is illustrated. In this simulation, $n_{max} = 24$, and so $\beta_{24} = 0$. The values of β_ℓ were computed by atmospheric ratios (see equation (5.20)) from an earlier run. The binning of states of length $\ell \in [0, 24]$ in a sequence of length 100000 with the β_ℓ as given, are indicated by the open bars with scale on the left hand axis. The states

n	c_n	GAS $\langle W_n \rangle$	flatGAS $\langle W_n \rangle$
0	1	1	1.00040
1	6	6.00013	6.00254
2	30	30.0131	29.9971
3	150	150.202	149.95
4	726	727.451	725.929
5	3534	3541.02	3535.28
6	16926	16958.1	16944.4
7	81390	81540.4	81497.5
8	387966	388757	388447
9	1853886	1.85830×10^6	1.85568×10^6
10	8809878	8.83188×10^6	8.81789×10^6
11	41934150	4.20464×10^7	4.19746×10^7
12	198842742	1.99482×10^8	1.99077×10^8
13	943974510	9.47294×10^8	9.45217×10^8
14	4468911678	4.48663×10^9	4.47232×10^9
15	21175146054	2.12653×10^{10}	2.11767×10^{10}
16	100121875974	1.00532×10^{11}	1.00112×10^{11}
17	473730252102	4.75742×10^{11}	4.73740×10^{11}
18	2237723684094	2.24647×10^{12}	2.23784×10^{12}
19	10576033219614	1.06104×10^{13}	1.05755×10^{13}
20	49917327838734	5.00652×10^{13}	4.98981×10^{13}
25	116618841700433358	1.16875×10^{17}	1.16462×10^{17}
30	270569905525454674614	2.71271×10^{20}	2.70128×10^{20}

TABLE 3. Approximate Enumeration in 3d with GAS and flatGAS.

with $\ell = 0$ and $\ell = 24$ were sampled about half as frequently as those states with $\ell \in (0, 24)$.

The weights along a sequence ϕ realised by flatGAS are defined as before, only now with β dependent on ℓ :

$$(5.24) \quad W(\phi) = \left[\frac{a_-^g(s_0) + a_0^g(s_0) + \beta_{\ell_0} a_+^g(s_0)}{a_-^g(s_L) + a_0^g(s_L) + \beta_{\ell_L} a_+^g(s_L)} \right]^{\|\phi\|-1} \prod_{j=0}^{\|\phi\|-1} \beta_{\ell_j}^{\sigma(\phi_j, \phi_{j+1})}.$$

The argument proceeds similarly as set out for GAS above: The result is that the ratio of mean accumulated weights of walks of lengths n and m is equal to the ratio c_n/c_m :

$$(5.25) \quad \frac{\sum_{|\tau|=n} \langle W(\phi) \rangle_{\tau}}{\sum_{|\sigma|=m} \langle W(\phi) \rangle_{\sigma}} = \frac{c_n}{c_m}.$$

The implementation of flatGAS proceeds by the realising N independent sequences ϕ and computing weights for each while updating the values of the β_{ℓ} following each sequence ϕ . If the initial choices for the β_{ℓ} were erroneous, updated estimates quickly improves, and the algorithm self-tunes to produce a flat histogram on the interval $(0, n_{max})$.

In table 3 the estimates of c_n for cubic lattice walks are given using both a GAS and a flatGAS simulation. In this simulation GAS was implemented using endpoint atmospheres. The implementation used $\beta = 0.212$ and realised 100 sequences of length 10^7 . In these simulations, $n_{max} = 71$. The flatGAS simulation quickly settled down into flat histogram sampling. The observed errors show that the GAS and flatGAS data are scattered about the exact enumeration data (obtained from [4]). Overall, these relatively short runs produced estimates that deviate by less than 1% from the exact data. More accurate results can be obtained by increasing either the length of the sequences, or by increasing the number of sequences.

These implementations of GAS are in every respect, like Rosenbluth and PERM sampling, GARM and flatGARM, an approximate enumeration method. An implementation of GAS with endpoint atmospheres is a generalisation of the Berretti-Sokal dynamic Metropolis Monte Carlo algorithm [1]; but with sampling along weighted Markov Chains. This implementation is called GABS in reference [22], and its flat histogram implementation flatGABS is an efficient approximate enumeration tool for self-avoiding walks; primarily because the implementation performs atmospheric moves in $O(1)$ CPU-time, giving high quality estimates for c_n .

6. CONCLUSIONS

The implementation of GARM and flatGARM have been extended to two dimensional polygons in reference [37], and can easily be shown to be useful in sampling lattice trees and animals, as well as other lattice objects such as plaquette surfaces. This is the real improvement of GARM over PERM sampling; it generalises PERM by the use of atmospheres such that the application of this sampling technique to other lattice objects becomes a matter of defining atmospheres and computing weights. In reference [7] GARM was used to determine the size of classes of group elements.

The GAS and flatGAS algorithms are similarly a generalisation of GARM. GAS introduces negative atmospheric moves into the mix of moves in GARM, while flatGAS shows that the algorithm can self-tune to flat histogram sampling without

the introduction of enrichment or pruning. Instead, the algorithm achieves flat-histogram sampling by tuning its single parameter β locally (for each n). More detail can be found on this algorithm in reference [22].

Overall the estimation of critical exponents and growth constants for walks using approximate enumeration data generated by GARM or GAS do not approach the accuracy of the exact enumeration for these quantities (see equations (1.9), (1.11) and (1.14)). In reference [22] the flatGAS estimates $\mu = 2.6383 \pm 0.0001$ and $\gamma = 1.34 \pm 0.02$ in two dimensions and $\mu = 4.684 \pm 0.001$ and $\gamma = 1.15 \pm 0.02$ in three dimensions have been obtained using an endpoint atmosphere implementation of flatGAS (this is flatGABS). A more efficient implementation of these algorithms, using coding techniques such as in reference [3], should improve these estimates, and is a natural next step in the examination of the properties of GARM and GAS algorithms.

Acknowledgements: The author is grateful to A. Rechnitzer for fruitful discussions.

REFERENCES

1. A. Berretti and A.D. Sokal *New Monte Carlo Method for the Self-Avoiding Walk*. J. Stat. Phys. **40** (1985), 483-531
2. S. Caracciolo, M.S. Causo and A. Pelissetto *High-Precision Determination of the Critical Exponent Gamma for Self-Avoiding Walks*. Phys. Rev. E **57** (1998), 1215-1218
3. N. Clisby *Accurate Critical Exponents for Self-Avoiding Walks via a Fast Implementation of the Pivot Algorithm*. In preparation (2008).
4. N. Clisby, R. Liang and G. Slade *Self-Avoiding Walk Enumeration via the Lace Expansion*. J. Phys. A: Math. Theor. **40** (2007), 10973-11017
5. B. Duplantier *Polymer Network of Fixed Topology: Renormalisation, Exact Critical Exponent γ in Two Dimensions*. Phys. Rev. Lett. **57** (1986), 941-944
6. P.-G. de Gennes *Scaling Concepts in Polymer Physics*. Cornell University Press: New York (1979)
7. M. Elder, E. Fusy and A. Rechnitzer *Counting Elements and Geodesics in Thompson's Group F*. Preprint.
8. P.J. Flory *The Configuration of a Real Polymer Chain*. J. Chem. Phys. **17** (1949), 303-310
9. P.J. Flory *Statistical Thermodynamics of Semi-Flexible Chain Molecules*. Proc. Roy. Soc. (London) A **234** (1955), 60-73
10. P.J. Flory *Statistical Mechanics of Chain Molecules*. Wiley Interscience: New York (1969)
11. P. Grassberger *Pruned-Enriched Rosenbluth Method: Simulation of θ -polymers of Chain Length up to 1 000 000*. Phys. Rev. E **56** (1997), 3682-3693
12. A.J. Guttmann and A.R. Conway *Square Lattice Self-Avoiding Walks and Polygons*. Ann. of Comb. **5** (2001), 319-345
13. J.M. Hammersley *Limiting Properties of Numbers of Self-Avoiding Walks*. Phys. Rev. **118** (1960), 656-656
14. J.M. Hammersley *The Number of Polygons on a Lattice*. Math. Proc. Camb. Phil. Soc. **57** (1961), 516-523
15. J.M. Hammersley *Generalisation of the Fundamental Theorem on Sub-Additive Functions*. Math. Proc. Camb. Phil. Soc. **58** (1962), 235-238
16. J.M. Hammersley and K.W. Morton *Poor Man's Monte Carlo*. J. Roy. Stat. Soc. B **16** (1954), 23-38
17. T. Hara and G. Slade *The Lace Expansion for Self-Avoiding Walks in Five or More Dimensions*. Rev. Math. Phys. **4** (1990), 235-327
18. T. Hara and G. Slade *Critical Behaviour of Self-Avoiding Walks in Five or More Dimensions*. Bull. AMS **25** (1991), 417-423

19. T. Hara, G. Slade and A.D. Sokal *New Lower Bounds on the Self-Avoiding-Walk Connective Constant*. J. Stat. Phys. **72** 3/4 (1993), 479-517
20. H.-P. Hsu, W. Nadler and P. Grassberger *Scaling of Star Polymers with 1-80 Arms*. Macromol. **37** (2004), 4658-4663
21. E.J. Janse van Rensburg and A. Rechnitzer *Atmospheres of Polygons and Knotted Polygons*. J. Phys. A: Math. Theo. **41** (2008), 105002-105025
22. E.J. Janse van Rensburg and A. Rechnitzer *Generalised Atmospheric Sampling of Self-Avoiding Walks*. Preprint (2008)
23. I. Jensen *A Parallel Algorithm for the Enumeration of Self-Avoiding Polygons on the Square Lattice*. J Phys. A: Math. Gen. **36** (2003), 5731-5745
24. I. Jensen *Enumeration of Self-Avoiding Walks on the Square Lattice*. J. Phys. A: Math. Gen. **37** (2004), 5503-5524
25. I. Jensen and A.J. Guttmann *Self-Avoiding Walks, Neighbour-Avoiding Walks and Trials on Semi-Regular Lattices*. J. Phys. A: Math. Gen. **31** (1998), 8137-8145
26. I. Jensen and A.J. Guttmann *Self-Avoiding Polygons on the Square Lattice*. J. Phys. A: Math. Gen. **32** (1999), 4867-4876
27. H. Kesten *On the Number of Self-Avoiding Walks*. J. Math. Phys. **4** (1963), 960-969
28. H. Kesten *On the Number of Self-Avoiding Walks II*. J. Math. Phys. **5** (1964), 1128-1137
29. J. Krawczyk, T. Prellberg, A.L. Owczarek and A. Rechnitzer *Stretching of a Chain Polymer Adsorbed at a Surface*. J. Stat. Mech.: Theo. Expr. **P10004** (2004)
30. J.C. Le Guillou and J. Zinn-Justin *Accurate Critical Exponents from Field Theory*. J. de Physique **50** (1985), 1365-1370
31. N. Madras and G. Slade *The Self-Avoiding Walk*. (Birkhäuser: Boston) (1993)
32. N. Madras and A.D. Sokal *Nonergodicity of Local, Length-preserving Monte Carlo Algorithms for the Self-Avoiding Walk*. J. Stat. Phys. **47** (1987), 573-595
33. P. Nidras *Grand Canonical Simulations of the Interacting Self-Avoiding Walk Model*. J. Phys. A: Math. Gen. **29** (1996), 7929-7942
34. P. Nidras and R. Brak *New Monte Carlo Algorithms for Interacting Self-Avoiding Walks*. J. Phys. A: Math. Gen. **30** (1997), 1457-1469
35. G.L. O'Brien *Monotonicity of the Number of Self-Avoiding Walks*. J. Stat. Phys. **59** (1990), 969-979
36. A. Rechnitzer and E.J. Janse van Rensburg *Canonical Monte Carlo Determination of the Connective Constant of Self-Avoiding Walks*. J. Phys. A: Math. Gen. **35** (2002), L605-L612
37. A. Rechnitzer and E.J. Janse van Rensburg *Generalised Atmospheric Rosenbluth Methods (GARM)*. J. Phys. A: Math. Theo. **41** (2008), 442002-442010
38. M.N. Rosenbluth and A.W. Rosenbluth *Monte Carlo Calculation of the Average Extension of Molecular Chains*. J. Chem. Phys. **23** (1955), 356-359
39. F.T. Wall and J.J. Erpenbeck *Statistical Computation of Radii of Gyration and Mean Internal Dimensions of Polymer Molecules*. J. Chem. Phys. **30** (1959), 634-637

DEPARTMENT OF MATHEMATICS AND STATISTICS, YORK UNIVERSITY, TORONTO, ONTARIO, M3J 1P3, CANADA

Current address: Department of Mathematics and Statistics, York University, Toronto, Ontario, M3J 1P3, Canada

E-mail address: rensburg@yorku.ca