A nonlocal Monte Carlo algorithm for lattice trees

J. Phys. A: Math. Gen. **25** (1992) 303–333. Printed in the UK

A non-local Monte Carlo algorithm for lattice trees

E J Janse van Rensburg[†] and N Madras[‡]

[†] Super Computer Computations Research Institute, Florida State University, Tallahassee, FL 32306-4052, USA

[‡] Department of Mathematics and Statistics, York University, 4700 Keele Street, North York, Ontario, M3J 1P3, Canada

Received 29 May 1991, in final form 1 August 1991

Abstract. A new non-local algorithm for the simulation of trees on the lattice Z^d is proposed. We study the implementation and the properties of the algorithm, and show that it is decisively better than an algorithm which performs only local moves. We use the new algorithm to investigate the properties of lattice trees in two, three , four, eight and nine dimensions.

1. Introduction

The numerical and theoretical study of lattice trees provides a natural model for calculating the properties of branched polymers in dilute solution. It is also believed that lattice trees share the same universality class as lattice animals (Lubensky and Isaacson 1979, Seitz and Klein 1981, Duarte and Ruskin 1981), so that the critical exponents of animals can be determined numerically by investigating trees, which are simpler to simulate than animals.

The critical exponents of animals in d-dimensions are related to the Lee–Yang edge singularity (Parisi and Sourlas 1981, Fisher 1978, Kurze and Fisher 1979, Bovier *et al* 1984) of the Ising model in an imaginary magnetic field in $(d-2)$-dimensions through the relations

$$\nu(d+2) = (\sigma(d)+1)/d \tag{1.1}$$

$$\theta(d+2) = \sigma(d)+2 \tag{1.2}$$

where σ is the exponent which controls the magnetization of the Ising model near the edge singularity. The exponents ν and θ are defined by $\langle r^2\rangle_n \sim n^{2\nu}$ and $t_n \sim n^{-\theta}\lambda^n$, where $\langle r^2\rangle_n$ is the mean square radius of gyration of trees with n vertices, and t_n is the number of (unrooted) trees with n vertices. λ is the lattice-dependent growth constant of lattice trees. Since the Ising model is exactly solvable in zero and one dimensions ($\sigma(0) = -1$, and $\sigma(1) = -1/2$), we can get the 'exact' values $\theta(2) = 2$ and $\theta(3) = 3/2$, and $\nu(3) = 1/2$. (Note that equation (1.1) breaks down if $d = 0$.)

Subsequent results indicated that the dimensional reduction used to derive equations (1.1) and (1.2) fails if the Ising model is in a real magnetic field (Fisher *et al* 1984, Imbrie 1984). It is therefore necessary to determine the validity of these equations by a high precision numerical simulation. To achieve this aim, several numerical studies have been performed (Glaus 1985, Duarte 1986, Duarte and Cadilhe 1989). In this paper we aim to propose a better algorithm than those used in those studies, and we use it to estimate $\nu(d)$ and in particular to consider the validity of equation (1.1).

Aside from their importance in statistical mechanics, lattice trees are also of considerable interest in chemical physics as models of branched polymers in dilute silution. There is an enormous literature on simulation of polymers in general (e.g. Kremer and Binder 1988). Most of the Monte Carlo methods developed for branched polymers are for trees with fixed topology, such as 'stars' or 'brushes' (e.g. Carmesin and Kremer 1988, Whittington *et al* 1986, Lipson *et al* 1987), rather than arbitrary trees. (Polymer chemists' emphasis is frequently on polymer dynamics, but non-local algorithms such as ours generally only give information about static properties.)

An even more basic example of a random geometric object with non-local interactions is the self-avoiding walk. The simplicity of this model makes for innovative algorithm designs for exact enumeration studies and Monte Carlo simulations. Among Monte Carlo algorithms, the pivot algorithm brought about the most dramatic improvement in the simulation of walks, especially when simulated in the canonical ensemble (that is, with fixed length) (Madras and Sokal 1987, Janse van Rensburg *et al* 1990). The basic idea of the pivot algorithm is the use of non-local elementary transitions; that is, it tries to change large parts of the walk all at once. Borrowing from this idea, we shall consider an implementation of large, non-local elementary transitions in a Monte Carlo simulation of lattice trees, and we shall show that this brings about a dramatic improvement in the simulation.

Earlier studies of lattice trees and animals by exact enumeration and by Monte Carlo methods (Peters *et al* 1979, Redner 1979, Gould and Holl 1981, Seitz and Klein 1981, Gaunt *et al* 1982) that were concerned with estimating ν had estimates varying from 0.45 to 0.53 in three dimensions; a considerable spread of results. Glaus (1985) and Caracciolo and Glaus (1984) performed a grand canonical Monte Carlo simulation of lattice trees estimating $\nu(3) = 0.495 \pm 0.013$ in three dimensions and $\nu(2) = 0.635 \pm 0.015$ in two dimensions. These last results strongly support the validity of the dimensional reduction leading to equations (1.1) and (1.2). In this paper, we estimate $\nu(3) = 0.4960 \pm 0.0052$ and $\nu(2) = 0.637 \pm 0.012$.

Our paper is organized as follows. Section 2 presents basic definitions, describes our algorithms and proves that they are ergodic and reversible, and discusses properties of lattice trees that we will study (e.g. radius of gyration, span, longest path and mean branch size). Section 3 discusses the implementation of these algorithms on the computer, with particular reference to the efficiency of the various steps. Section 4 is a detailed analysis of the numerical results of our simulations. Our conclusions about our algorithms are present in section 5.

2. Basic definitions and methods

Let Z^d be the d-dimensional hypercubic lattice. A *lattice bond animal* (or simply an *animal*) is a connected subgraph of Z^d. We define a *lattice bond tree* (or simply a *tree*) as an animal with no cycles. (A cycle is a walk containing at least two edges, with all its vertices (or *sites*) distinct, except the first and last vertices, which are the same.) Therefore, on a tree, there is only one path between any two given points: it is a *simply connected* object.

Let ω be a tree, and let v be a vertex in ω. Then we say that v is a vertex of degree i if there are i edges in ω incident on v. An edge χ in ω is a *leaf* if one endpoint of χ is a vertex of degree one. Deleting any edge of a tree results in two connected components, each itself a subtree of the original tree; any subtree which can be obtained by deleting a single edge is called a *branch*. Evidently, every leaf corresponds to a branch consisting of one vertex, and vice versa; we shall sometimes abuse our terminology by referring to a leaf as a branch consisting of one edge.

2.1. *Canonical Monte Carlo algorithms for lattice trees*

The symmetry group of the cubic lattices is the octahedral group O_h. Every tree in the lattice can be transformed by any element of O_h, which is typically a reflection or a rotation. We define now two possible algorithms for lattice bond trees in the canonical ensemble (fixed number of sites n). Let d be the number of dimensions, and suppose that ω is an (unrooted) tree with n sites and $(n-1)$ edges.

Algorithm A: (Leaf-mover). In this algorithm we attempt only small moves on the tree, i.e. one edge at a time. The essential idea is similar to that of Duarte (1986), and to the grand canonical algorithms of Glaus (1985), and Caracciolo and Glaus (1984). The algorithm flows as follows:

A1. Pick an edge at random on the tree.
A2. If this edge is not a leaf, then we count this as a failed transition and go to step A1. Otherwise, we delete the leaf.
A3. Pick a vertex on the rest of the tree (which has $(n-1)$ vertices).
A4. Try to append a leaf to this vertex by randomly choosing one of its $2d$ nearest neighbours. If this creates a cycle then we count this as a failed transition and we go to step A1. Otherwise, we have a succesful transition and we update the old tree before we go to step A1 for the next attempt.

Algorithm B: (Branch-mover). In this algorithm we attempt large, non-local transitions. The essential idea of the algorithm is the following. We pick a branch in the tree at random and we break it off. The branch is then transformed (e.g. rotated) by an element of O_h. We then attempt to append the branch at another location in the tree. The algorithm flows as follows:

B1. Pick an edge at random in the tree.
B2. Delete this edge. This breaks the tree into two subtrees, with one subtree typically bigger than the other.
B3. Find the smaller subtree and apply a randomly chosen element of the octahedral group to it.
B4. Pick two vertices at random, one on each of the two subtrees.
B5. Translate the smaller subtree such that the vertices chosen in step B4 are nearest neighbours on the lattice, in any of $2d$ possible orientations. Look for intersection between the (rotated and translated) smaller subtree and the bigger subtree. If there is an intersection, then we have a failed attempt, so go back to step B1 for the next attempt. Otherwise, we have a new tree, consisting of the bigger subtree, the (rotated and translated) smaller subtree, and a new edge joining the two vertices chosen in step B4. Update the old tree and go to step B1 for the next attempt.

The elementary transitions in algorithm A are just special cases of the possible elementary transitions which can occur in algorithm B; if the smaller subtree (branch) in algorithm B is a single vertex, then the attempted transition is identical to that of algorithm A.

Algorithms A and B are Monte Carlo algorithms (Metropolis *et al* 1953) simulating trees in the canonical ensemble (with a fixed number, n, of vertices). Let T_n be the set of all (unrooted) trees (modulo a translation) with n vertices in the hypercubic lattice. Let the cardinality of T_n be t_n. Then it is believed that

$$t_n \sim n^{-\theta}\lambda^n \tag{2.1}$$

where λ is the growth constant for trees on the lattice, and θ is a critical exponent. We assign an equal weight to each tree in T_n. Algorithms A and B have finite state space T_n and we shall see that they each have the uniform invariant probability measure

$$\pi_\omega = t_n^{-1} \qquad \forall\omega \in T_n. \tag{2.2}$$

The basic elementary transitions of each algorithm are described by a transition probability matrix $\mathbf{P} = \{p(\omega \to \nu)\} = \{p_{\omega\nu}\}$ which has the following properties:

(1) For each $\omega, \nu \in T_n$ there exists an $m > 0$ such that the m-step probability from ω to ν, $p_{\omega\nu}(m)$, is positive. This is *ergodicity* of the algorithm, and we prove it in section 2.2.
(2) For each tree $\nu \in T_n$, $\sum_{\omega\in T_n} \pi_\omega p_{\omega\nu} = \pi_\nu$. This will be proven for algorithms A and B in section 2.2. Therefore, it follows that π_ω is the unique limit distribution of the Markov chain with state space T_n and transition probability matrix $\mathbf{P}$ (Kemeny and Snell 1976).

Let the observed states of this Markov chain be represented by X_i. The states X_i and X_{i+1} are in general correlated, so that the calculation of error bars for the mean of a real-valued function $A(\omega)$, $\omega \in T_n$, is a complicated procedure. If we start the Markov chain in equilibrium, then $\{A_i\} = \{A(X_i)\}$ is a stationary stochastic process with mean

$$\langle A_t \rangle = \sum_{\omega\in T_n} \pi_\omega A(\omega) \tag{2.3}$$

and unnormalized autocorrelation function

$$C_{AA}(s) = \langle A_t A_{t+s}\rangle - \langle A_t\rangle^2. \tag{2.4}$$

The normalized autocorrelation function is defined by

$$\rho_{AA}(s) = \frac{C_{AA}(s)}{C_{AA}(0)}. \tag{2.5}$$

Once the Markov process is in equilibrium, then the *integrated autocorrelation time* is given by

$$\tau_A = \frac{1}{2}\sum_{t=-\infty}^{\infty} \rho_{AA}(t). \tag{2.6}$$

The integrated autocorrelation time controls the statistical error in the Monte Carlo measurements of the mean $\langle A_t \rangle$ of the observable A. The variance in the sample mean $\bar{A}$, over N observations, is asymptotically given by

$$\sigma^2(A) \sim \frac{1}{N}(2\tau_{\text{int}}(A))C_{AA}(0). \tag{2.7}$$

In other words, the effective number of independent observations is $N/(2\tau_{\text{int}})$ (Madras and Sokal 1988).

The relaxation time of the slowest mode in the system is called the *exponential autocorrelation time* τ_e (Madras and Sokal 1988). If the normalised autocorrelation function decays exponentially, then τ_e is the rate of that decay associated with the slowest mode in the system. If we estimate the exponential autocorrelation time associated with a variable x, then we indicate it by $\tau_{e,x}$. Typically, τ (the integrated autocorrelation time) is of the same order as τ_e (or better).

2.2. *Ergodicity and reversibility*

It is now necessary to prove that algorithms A and B are ergodic and reversible (satisfy detailed balance).

2.2.1. Ergodicity. Both algorithms A and B are ergodic because any tree of $(n-1)$ edges can be transformed into a straight line in $(n-1)$ steps. In detail: let ω be a tree with n vertices (and consequently $(n-1)$ edges). Let $\{e_i\}$ be the set of unit vectors in Z^d. We find the *top vertex* t of ω by a lexicographic ordering of all the vertices. Since ω is a tree, it has at least two leaves. There is therefore at least one leaf which has an endpoint of degree 1 which is not t. Move this leaf and append it to t in the e_1 direction. Then there is a new top vertex: $(t+e_1)$. Repeat this process, delete a leaf and append it to the top vertex. Every edge that we remove must have had endpoints with first components less than or equal to the first component of t (by the definition of t); and is moved to a location with endpoints $(t+(i-1)e_1, t+ie_1)$ (if it is the ith leaf to be moved). After $(n-1)$ iterations the vertices in the new tree will be $\{t, t+e_1, t+2e_1, \ldots, t+(n-1)e_1\}$, and the associated edges will be $(t+(i-1)e_1, t+ie_1)$, where $i \in \{1, 2, \ldots, n-1\}$. Algorithms A and B are therefore ergodic.

2.2.2. Reversibility of algorithm A. Let ω be a tree, and let ω' be a tree that can be obtained from ω by moving a single leaf. The probability of obtaining ω' is given by the probability of (1) picking the correct leaf, (2) picking the correct site to append it to, and (3) picking the correct orientation when we append the new leaf. In step A1 of algorithm A we see that the probability for picking a particular leaf is $1/(n-1)$. The resulting tree has $(n-1)$ sites left, so the probability for picking a particular site is $1/(n-1)$, and there are $2d$ possible ways of attempting to append the new leaf. Therefore

$$p(\omega \to \omega') = \frac{1}{2d(n-1)^2} \tag{2.8}$$

where $p(\omega \to \omega')$ is the probability of obtaining ω' from ω by moving a single leaf. The reverse process is obviously the same: therefore

$$p(\omega \to \omega') = p(\omega' \to \omega) \tag{2.9}$$

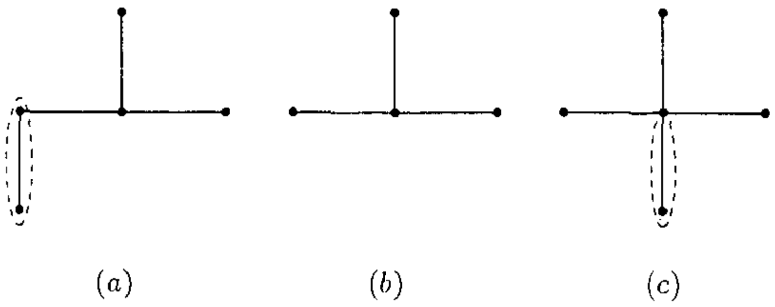

Figure 1. If an algorithm for trees picks leaves and places them back by selecting a site at random from the perimeter of the tree, then it is not reversible. The marked leaf in (*a*) is picked with probability $1/3$. It is put back in (*b*) with probability $1/8$. If we reverse the process, then we find $1/4$ to move from (*c*) to (*b*), and $1/8$ to move from (*b*) to (*a*).

whenever $p(\omega \to \omega') > 0$. Also, equation (2.9) clearly holds if $p(\omega \to \omega') = 0$, and so algorithm A is reversible.

We note that the algorithm for site trees of Duarte (1986), and also Duarte and Cadilhe (1989), is *not* reversible. Consider for example the tree in figure 1(*a*). In their algorithm a choice is made from the *leaves*, and the leaf is deleted. With probability $1/3$ we obtain the tree in figure 1(*b*). The algorithm then selects a site from the perimeter set of the tree, and attempts to append the leaf there. In figure 1(*b*), there are eight perimeter sites, so we obtain figure 1(*c*) with probability $1/8$ from 1(*b*), and with probability $1/24$ from 1(*a*). A similar argument shows that the probability of obtaining 1(*a*) from 1(*c*) is $1/32$. The algorithm is therefore not reversible.

Reversibility of Algorithm B. Let ω be a tree, and let ω' be a tree that can be obtained from ω by moving a single branch of ω. The probability of obtaining ω' is given by the probability of (1) picking the correct branch, (2) choosing the correct element of the octahedral group to rotate the chosen branch, (3) picking the correct vertices on each branch to reconnect the tree, and (4) putting the chosen vertices in the correct orientation to each other. We choose a branch by deleting an edge; this can be done in $(n-1)$ ways, if the tree has n vertices. Suppose that the octahedral group has o_h elements, that the number of vertices in the branch is k, and that we perform the algorithm in d dimensions. Then

$$p(\omega \to \omega') = \frac{1}{2do_h k(n-k)(n-1)}. \tag{2.10}$$

The procedure can easily be seen to have the same probability if we start from ω' and construct ω. It is therefore in detailed balance. (Note that every $g \in \mathrm{O}_h$ has an inverse, so we can always perform the attempted transition in reverse.)

2.3. *Properties of lattice trees*

In this section we consider the properties of lattice trees (for an example of a lattice tree, see figure 2) that can be measured by a canonical simulation of lattice trees in the cubic lattice. Let $x_i(v)$, $i = 1, 2, \ldots, d$, be the ith cartesian coordinate of the vertex v. Let ω be a lattice tree, and let ω_i be the ith vertex of the tree. Then we have the following definitions.

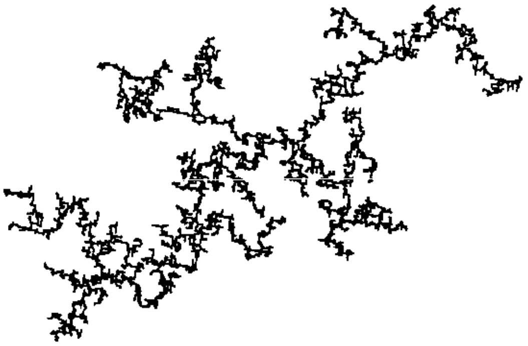

Figure 2. A lattice tree with 5000 vertices.

2.3.1. Mean square radius of gyration. Let $r_n^2(\omega)$ be the square radius of gyration of ω. That is

$$r_n^2(\omega) = \frac{1}{n}\sum_{i=1}^{n}\left[\sum_{j=1}^{d}\left(x_j(\omega_i) - \frac{1}{n}\sum_{k=1}^{n} x_j(\omega_k)\right)^2\right]. \tag{2.11}$$

It is widely believed that there exists a critical exponent ν such that

$$\langle r_n^2 \rangle \sim n^{2\nu} \tag{2.12}$$

where the mean is taken over all conformations of trees with n vertices. $\langle r_n^2 \rangle$ measures a length scale for trees.

2.3.2. Mean span. Let d be the spatial dimension. Then we define $s_n(\omega)$, the mean span of the tree ω, by

$$s_n(\omega) = \frac{1}{d}\sum_{i=1}^{d}\left[\max_{j,k=1}^{n}(|x_i(\omega_j) - x_i(\omega_k)|)\right]. \tag{2.13}$$

If we assume that there is only one length scale for lattice trees, then we expect from equation (2.12) that

$$\langle s_n \rangle \sim n^{\nu} \tag{2.14}$$

where the mean is taken over all the possible conformations of trees.

2.3.3. Longest path in the tree. Let $p_n(\omega)$ be the length of the longest self-avoiding walk in the tree ω. We are immediately interested in the behaviour of $p_n(\omega)$ with n. We can easily find $p_n(\omega)$ in the following way. Let a be any vertex in ω. Let ab be a longest path in the tree starting from a, and let bz be a longest path in the tree starting from b. Then bz is a longest path in the tree (Dewdney 1985a; for an elegant proof, see Dewdney 1985b). The longest path in the tree is an intuitive 'measure' of how far the tree differs from a self-avoiding walk, that is, to what extent is it entropically favourable to add appendages onto the walk rather than making it longer. We define the exponent ρ by

$$\langle p_n \rangle \sim n^\rho \tag{2.15}$$

where the mean is taken over all the possible conformations of trees. If $\rho = 1$, then a lattice tree is essentially a decorated self-avoiding walk. Since self-avoiding walks and trees do not belong to the same universality class, we expect that $\rho < 1$. It is also useful to note that p_n is at least as long as the span, so that $\nu \leqslant \rho$.

2.3.4. Mean branch size. Let $b_n(\omega)$ be the mean size of a branch in ω obtained by deleting an edge in ω with uniform probability. We define the exponent ϵ by

$$\langle b_n \rangle \sim n^\epsilon \tag{2.16}$$

where the average of b_n is taken over all the conformations of trees. We can guess the value of ϵ in the following way. Suppose that the structure of the tree is that of a longest path which contains $\Theta(n^\rho)$ edges, and with $\Theta(n^\rho)$ smaller branches which sprout from the longest path. (Here $\Theta(n^x)$ means at least cn^x and at most Cn^x for some $C \geqslant c > 0$). The total number of edges in these smaller branches is $n - \Theta(n^\rho)$, so each branch has an average size of $\Theta(n^{1-\rho})$. If we pick an edge from the longest path, then we separate the tree into two pieces which each have $\Theta(n)$ edges, since each piece of the longest path will have $\Theta(n^\rho)$ edges and of the same order of side branches. If we pick an edge not on the longest path, then the size of the branch picked is $\Theta(n^\sigma)$, where σ satisfies $0 \leqslant \sigma \leqslant 1 - \rho$. We therefore expect that

$$\langle b_n \rangle \sim \frac{\Theta(n).\Theta(n^\rho) + \Theta(n^\sigma).\Theta(n - n^\rho)}{n} \sim \Theta(n^\rho) + \Theta(n^\sigma). \tag{2.17}$$

Hence we conclude that $\epsilon = \max\{\rho, \sigma\}$, where $\sigma \leqslant (1 - \rho)$. Next, to see that $\epsilon = \rho$, we argue heuristically as follows. Take a branch of the tree of size $\Theta(n^{1-\rho})$ and delete an edge at random; the size of the component of the branch that does not contain the edge which touches the longest path is $\Theta(n^\sigma)$, by definition of σ. Therefore, in a tree of n vertices with a single randomly labelled leaf, if we delete an edge and take the component which does not contain the labelled leaf, then the size of this component is $\Theta(n^{\sigma/(1-\rho)})$. The probability that the labelled edge is in the smaller branch is $\langle b_n \rangle / n = \Theta(n^\epsilon / n)$, by definition of ϵ, so the expected size of the branch which does not contain the labelled leaf is $\Theta(n^{\sigma/(1-\rho)}) = (n - \Theta(n^\epsilon))\Theta(n^\epsilon/n) + \Theta(n^\epsilon)(1 - \Theta(n^\epsilon/n)) = \Theta(n^\epsilon)$. Therefore, $\sigma = \epsilon(1 - \rho) < \epsilon$. Combining this with $\epsilon = \max\{\rho, \sigma\}$, we conclude that

$$\rho = \epsilon. \tag{2.18}$$

2.3.5. Acceptance fraction of algorithms A and B. A pattern theorem for trees (Madras 1989) allows us to estimate the behaviour of the acceptance fraction af the algorithms. It implies that any fixed arrangement of edges in a tree will occur $\Theta(n)$ times on average in a tree of sufficient size. If an arrangement of edges are selected which allows an additional edge to be added to it, then it appears that there are $\Theta(n)$

locations in the tree where a proposed leaf may be added on average. Also, there are $\Theta(n)$ leaves. Therefore, the acceptance fraction of algorithm A, f_n^{A}, satisfies $\liminf_{n\to\infty} f_n^{\mathrm{A}} > 0$, and we in fact expect it to have a limit:

$$\lim_{n\to\infty} f_n^{\mathrm{A}} = C \tag{2.19}$$

where the constant C is dependent on the spatial dimension. The same applies to algorithm B, since $f_n^{\mathrm{B}} \geqslant f_n^{\mathrm{A}}$.

2.3.6. The degrees of the vertices in the tree. The pattern theorem for trees also predicts that the frequency of vertices of degree t_i, $i = 1, 2, \ldots, 2d$, in a tree is $\Theta(n)$. If the number of vertices of degree i in a given tree is given by t_i, then we expect that

$$\lim_{n\to\infty} \frac{\langle t_i \rangle}{n} = \zeta_i \qquad \text{for} \quad 1 \leqslant i \leqslant 2d \tag{2.20}$$

where ζ_i is a constant dependent on d, and where the tree has n vertices.

3. Implementation of the algorithms

The implementation of the algorithms deserves some attention, since big savings in computer time can be obtained by carefully designing the code. The algorithms were both programmed in FORTRAN77. We first describe the programming of algorithm B, since algorithm A can be viewed as a special version of algorithm B, where we only allow the moving of leaves.

Suppose that we are considering a tree in d dimensions with n vertices. The following permanent data structures were set up:

(A) A list of vertices of the tree in an $n \times 3d$ array V. The first d addresses in the ith row of V contain the coordinates of the ith vertex of the tree. The remaining $2d$ addresses $(V(i, d+1), V(i, d+2), \ldots, V(i, 3d))$ are pointers which point to the labels (i.e. row numbers) of vertices which are connected to the ith vertex in the tree. (Some of these addresses will be empty if the degree of the ith vertex is less than $2d$.)

(B) A hash-table $HTAB$ (an $m \times d$ array) (Knuth 1973). Here m is a fixed number ($m = 10n$ is sufficient). The vertices of the tree are hashed into the table using a hash-function (Madras and Sokal 1988) and linear probing. We need to perform three operations on $HTAB$. If v is any vertex, then these operations are
  - (a) $v \in HTAB$?
  - (b) Add v to $HTAB$.
  - (c) Remove v from $HTAB$.

  It is easy to write fast subroutines to perform these tasks on $HTAB$ (see Knuth (1973) for details). The advantage of using the hash-table is that we can use operation (a) to perform an efficient check that a proposed new configuration is self-avoiding (and therefore a tree).

(C) A list of labels $SMALL$ (an $n/2 \times 2$ array). We store the labels of the branch that we attempt to move in one of the two columns of $SMALL$.

(D) A list A (an n-element vector). A is initialized to contain a zero in each address. We shall use A to identify which vertices are not in the branch that we attempt to move.

We can now implement algorithm B in the following way:

(B1–B2) Choose an edge at random from the tree. We do this by using rejection: We pick $i \in \{1,2,\ldots,n\}$ and $j \in \{d+1, d+2, \ldots, 3d\}$ at random. If $V(i,j)$ is empty, then we try again; otherwise, we pick the edge whose endpoints are given by i and $V(i,j)$. On average, the number of tries needed to find an edge is d. We delete this edge, separating the tree into two subtrees or branches.

(B3) Perform a depth-first or breadth-first search (Wilson and Watkins 1990) on the two subtrees *simultaneously*, starting at the endpoints of the chosen edge, to find the branch which we will attempt to move. This amounts to searching alternatingly on the two subtrees, reading the labels of the vertices encountered into $SMALL(.,i)$, choosing $i = 1$ or 2 for each of the two subtrees. As soon as one of the two subtrees has been completely searched, we know that it must be the smaller subtree (branch), and we pick i to be 1 or 2, whichever corresponds to the branch. The labels of the vertices in the branch are then written in the ith column of $SMALL$, in order as we detected them by the search. Once we have determined the labels of the vertices of the branch, we update the list A by putting a 1 in each address which is a label of the vertices on the branch. This is a very convenient arrangement. By simply querying A, we can detect whether a vertex is in the branch, or in the rest of the tree. It is also easy to reset A to all contain all zeros: The addresses of A which contains non-zero elements are the labels of the vertices on the branch, which are listed in $SMALL$. Note that the amount of 'work' performed in this step is of the order the size of the smaller subtree.

(B4) Pick a vertex x on the branch, and another vertex y on the rest of the tree. The vertex x can be picked uniformly from the list in $SMALL$. We pick the vertex y by rejection from V, querying the list A to determine whether the vertex is in the branch. Since the number of vertices in the branch is at most $n/2$, the average number of attempts is at most 2.

(B5) The branch is now rotated by applying a randomly chosen element of the octahedral group to it. We then translate the rotated branch so that the vertex x on it will be a nearest neighbour of the vertex y on the larger subtree. This is one of $2d$ possible positions, chosen at random. The proposed tree is now formed by adding the edge between the vertices x and y. The last step in the algorithm is to check for self-intersections in the proposed tree. This is easily done by querying the hash-table $HTAB$ and the list A. A particularly effective way of performing the check is to start at the vertices x and y, alternatingly performing a breadth-first search on the two subtrees. Since the two subtrees are in closest contact at x and y, it seems likely that an intersection will occur near these vertices, if it occurs at all. We perform step B5 efficiently by rotating the vertices on the smaller tree one by one, testing for intersections each time we calculate the new coordinates of another vertex.

If the conformation is not self-avoiding, then we reject it, we reset the list A to its null-values and start at step $B1$. Otherwise we accept the new tree, update the vertices in V and update the hash-table by removing the old vertices from it and adding the new vertices to it. Lastly, we reset the list A to its null-values we take data.

The size of an average branch on the tree is expected to grow as n^ϵ (recall (2.16)). The searches (depth-first and breadth-first) which we perform to identify the branches take therefore $O(n^\epsilon)$ operations. Updating the old tree in case of a successful attempt, and updating the hash-table and other lists also takes $O(n^\epsilon)$ operations. We therefore expect the average amount of work per iteration in the case of algorithm B to be $O(n^\epsilon)$.

The implementation of algorithm A is simpler than that of algorithm B. We retain the data structures V and $HTAB$, but we note now that we do not need $SMALL$ and A, both of which are necessary to search and store branches in algorithm B. An implementation of algorithm A would be:

(A1, A2) Choose an edge from the tree, in exactly the same manner as was done in algorithm B. If the edge has one vertex which is of degree one, then delete it, else we reject the attempt, and try again to select an edge.

(A3, A4) Pick a vertex on the tree, and one of its neighbouring sites with uniform probability, and attempt to add an edge between the two sites. If the neighbouring site is already occupied, then we reject the attempt, and start again at step A1. Otherwise we add the edge, and update the tree and the hash-table. We take data and start again from step A1.

The amount of work per attempted iteration of algorithm A is evidently $O(1)$. At every point in the algorithm we only deal with at most two vertices, and there is no explicit n-dependence in any of the operations performed.

The number $n^2 r_n^2(\omega)$ of a tree ω with n vertices, where r_n^2 is the square radius of gyration, can be calculated using only integer operations. We also note that after every successful attempted transition we can 'update' $n^2 r_n^2$ by simply subtracting the old vertices and adding the new. The span of a tree, s_n, the longest path p_n and the degrees of the vertices of the tree take $O(n)$ operations to calculate. In view of these facts, it seems best to sample these properties of the tree every n attempted transitions, for it is not sensible to spend $O(n)$ operations calculating the span of a tree while less than n iterations seperates it from the last configuration that was sampled: We may end up spending more time calculating the span and other properties than updating the tree into new regions of configuration space. The acceptance fraction, the mean branch size, and $n^2 r_n^2$ can be updated after every attempted iteration; we therefore calculated these numbers as block-averages over blocks of data of length n.

The data were written as a stream of numbers during the runs and stored for analysis. We used a time series analysis to find the autocorrelation times for each of the variables (Madras and Sokal 1988). The number of iterations performed for each run was typically $10\,000n$, which gives us 10 000 data points, and error bars of about one per cent on the calculated variables. The program proved to be extraordinarily efficient; for example, the results in four dimensions were obtained by a total of 11.5 hours of computer time on a DEC5000 workstation (with RISC-technology). At each value of n the initial tree was chosen to be a straight sequence of edges with no branches. Algorithm B was applied to this initial tree until all initial bias disappeared from the data. This relaxation was very fast for small n, but took up to 5×10^5 iterations for longer trees.

4. Numerical results

4.1. Comparing algorithms A and B

It is not immediately obvious that algorithm B will perform better than algorithm A in a numerical test. This is because algorithm A can perform $O(m)$ attempted iterations for every one attempted move of a branch of m vertices by algorithm B.

In this section we shall compare these algorithms by comparing the autocorrelation times in units of the amount of work performed by the CPU. A convenient measure of 'work' is the CPU time used by the algorithms in generating the trees, excluding the time required for taking any measurements of the properties of the trees. By comparing the CPU time per iteration of algorithm A and algorithm B, we can find a ratio $r_{\rm A}^{\rm B}$ which is the average number of attempted transitions in algorithm A per one attempted transition in B (for the same amount of CPU time). If we calculate the autocorrelation times of algorithm B in units of n attempted iterations, then we should express it in units of $r_{\rm A}^{\rm B}n$ attempted iterations for algorithm A. The algorithms are then compared by taking the ratio of the autocorrelation time of algorithm A (in units of $r_{\rm A}^{\rm B}n$ attempted iterations) to that of algorithm B (in units of n attempted iterations). Since the number of 'independent observations' in the data stream of the algorithm is inversely proportional to τ, we are in fact calculating the ratio of the number of independent observations obtained by algorithm B for every independent observation by algorithm A (for the same amount of CPU time).

Table 1(*a*). A comparison between the autocorrelation times with respect to the mean square radius of gyration of algorithms A and B in two and three dimensions. The autocorrelation times are in units of $r_{\rm A}^{\rm B}n$ and n attempted iterations for algorithms A and B respectively.

n		$r_{\rm A}^{\rm B}$	τ_n (A)	τ_n (B)	M(B,A)
Two dimensions	25	6.2	4.34	1.51	0.60
	50	7.4	10.4	1.92	0.43
	100	9.0	34	2.9	0.29
	200	11.7	76	3.8	0.23
	400	15.6	124	5.7	0.21
Three dimensions	25	7.0	2.7	0.93	0.59
	50	8.2	7.1	1.00	0.38
	100	9.2	16	1.03	0.25
	200	10.6	35	1.6	0.21

In table 1(*a*) we compare the algorithms in two and three dimensions with respect to the mean square radius of gyration. The results for the other global properties (mean span, mean longest path, and mean branch size) are very similar. In the first column we list the number of vertices of the tree under consideration. The second column contains $r_{\rm A}^{\rm B}$, the number of attempted iterations of algorithm A for every attempted iteration in algorithm B. The third and fourth columns contain the autocorrelation times of algorithms A and B respectively, and in the last column we list M(B,A), the square root of the ratio of the autocorrelation times of algorithm B to that of algorithm A. We can expect that the confidence intervals in algorithm B will be smaller than that of A by this factor. We see that algorithm B outperforms algorithm A significantly. In two dimensions, for $n = 400$, we can expect error bars

Table 1(b). The autocorrelation times of algorithm B, measured from the mean square radius of gyration and in units of n attempted iterations. Error bars are standard deviations.

n	τ_{r^2} (2D)	τ_{r^2} (3D)	τ_{r^2} (4D)	τ_{r^2} (8D)	τ_{r^2} (9D)l
25	1.509(77)	0.927(40)	0.794(30)	0.728(31)	0.702(27)
50	1.92(12)	1.003(43)	0.828(31)	0.685(22)	0.682(26)
100	2.90(30)	1.033(49)	0.750(24)	0.642(21)	0.676(26)
200	3.80(50)	1.57(14)	0.840(36)	0.621(20)	0.638(24)
300	4.36(37)	1.230(68)	0.783(30)	0.615(24)	0.610(20)
400	4.79(42)	1.271(60)	0.776(30)	0.622(30)	0.589(19)
600	6.00(60)	1.249(59)	0.729(28)	0.612(23)	0.585(19)
800	7.08(73)	1.313(67)	0.752(29)	0.585(19)	0.611(23)
1200	8.25(93)	1.48(11)	0.708(27)	0.566(18)	0.590(23)
1600	11.5(20)	1.452(69)	0.732(28)	0.557(17)	0.570(22)

which are about five times smaller had we run algorithm A (for the same amount of work).

It is striking how the superiority of algorithm B improves with n, the number of vertices in the tree. A plot of the autocorrelation time against n on a log–log scale shows linear behaviour; we therefore expect that τ will grow as a power of n. A simple weighted least squares fit to the data in the table (where we also take into account data for algorithm B for trees consisting of up to 1600 vertices, as listed in table 1(b) implies that

$$\tau_{r^2}(A) \sim n^{1.3\pm0.2} \tag{4.1}$$
$$\tau_{r^2}(B) \sim n^{0.45\pm0.04} \tag{4.2}$$

in two dimensions, and in three dimensions,

$$\tau_{r^2}(A) \sim n^{1.3\pm0.1} \tag{4.3}$$
$$\tau_{r^2}(B) \sim n^{0.12\pm0.03}. \tag{4.4}$$

where the statistical errors are 95% confidence intervals. Here we have measured the autocorrelation times in units of n attempted iterations for algorithm B, and in $r^{\mathrm{B}}_{\mathrm{A}}n$ iterations for algorithm A. (Note that since the mean CPU time per attempted iteration in algorithm B is believed to go as n^{ϵ}, if we want to measure autocorrelation times in CPU seconds, then we must add $(1+\epsilon)$ to all of the exponents above).

Comparing equations (4.2)–(4.4) suggests that algorithm B performs better in three dimensions than in two, with the autocorrelation times growing slower with the number of vertices in the trees in three dimensions. In contrast to this, algorithm A has autocorrelation times which depend similarly upon n in two and three dimensions.

In higher dimensions, an examination of table 1(b) suggests that the data points are only weakly correlated, if at all. In retrospect, the algorithm is even better than we suspected. Assume that the path between two given vertices on the tree has, on average, $\Theta(n^{\epsilon})$ edges. The correlation between these points are destroyed in the algorithm if we pick an edge on the path between the vertices and perform a successful transition. In the mean field approximation, the probability of this event is roughly $\Theta(n^{\epsilon-1})$. After m iterations, the correlation between these vertices are

$$S_m \sim (1-\Theta(n^{\epsilon-1}))^m. \tag{4.5}$$

The exponential autocorrelation time (of a global quantity X, such as the path length between the vertices, or the Euclidean distance between the vertices) can be esimated from this expression (for a similar analysis involving the pivot algorithm and the self-avoiding walk, see Madras and Sokal (1988)). We find

$$\tau_{e,X}(B) = \Theta(n^{1-\epsilon}) \quad \text{iterations.} \tag{4.6}$$

We calculate the mean field value of ϵ in appendix A to be $1/2$. Therefore, the best possible behaviour for the algorithm is when

$$\tau_{e,X}(B) = \Theta(n^{1/2}) \quad \text{iterations.} \tag{4.7}$$

(If we express this in units of n iterations, as we do in equations (4.1)–(4.4), then we have $\tau_{e,X}(B) \sim n^{-0.5}$ for the best possible behaviour of algorithm B.) Thus, the data in table 1(*b*) tell us that τ is between $\Theta(\sqrt{n})$ and $\Theta(n)$ iterations in four and more dimensions. In general, the autocorrelation times will depend on other factors, such as the acceptance fraction of moves involving large subtrees. We discuss this in the next section.

Table 2(*a*). *The acceptance fraction of algorithm* A, f_n^A, *in two and three dimensions.* Error bars are standard deviations.

n	f_n^A (2D)	f_n^A (3D)
25	0.12500 (25)	0.20790 (26)
50	0.10685 (22)	0.18867 (22)
100	0.09860 (20)	0.17891 (20)
200	0.09340 (20)	0.17360 (16)
400	0.09180 (20)	—

4.2. *The acceptance fraction of proposed moves*

As explained in section 2.3, we expect that the acceptance fraction of algorithm A will converge to a constant value, as n tends to infinity. The acceptance fractions of algorithm A in two and three dimensions are listed in table 2(*a*). We obtained the data over $40000 n r_A^B$ observations, where the r_A^B are listed in table 1(*a*). To obtain an estimate of the limit C in equation (2.19), we assume that

$$f_n^A = C + \gamma n^{-\delta} \tag{4.8}$$

where C, γ and δ are parameters which we should obtain by a weighted least squares fit to the data in table 2(*a*). This form assumes possible non-analytic terms in f_n^A, and it seeks to estimate the largest of them. From a three-parameter weighted least squares fit to the data we find that

$$\lim_{n\to\infty} f_n^A = \begin{cases} 0.0897 \pm 0.0007 & \text{if } d = 2 \\ 0.168 \pm 0.002 & \text{if } d = 3. \end{cases} \tag{4.9}$$

Table 2(*b*). The acceptance fraction of algorithm B, f_n^{B}, in two, three, four, eight and nine dimensions. Error bars are standard deviations.

n	f_n^{B} (2D)	f_n^{B} (3D)	f_n^{B} (4D)	f_n^{B} (8D)	f_n^{B} (9D)
25	0.22596 (92)	0.4301 (11)	0.5733 (11)	0.80441 (81)	0.82925 (76)
50	0.17550 (70)	0.36767 (76)	0.51964 (74)	0.78764 (60)	0.81453 (56)
100	0.14950 (50)	0.32980 (52)	0.48130 (53)	0.77509 (44)	0.80570 (40)
200	0.13650 (30)	0.30619 (39)	0.45634 (39)	0.76852 (31)	0.80042 (29)
300	0.13182 (29)	0.29736 (32)	0.44613 (33)	0.76545 (26)	0.79876 (24)
400	0.12912 (25)	0.29342 (28)	0.44063 (28)	0.76356 (22)	0.79697 (20)
600	0.12755 (20)	0.28848 (25)	0.43371 (23)	0.76148 (18)	0.79585 (17)
800	0.12651 (17)	0.28622 (21)	0.43015 (20)	0.76037 (16)	0.79516 (15)
1200	0.12505 (15)	0.28325 (18)	0.42609 (15)	0.75900 (13)	0.79428 (12)
1600	0.12466 (14)	0.28236 (16)	0.42396 (15)	0.75843 (11)	0.793951 (97)

The error bars are 95% confidence levels. In both cases, the value of δ was close to 1 (1.02 ± 0.05 and 0.94 ± 0.05 for $d = 2$ and $d = 3$ respectively).

We expect that algorithm B will have a higher acceptance fraction than algorithm A, since we will have all the succesful transitions which occur in algorithm A, as well as succesful transitions involving branches of different sizes. The acceptance fractions of algorithm B are listed in table 2(*b*) for two, three, four, eight and nine dimensions. Assuming equation (4.8) again, we find that

$$\lim_{n\to\infty} f_n^{\mathrm{B}} = \begin{cases} 0.1229 \pm 0.0003 & \text{if } d = 2 \\ 0.2761 \pm 0.0005 & \text{if } d = 3 \\ 0.4114 \pm 0.0004 & \text{if } d = 4 \\ 0.7920 \pm 0.0003 & \text{if } d = 9. \end{cases} \tag{4.10}$$

(We discuss the case $d = 8$ below). We find that δ is close to 1 only in two dimensions ($\delta = 0.98 \pm 0.04$). In three dimensions we find that $\delta = 0.79 \pm 0.02$, in four dimensions $\delta = 0.62 \pm 0.02$ and in nine dimensions $\delta = 0.71 \pm 0.03$. The rate of approach to the limit is considerably slower than n^{-1} in these three cases.

We see in tables 2(*a*) and 2(*b*) that the confidence intervals on our data points are very small (a few tenths of a per cent at most). We expect that the presence of a term that goes to zero slower than a power of n will probably spell difficulty for our fitting algorithm. In fact, in eight dimensions Newton's method fails to converge for the data in table 2(*b*). This suggests that there is a term which converges slower than a power of n. One way of dealing with this is to assume that

$$f_n^{\mathrm{B}} = C + \gamma |\log n|^{-\delta}. \tag{4.11}$$

A weighted least squares analysis gives

$$\lim_{n\to\infty} f_n^{\mathrm{B}} = 0.7484 \pm 0.0007 \qquad \text{if} \quad d = 8. \tag{4.12}$$

We obtain $\delta = 2.12 \pm 0.07$. (To check assumption (4.11) we plotted $\log(f_n^{\mathrm{B}} - C)$ against $\log|\log n|$ in eight dimensions using C from equation (4.10). The result was a straight line. The analogous plot in nine dimensions was strongly curved.)

The acceptance fraction of algorithm B increases significantly with the dimension, which suggests that the algorithm is more efficient in higher dimensions. To examine

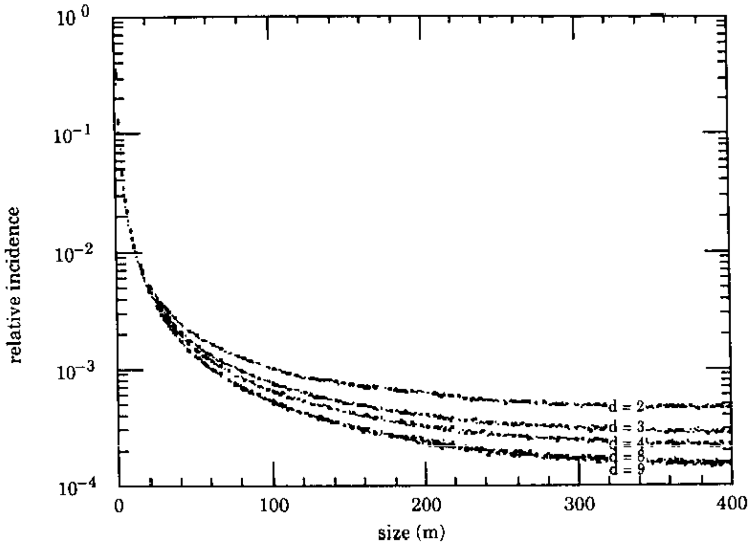

Figure 3. The size distribution of proposed branches in the algorithm. The data is from trees of size 800.

this suggestion more precisely, we consider the mean size of the proposed branches and the mean size of the accepted branches in different dimensions for fixed n (the number of vertices in the tree). In figure 3 we plot the number of proposed branches of size m against m on a linear-logarithmic scale. The number of iterations was 8000000 and n was 800. We see that the biggest branches are proposed in two dimensions, and that the incidence of big proposed branches declines as we increase the number of dimensions. In fact the mean size of a proposed branch declines from 51.4 in two dimensions, through 35.7 in three dimensions to 29.9 in four dimensions, 23.6 in eight dimensions and 23.2 in nine dimensions. While these numbers at first sight seem to indicate that algorithm B will be more successful in lower dimensions, it is really the acceptance fraction of the larger branches which will make the difference. In figure 4 we plot the acceptance fraction of branches of size m, $f_n^B(m)$, against m for algorithm B. We see that the acceptance fraction for a given value of m increases rapidly with dimension, reflecting the behaviour of the acceptance fraction integrated over m. Combining the above data, we calculated the mean size of the *accepted* branches: This is only 2.45 vertices in two dimensions, and increases through 4.85 in three dimensions to 8.61 in four dimensions, 19.61 in eight dimensions and 20.26 in nine dimensions (for eight and nine dimensions these are 83% and 87% of the proposed branch sizes respectively). Thus, even though the mean size of the proposed branches decreases with dimension, we find that the mean size of branches which are involved in successful transitions increases with the number of dimensions. We expect therefore that algorithm B will perform more effectively in higher dimensions. These effects can also be seen in the scaling of the autocorrelation time with n, which we discussed in section 4.1.

4.3. The mean square radius of gyration and the mean span

The mean square radius of gyration, $\langle r^2 \rangle$, and the mean span, $\langle s \rangle$, of trees measure a universal length scale defined by the exponent ν on the lattice. To estimate ν,

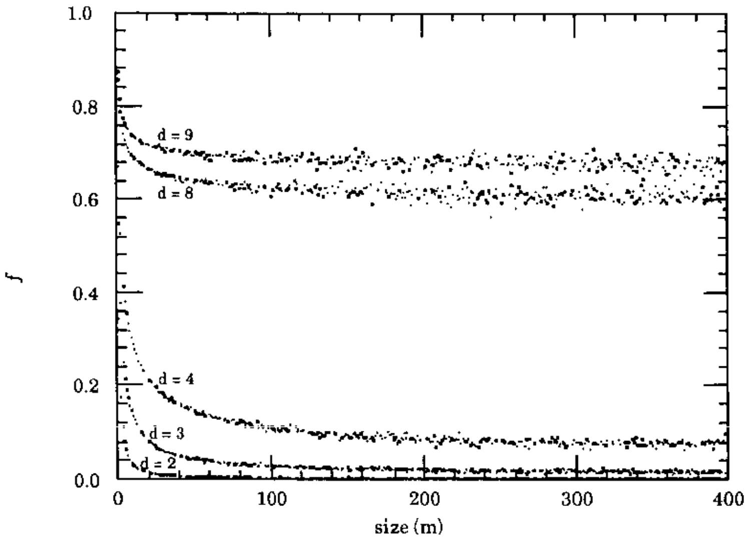

Figure 4. The acceptance fraction of branches of size m. The higher the number of dimensions, the higher the acceptance fraction. More interesting, the curves do not go to zero with increasing branch size, but reach a plateau. It is almost as likely for a branch of size 400 to be accepted as a branch of size 100. The data is from trees of size 800.

we begin with the scaling relation $\langle r^2 \rangle \sim An^{2\nu}$. We can write this asymptotic relation as an equality with (infinitely many) correction-to-scaling terms: $\langle r^2 \rangle = An^{2\nu}(1 + bn^{-\Delta} + \cdots)$. Our job is to fit a curve $\langle r^2 \rangle = f(n)$ to the data. There are two obvious choices for the form of this curve: either we should eliminate all of the correction-to-scaling terms, giving the two-parameter family of curves

$$\langle r^2 \rangle = An^{2\nu} \tag{4.13}$$

or else we should eliminate all but the dominant correction, giving the four-parameter family

$$\langle r^2 \rangle = An^{2\nu}(1 + bn^{-\Delta}). \tag{4.14}$$

The form (4.13) is appropriate if the values of n under consideration are all large enough so that the actual corrections to scaling are smaller than the statistical errors in the data; thus, if a log–log plot of the data is clearly linear, then we should be satisfied that we are in the asymptotic regime and work with the form (4.13). (Of course, one should also check fits of the form (4.14) as a standard procedure, even if the plot appears very straight to the eye.) On the other hand, if this plot shows strong curvature, then our first choice should be the form (4.14). Of course, there is no guarantee that the best curve of the form (4.14) will reflect the true value of Δ, since we do not know the size of the omitted correction terms (when n is small, the omitted terms are large, so it is hard to see Δ from data corresponding to small values of n; while when n is large and the omitted terms are small, then the included term $Bn^{-\Delta}$ is also small). It is very likely that one ends up estimating some effective exponent Δ_{eff} which has no intrinsic physical meaning. Thus we take the cautious view that unless the data speaks very strongly to the contrary, the parameter Δ in

(4.14) is no more than an aid to the extrapolation of a finite amount of data into the $n \to \infty$ asymptotic regime. In the data presented in this paper, we find no strong evidence for any particular values of Δ; instead, the best that we feel justified in doing is testing the consistency of other researchers' values of Δ with our data.

In practice, one would like to perform linear least squares regressions. This is accomplished by taking the logarithms of (4.13) and (4.14). The scaling assumptions are then

$$\log\langle r^2\rangle = a + 2\nu \log n \tag{4.15}$$
$$\log\langle r^2\rangle = a + bn^{-\Delta} + 2\nu \log n. \tag{4.16}$$

If we fix Δ in (4.16), then ν is obtained by a linear regression. The data points from the smallest trees will suffer most from corrections, so we attempt to minimize their influence by throwing away data points from the smaller trees. We estimate the parameter ν by performing weighted least squares regressions for the model (4.15), in which the weights are determined by the estimated error bars. The best choice of n_{min} (the smallest value of n that we do not throw away) is determined by the associated χ^2 statistic: the (weighted) sum of the squares of the distances of the data points from the fitted curve (see e.g. Silvey 1975). When the model is correct, this statistic has a χ^2_k distribution (here k, the number of degrees of freedom, equals the number of data points used minus the number of model parameters that we are trying to estimate). Since our scaling assumptions are imperfect, a typical regression will provide best estimates of our parameters in a biased fashion: there is a systematic error present. We can attempt to estimate this error, where possible, by comparing results from two different scaling assumptions. In most cases we can compare the results from a two-parameter fit (4.15) to a three-parameter fit (4.16).

Table 3. The mean span of trees. Error bars are standard deviations.

n	s_n (2D)	s_n (3D)	s_n (4D)	s_n (8D)	s_n (9D)
25	6.842 (32)	4.2107 (59)	3.2197 (44)	1.9026 (21)	1.7510 (19)
50	11.147 (22)	6.4020 (88)	4.7449 (58)	2.7970 (29)	2.5883 (25)
100	17.915 (42)	9.471 (13)	6.7354 (75)	3.8502 (35)	3.5669 (32)
200	28.188 (73)	13.773 (20)	9.3560 (99)	5.1000 (44)	4.7410 (39)
300	36.79 (11)	17.039 (23)	11.264 (12)	5.9578 (49)	5.5208 (43)
400	44.67 (14)	19.821 (28)	12.835 (13)	6.6149 (52)	6.1331 (47)
600	58.60 (21)	24.440 (35)	15.378 (16)	7.6392 (61)	7.0699 (53)
800	69.70 (26)	28.358 (39)	17.444 (17)	8.4546 (64)	7.7848 (58)
1200	90.48 (36)	34.929 (51)	20.925 (20)	9.6698 (71)	8.9017 (64)
1600	109.82 (55)	40.386 (57)	23.657 (22)	10.6155 (77)	9.7781 (69)

We examine the mean square radius of gyration (with scaling assumptions (4.15) and (4.16)) and the mean span (with scaling assumptions analogous to (4.15) and (4.16)) for each of the dimensions considered in our simulations. We list our results in tables 3 and 4. In what follows we give all statistical error bars as 95% confidence intervals.

Table 4. The mean square radius of gyration of trees. Error bars are standard deviations.

n	r_n^2 (2D)	r_n^2 (3D)	r_n^2 (4D)	r_n^2 (8D)	r_n^2 (9D)
25	8.181 (35)	4.673 (14)	3.7028 (86)	2.8717 (46)	2.8108 (42)
50	19.43 (10)	9.054 (28)	6.460 (16)	4.5208 (69)	4.3829 (65)
100	46.89 (29)	17.823 (55)	11.293 (26)	6.8984 (96)	6.6559 (91)
200	112.40 (80)	34.75 (12)	19.686 (45)	10.356 (14)	9.948 (13)
300	186.3 (15)	51.66 (19)	27.263 (60)	13.053 (16)	12.479 (15)
400	271.2 (22)	68.73 (26)	34.312 (76)	15.356 (18)	14.597 (16)
600	474.7 (44)	102.51 (37)	47.72 (10)	19.269 (21)	18.238 (19)
800	654.0 (66)	136.25 (51)	60.31 (13)	22.635 (23)	21.298 (21)
1200	1093.0 (13)	203.97 (83)	84.64 (17)	28.211 (26)	26.503 (24)
1600	1622.0 (22)	270.6 (11)	107.27 (22)	33.008 (29)	30.862 (26)

4.3.1. The mean square radius of gyration.

Two dimensions. A log–log plot of the mean square radius of gyration against n (figure 5) shows no evidence of curvature, but a two-parameter fit to the data gives a very poor χ^2 statistic (a minimum of 28.5 at $n_{\min} = 25$ with 8 degrees of freedom). A close examination of figure 5 indicates that the point at $n = 600$ seems to be off the regression line. If we examine our data without this point, then $\chi_7^2 = 12.53$, ($< 95\%$). We conclude that the data point at $n = 600$ is an outlier and we estimate that $\nu = 0.6326 \pm 0.0019$.

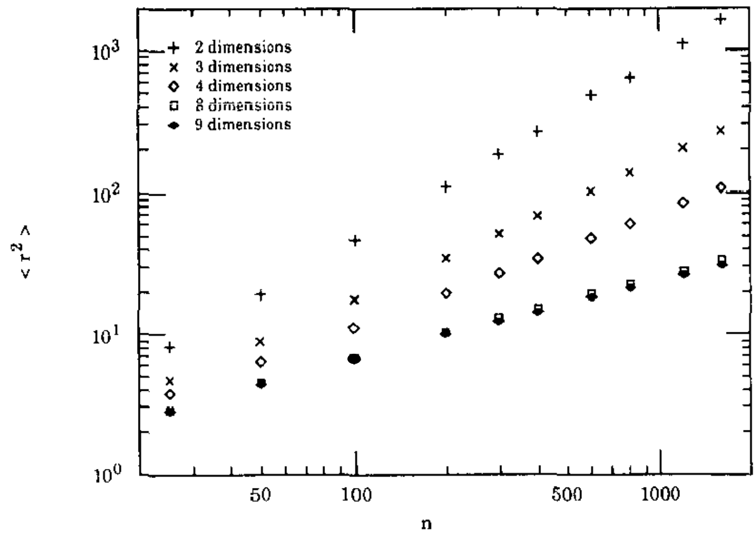

Figure 5. The mean square radius of gyration of trees plotted against n on logarithmic axes.

If a three-parameter analysis of the data is performed, where we assume equation (4.16), then we are unable to minimize the χ^2 statistic by tuning Δ. Instead, we find that χ^2 decreases slowly as we take Δ to zero. It seems therefore plausible that there is a very weak correction present. To assess the effect of this correction on our results, we can fix Δ at some value, and find a best estimate for ν (it would not be unreasonable to choose $\Delta = 0.5$: while this is not the best value, it nevertheless *indicates* the size of a systematic error in the two parameter fit by showing the dependence of ν on variations of Δ; if there are no corrections, then the amplitude of the correction should be near zero, and ν will be consequently independent of Δ). Our regression analysis with $\Delta = 0.5$ gives $\chi^2_6 = 4.65$ ($< 75\%$). The best estimate is $\nu = 0.6422 \pm 0.0071$. We obtain our best estimate of ν by taking the average of the values produced by the two regressions. A systematic error is estimated by the difference between the best estimate (of ν) and either of the two regressions. The statistical error is taken as the maximum from the two regressions. We find

$$\nu_{\mathrm{r}} = 0.6374 \pm 0.0071 \pm 0.0048 \qquad (4.17)$$

where the format is *best value* $\pm$ *95% confidence interval* $\pm$ *systematic error*. The subscript r is to emphasize that we obtained this value from the mean square radius of gyration.

Three dimensions. An examination of the data in figure 5 shows a slight deviation from a straight line at the smaller n values: the curve is slightly convex. A two-parameter fit is good when $n_{min} = 200$ ($\chi^2_5 = 1.10$, $< 50\%$). The value of the exponent is $\nu = 0.4939 \pm 0.0021$. The same behaviour is observed for a three-parameter fit as in two dimensions: the χ^2 statistic decreases with Δ. We follow the same strategy here; we assume $\Delta = 0.5$ to gauge a systematic error in our analysis. A fit with $n_{\mathrm{min}} = 25$ is good ($\chi^2_6 = 4.5$, $< 50\%$). The result for the exponent is $\nu = 0.4981 \pm 0.0031$. We can now make our best estimate:

$$\nu_{\mathrm{r}} = 0.4960 \pm 0.0031 \pm 0.0021. \qquad (4.18)$$

Four dimensions. We find that there are in general strong corrections to scaling here. The plot (figure 5) looks very straight to the eye, but a two-parameter fit is not good unless we take $n_{\mathrm{min}} = 400$. Then $\chi^2_3 = 5.91$, $< 90\%$. This gives $\nu = 0.4117 \pm 0.0019$. A three-parameter regression behaves as in two and three dimensions, so we took $\Delta = 0.5$. For $n_{\mathrm{min}} = 200$ we find $\chi^2_4 = 2.7$, $< 50\%$. This gives $\nu = 0.4282 \pm 0.0079$. Comparing these results, we find our best estimate

$$\nu_{\mathrm{r}} = 0.4200 \pm 0.0079 \pm 0.0083. \qquad (4.19)$$

Eight dimensions. In eight dimensions we expect logarithmic corrections to scaling. Examination of the data (figure 5), however, shows that it is very difficult to decide whether we see a power law correction as opposed to a logarithmic correction. Therefore, we analyse the data here as in lower dimensions, and we treat the corrections to scaling as a hindrance to the extraction of the critical exponents. (It seems that n is not large enough to expose a logarithmic correction. The power law assumption is only an artifact used here to aid our numerical extrapolation.) The plot (figure 5) shows a slight curvature. A two-parameter fit does not provide satisfactory results here. A three-parameter fit with $\Delta = 0.556$ and $n_{\mathrm{min}} = 25$ gives $\chi^2_6 = 6.5$, $< 75\%$. This gives $\nu = 0.2651 \pm 0.0010$. To find a systematic error, we examine the sensitivity of ν to the parameter Δ. A convenient *ad hoc* approach is to change the value of Δ until the value of the χ^2 statistic has *doubled*, and then to vary n_{min} to find ν for these (now fixed) values of Δ. Our reasoning is as follows. The original χ^2 statistic measures how well we can choose a value of Δ to make the model (4.16) fit the data. The value of Δ which is 'best' in this sense may be far from the 'true' value, so we must guard against the misspecification of Δ in case the resulting ν is too sensitive to it. To get an idea of the systematic error, then, we also want to examine fits when we do not have the freedom to vary Δ. Our criterion chooses new values of Δ where the fit is decidedly poorer, but where we can still find reasonable fits by varying $n_{\min}$. In the present case, the χ^2 statistic is doubled at Δ equal to 0.705 (where we find $\nu = 0.2682 \pm 0.0016$ ($n_{\min} = 100$)) or 0.422 (where we find $\nu = 0.2594 \pm 0.0017$ ($n_{\min} = 50$)). Comparing the results, we find that our best estimate is

$$\nu_r = 0.2651 \pm 0.0010 \pm 0.0057. \tag{4.20}$$

The statistical error is the 95% confidence interval obtained at $\Delta = 0.484$, where we obtained the best value for ν. The systematic error is the maximum difference between the best estimate of ν, and the estimates obtained from the fits where the χ^2 statistic was doubled. This answer is close to the expected mean field result, which is 0.25.

Nine dimensions. The situation is similar to eight dimensions. We find with $\Delta = 0.505$ that $\nu = 0.2560 \pm 0.0010$ ($\chi_6^2 = 5.1$, $< 50\%$ and $n_{\min} = 25$). The χ^2 statistic doubles if $\Delta = 0.614$ ($\nu = 0.2608 \pm 0.0008$ ($n_{\min} = 25$)) and if $\Delta = 0.398$ ($\nu = 0.2487 \pm 0.0012$ ($n_{\min} = 25$)). Comparing these results we find that

$$\nu_r = 0.2560 \pm 0.0010 \pm 0.0073. \tag{4.21}$$

4.3.2. The mean span.

Two dimensions. A two-parameter fit gives a poor χ^2 statistic (a minimum of 18.6 at $n_{\min} = 100$ with 6 degrees of freedom). This value is over the 99.5 percentile range, so that we do not have much confidence in this fit. A close examination of the data shows that the point at $n = 600$ seems to be off the regression line in two dimensions. If we reanalyse our data with this point deleted, then $\chi_5^2 = 8.4$, ($< 90\%$), a dramatic inprovement, where $n_{\min} = 100$ as before. We therefore conclude that the data point at $n = 600$ is an outlier. We estimate $\nu = 0.6533 \pm 0.0028$ from this regression.

As an alternative, we can instead perform a three-parameter analysis of our data. If we ignore the data point at $n = 600$, then the best χ^2 statistic is obtained at $\Delta = 0.915$ with $n_{\min} = 25$. We find $\chi_5^2 = 6.9$ ($< 90\%$). Here, we estimate $\nu = 0.6444 \pm 0.0019$. We can now calculate a best value of ν by taking the average of the values produced by the two methods, and a systematic error by considering the difference between this average and either of the two regressions. We take the statistical error as the maximum from the two regressions. We find

$$\nu_s = 0.6489 \pm 0.0028 \pm 0.0045. \tag{4.22}$$

The subscript s is to emphasize that this value was obtained from the mean span data.

Three dimensions. The χ^2 statistic in a two-parameter linear regression decreases fast in value with increasing $n_{\min}$. For $n_{\min} = 300$ we find that $\chi_4^2 = 7.72$ ($< 90\%$). The value of the exponent is $\nu = 0.5157 \pm 0.0020$. A three-parameter analysis is very good. We find a minimum in χ^2 at $\Delta = 0.734$ ($\chi_6^2 = 1.26$, $< 50\%$, $n_{\min} = 25$).

The exponent is $\nu = 0.5034 \pm 0.0018$. We can now calculate the best value of ν from the span data:

$$\nu_s = 0.5096 \pm 0.0020 \pm 0.0062. \tag{4.23}$$

Four dimensions. We find that there are strong corrections to scaling here. A two-parameter fit is found to be unsuitable for any $n_{\min}$ in this case. The χ^2 statistic decreases fast with increasing $n_{\min}$, but never to an acceptable value. In contrast to this, a three-parameter fit with $\Delta = 0.78$ works well ($\chi_6^2 = 8.7$, $< 90\%$, and $n_{\min} = 25$). The result for the exponent is $\nu = 0.4303 \pm 0.0013$. To find a systematic error, we examine the sensitivity of ν to the parameter Δ. The χ^2 statistic is doubled at $\Delta = 0.66$ and $\Delta = 0.91$. At $\Delta = 0.66$ we find that $\nu = 0.4254 \pm 0.0020$ ($n_{\min} = 50$) and at $\Delta = 0.91$ we find that $\nu = 0.4342 \pm 0.0016$ ($n_{\min} = 50$). A comparison gives

$$\nu_s = 0.4303 \pm 0.0013 \pm 0.0049. \tag{4.24}$$

where the statistical error is the 95% confidence interval obtained

Eight dimensions. We analyse the data here as for the mean square radius of gyration in eight dimensions. Two-parameter fits are poor, so we assume a power law correction and proceed as in four dimensions for the mean span. The best fit is obtained for $\Delta = 0.484$ and $n_{\min} = 50$ ($\chi_5^2 = 14.5$, $< 99\%$). This gives $\nu = 0.2878 \pm 0.0022$. We examine the dependence of ν on Δ now. The χ^2 statistic doubles if we take Δ to be 0.327 or 0.663. At these values we find the best fits ($\nu = 0.2540 \pm 0.0048$ ($n_{\min} = 100$)) and ($\nu = 0.3021 \pm 0.0044$ ($n_{\min} = 200$)). Comparing the values of ν from these fits to our best estimate, we find that

$$\nu_s = 0.288 \pm 0.003 \pm 0.034. \tag{4.25}$$

Nine Dimensions. If we repeat the analysis here (similar to that in eight dimensions), then we find a best estimate at $\Delta = 0.627$ where $\nu = 0.2975 \pm 0.0025$ ($\chi_4^2 = 3.3$, $< 10\%$ and $n_{\min} = 100$). The value of the χ^2 statistic doubles at $\Delta = 0.687$ ($\nu = 0.3028 \pm 0.0023$ if $n_{\min} = 100$) and $\Delta = 0.566$ ($\nu = 0.2922 \pm 0.0018$ if $n_{\min} = 50$). Comparing these results we find that

$$\nu_s = 0.2975 \pm 0.0025 \pm 0.0053. \tag{4.26}$$

4.3.3. Discussion. It is generally believed that the mean span is a poorly behaved variable, and that the calculation of ν from it is not reliable (in support of this belief, observe that (4.25) and (4.26) are far from the 'known' value $1/4$). As our best estimates we take results from the analysis of the mean square radius of gyration. The results for the exponent ν are consistently near the Flory values for ν, where $\nu = 5/2(d+2)$ (Isaacson and Lubensky 1980, see also Bovier *et al* 1984). Of particular interest are the results in three dimensions. The results from the mean span barely exclude 0.5, and the results from the mean square radius of gyration include the expected value well within error bars. The estimates of ν are consistent with 0.5. We therefore conclude that we see no evidence that equation (1.1) breaks down in this calculation.

In eight and nine dimensions we expect that $\nu = 0.25$. Unfortunately, our data suffer from strong corrections to scaling in these cases. (In general, the corrections got worse with increasing dimension.) We extracted the best estimates for ν only with great difficulty, and we urge the reader to reanalyse the data in tables 3 and 4. In eight dimensions the estimate from the mean square radius of gyration data excludes the expected value, but, due to a large systematic error, the estimate from the mean span is consistent with the expected value. In nine dimensions the situation is reversed. The estimate from the mean span data is inconsistent with the expected value, but then we note that the mean square radius of gyration gives an estimate which is close to $1/4$.

The value of Δ has been esimated using series analysis in two dimensions by Ishinabe (1989) ($\Delta = 0.635 \pm 0.030$). As a consistency check, we performed a regression for the mean square radius of gyration data assuming this value of Δ in (4.14). We find that $\nu = 0.6404 \pm 0.0058$, $A = 0.1248 \pm 0.0090$ and $b = 0.47 \pm 0.33$, while $\chi_6^2 = 4.8$ ($< 50\%$). (Error bars are 95% confidence intervals.) These are consistent with the exact enumeration results (of Ishinabe), which are $\nu = 0.644 \pm 0.004$ and $A = 0.1156$. Observe that fixing Δ at 0.635 *does not* change the estimate of ν by more than one standard deviation, and is therefore statistically indistinguishable from (4.17). If we consider the mean span instead, then a linear regression is good with $\chi_6^2 = 9.7$ ($< 90\%$). We find $\nu = 0.6357 \pm 0.0044$, while $A = 1.01 \pm 2.67$ and $b = -1.05 \pm 0.13$. This value of ν is not consistent with (4.22), but the sensitivity of ν on Δ is consistent with the idea that the mean span is a poorly behaved variable. (In these regressions we discarded the point at $n = 600$, which we believe is an outlier.)

In three and more dimensions we can use the values of Δ from Adler *et al* (1988). In three dimensions a regression with $\Delta = 1.3$ is fine ($\chi_7^2 = 11.7$, $< 95\%$). We find $\nu = 0.4921 \pm 0.0015$, $A = 0.189 \pm 0.004$ and $b = 0.49 \pm 0.16$. This value of ν is in the confidence interval of (4.18), and probably suffers from a sizable systematic error (since Δ is not at an optimal value in the regression). An attempt to fit the mean span data with $\Delta = 1.3$ is not good. In four dimensions $\Delta = 0.8$ (Adler *et al* 1988). An attempt to fit the mean square radius of gyration data to this Δ is not good. The mean span is better; as reported, the optimal value of Δ is 0.78, where $\nu = 0.4303 \pm 0.0024$, $A = 0.998 \pm 0.009$ and $b = -2.6 \pm 0.7$. Here $\chi_7^2 = 8.7$, $< 75\%$.

In summary, in two and three dimensions, we find little evidence for preferring one value of Δ over any other. In four dimensions, the data from the span does seem to suggest $\Delta = 0.78 \pm 0.10$, but the data from the radius of gyration does not really support any particular value. We leave the readers to draw their own conclusions; our own opinions are summarized at the beginning of this section (below (4.14)).

4.4. The mean longest path and mean branch size

In section 2.3 we considered the mean longest path in a tree, $\langle p_n \rangle$, and the mean branch size, $\langle b_n \rangle$, of trees. We argued that $\rho = \epsilon$, where $\langle p_n \rangle \sim n^\rho$, and $\langle b_n \rangle \sim n^\epsilon$. In this section we consider the estimation of ρ and ϵ from our data. A mean field calculation gives $\epsilon = 0.5$ (appendix A). The data are listed in tables 5 and 6. We proceed as in section 4.3. In each case we attempt two fits to the data; a comparison gives us a systematic error. If one of the fits is bad, then we consider the variation in the exponent with Δ to estimate a systematic error. We plot the mean longest path and the mean branch size of the trees in figures 8 and 9. We cannot visually detect

Table 5. The mean longest path of trees. Error bars are standard deviations.

n	p_n (2D)	p_n (3D)	p_n (4D)	p_n (8D)	p_n (9D)
25	15.792 (33)	14.435 (25)	13.959 (23)	13.453 (21)	13.354 (21)
50	26.560 (70)	22.866 (41)	21.697 (37)	20.463 (32)	20.376 (33)
100	44.21 (14)	36.364 (68)	33.442 (55)	30.601 (48)	30.354 (48)
200	74.05 (26)	57.06 (12)	51.300 (87)	45.051 (70)	44.758 (72)
300	122.15 (50)	74.57 (16)	65.62 (11)	56.564 (90)	55.869 (87)
400	98.92 (28)	90.01 (20)	78.12 (14)	65.96 (11)	65.37 (11)
600	166.15 (73)	117.36 (25)	100.02 (17)	82.44 (13)	81.06 (13)
800	202.83 (98)	141.48 (30)	119.74 (20)	96.40 (15)	94.58 (15)
1200	274.2 (15)	185.26 (42)	153.74 (26)	119.59 (19)	117.26 (19)
1600	343.6 (21)	222.90 (53)	182.01 (29)	139.07 (23)	136.27 (22)

Table 6. The mean branch size of trees. Error bars are standard deviations.

n	b_n (2D)	b_n (3D)	b_n (4D)	b_n (8D)	b_n (9D)
25	4.172 (11)	3.7966 (89)	3.6668 (80)	3.5476 (75)	3.5203 (74)
50	6.878 (20)	5.919 (14)	5.588 (12)	5.256 (11)	5.202 (11)
100	11.300 (40)	9.264 (22)	8.515 (17)	7.719 (14)	7.643 (14)
200	18.810 (70)	14.472 (34)	12.917 (25)	11.271 (18)	11.136 (17)
300	25.12 (11)	18.893 (48)	16.509 (29)	13.998 (21)	13.839 (15)
400	31.03 (14)	22.751 (57)	19.561 (34)	16.303 (22)	16.132 (22)
600	42.14 (20)	29.654 (69)	25.016 (40)	20.283 (25)	19.953 (25)
800	51.40 (27)	35.668 (82)	29.862 (48)	23.597 (28)	23.166 (28)
1200	69.34 (40)	46.56 (12)	38.318 (60)	29.191 (32)	28.672 (30)
1600	86.87 (54)	56.07 (15)	45.464 (68)	34.012 (34)	33.311 (33)

any curve in any of these plots, so corrections to scaling are less 'visible' here than in figures 6 and 7 where we considered the mean square radius of gyration and the mean span. We fit these data to either a two-parameter assumption

$$\langle p_n \rangle = D n^{\rho} \tag{4.27}$$

or a three-parameter form with a power law correction

$$\langle p_n \rangle = n^{\rho}(D + cn^{-\Delta}) \tag{4.28}$$

similar to the assumptions for the mean span and mean square radius of gyration. We assume similar functions for the mean branch size and the exponent ϵ.

Two dimensions. A two-parameter fit to the mean longest path data is good with $n_{\text{min}} = 100$ ($\chi^2_6 = 10.9$, $< 95\%$). The exponent is $\rho = 0.7357 \pm 0.0034$. If we attempt a three-parameter fit then we observe that the χ^2 statistic gets smaller as Δ gets smaller; it is also not very sensitive to changes in Δ. We therefore assume that $\Delta = 0.5$. The best fit is with $n_{\text{min}} = 50$ ($\chi^2_4 = 11.4$, $< 95\%$). We find $\rho = 0.738 \pm 0.011$. A comparison gives our best estimate

$$\rho = 0.737 \pm 0.011 \pm 0.002. \tag{4.29}$$

A two-parameter fit to the mean branch size data is good with $n_{\text{min}} = 100$ ($\chi^2_6 = 10.5$, $< 90\%$). The exponent is $\epsilon = 0.7328 \pm 0.0054$. A three-parameter fit behaves as for the mean longest path. The best Δ is small, but we observe that the χ^2 statistic does not change much with Δ. If we fix $\Delta = 0.5$, then a fit with $n_{\min} = 25$ is very good ($\chi_6^2 = 9.0$, $< 90\%$). The exponent is $\epsilon = 0.7388 \pm 0.0074$. If we compare these results, then we find our best estimate

$$\epsilon = 0.7358 \pm 0.0074 \pm 0.0030. \tag{4.30}$$

Three dimensions. The mean longest path data is well described by a two-parameter fit if $n_{\min} = 100$ ($\chi_6^2 = 3.3$, $< 25\%$). We find $\rho = 0.6545 \pm 0.0017$. A three-parameter fit works well with $\Delta = 0.78$ ($n_{\min} = 25$, and $\chi_6^2 = 8.6$, $< 90\%$). We find $\rho = 0.6534 \pm 0.0026$. If we compare these results, then

$$\rho = 0.6540 \pm 0.0026 \pm 0.0006. \tag{4.31}$$

The mean branch size fits well to the two-parameter assumption with $n_{\min} = 25$ ($\chi_8^2 = 12.0$, $< 90\%$). The value of the exponent is $\epsilon = 0.6480 \pm 0.0012$. The three-parameter fit is best if $\Delta = 0.65$, but the χ^2 statistic does not vary much with Δ ($\chi_6^2 = 1.92$, $< 10\%$). We find $\epsilon = 0.6530 \pm 0.0034$. If we compare these results then

$$\epsilon = 0.6505 \pm 0.0034 \pm 0.0025. \tag{4.32}$$

Four dimensions. A two-parameter fit to the mean longest path data with $n_{\min} = 100$ has $\chi_6^2 = 15.6$, $< 99\%$. We find $\rho = 0.6117 \pm 0.0014$. A three parameter fit with $\Delta = 1.11$ has $\chi_6^2 = 15.0$ ($< 99\%$) with $n_{\min} = 25$. The exponent is $\rho = 0.6095 \pm 0.0016$. If we compare these results, then

$$\rho = 0.6106 \pm 0.0016 \pm 0.0012. \tag{4.33}$$

A two-parameter fit to the mean branch size data has $\chi_8^2 = 18.3$ ($< 99\%$). The result is $\epsilon = 0.60503 \pm 0.00092$. A three-parameter analysis indicates that the exponent is insensitive to the value of Δ. In fact, changing Δ from 0.2 to 1.0 does not change the exponent outside its 95% confidence interval of the estimate at $\Delta = 0.5$. At $\Delta = 0.5$ we find $\epsilon = 0.6062 \pm 0.0060$. A comparison gives

$$\epsilon = 0.6056 \pm 0.0060 \pm 0.0006. \tag{4.34}$$

Eight dimensions. In eight dimensions one expects a logarithmic correction to scaling, but our numerical procedures cannot detect such behaviour. In fact, a power law assumption (such as equation (4.28)) is better in these cases. For the mean longest path, a two-parameter fit is very poor. A three-parameter fit is good if $\Delta = 0.46$ ($\chi_6^2 = 9.0$, $< 90\%$). We find $\rho = 0.5166 \pm 0.0030$. To estimate a systematic error, we consider choices of Δ which double the χ^2 statistic. If we compare the best value of ρ to regressions done at $\Delta = 0.18$ ($\rho = 0.464 \pm 0.018$ ($n_{\min} = 50$)) and 0.78 ($\rho = 0.522 \pm 0.008$ ($n_{\min} = 200$)), then we find that our best value has a large systematic error:

$$\rho = 0.517 \pm 0.003 \pm 0.053. \tag{4.35}$$

A two-parameter fit to the mean branch size data with $n_{\min} = 200$ is good ($\chi^2_5 = 9.5$, $< 95\%$). This gives $\epsilon = 0.5306 \pm 0.0014$. A three-parameter fit with $n_{\min} = 25$ has $\chi^2_6 = 3.9$, $< 50\%$, if $\Delta = 0.45$. This gives $\epsilon = 0.5161 \pm 0.0028$. If we compare these results, then we find

$$\epsilon = 0.5234 \pm 0.0028 \pm 0.0073. \tag{4.36}$$

Nine dimensions. In nine dimensions we expect mean field results for our exponents. For the mean longest path a two parameter fit is not too good, but with $n_{\min} = 300$ we find $\chi^2_4 = 6.4$, $< 90\%$, which is acceptable. This gives $\rho = 0.5326 \pm 0.0026$. A three-parameter fit works well, $\chi^2_6 = 2.6$, $< 25\%$ and $\Delta = 0.52$. This gives $\rho = 0.5121 \pm 0.0028$. If we compare these results, then we find our best estimate

$$\rho = 0.522 \pm 0.003 \pm 0.011. \tag{4.37}$$

The mean branch size data fits well to a two-parameter assumption if we take $n_{\min} = 300$ ($\chi^2_4 = 3.6$, $< 50\%$). This gives $\epsilon = 0.5244 \pm 0.0016$. A three-parameter fit is good if we take $\Delta = 0.31$ and $n_{\min} = 25$, then $\chi^2_6 = 4.3$, $< 50\%$. This gives $\epsilon = 0.4968 \pm 0.0040$. A comparison of these results gives our best estimate

$$\epsilon = 0.511 \pm 0.004 \pm 0.014. \tag{4.38}$$

4.4.1. Discussion. The mean field calculation in subsection 2.3.4 predicted that $\rho = \epsilon$. This seems to be the case in each dimension that we considered in this calculation, where ρ and ϵ agree consistently within error bars. In high dimensions ρ and ϵ also turn out to be close to $1/2$, which we expected to be the case from the calculation in appendix A.

4.5. The degrees of vertices

We argued in section 2.3 that the number of vertices of degree i, $\langle t_i \rangle$, is expected to rise linearly with n. In view of assumption (2.20), we can now assume that

$$\langle t_i \rangle / n = \zeta_i + \gamma n^{-1}. \tag{4.39}$$

We calculate the ζ_i for $i = 1, 2$ and 3 by taking a numerical limit as we have done in section 4.2. Also, let

$$t_{\geqslant 4}(\omega) = \sum_{i=4}^{2d} t_i(\omega) \tag{4.40}$$

be the number of vertices with degree greater than 3 in a tree ω. In the same way, we shall calculate $\zeta_{\geqslant 4}$, which we define to be the limit of $\langle t_{\geqslant 4} \rangle / n$.

Table 7. The fraction of vertices of degree i. Error bars are standard deviations.

d	ζ_1	ζ_2	ζ_3	$\zeta_{\geqslant 4}$
2	0.2637 (2)	0.4978 (2)	0.2134 (1)	0.02513 (5)
3	0.3096 (2)	0.4341 (2)	0.2075 (1)	0.04874 (6)
4	0.3284 (1)	0.4110 (1)	0.2017 (1)	0.05902 (4)
8	0.3522 (1)	0.3840 (1)	0.1917 (1)	0.07211 (3)
9	0.3545 (1)	0.3815 (1)	0.1906 (1)	0.07327 (3)

We list our results in table 7. The data suggest that the ζ_i converge as the dimension increases. It would be interesting to determine if there is any theoretical basis for this behaviour. The error bars in table 7 are standard deviations.

5. Conclusions

The introduction of large, non-local, elementary transitions in a Monte Carlo simulation of lattice (bond) trees proved very successful. We illustrated that the performance of the algorithm is enhanced significantly by comparison to an algorithm with only small elementary transitions (algorithm A). In addition, the performance of algorithm B improves with increasing dimension, a theoretical limit on the growth of the exponential autocorrelation time, measured in CPU seconds, can be estimated from equations (4.7) and (A.3). We find

$$\tau_e(B) \sim n \quad \text{CPU seconds.} \tag{5.1}$$

Below the critical dimension of eight, we cannot expect such good behaviour: in fact, we find that (in units of CPU seconds) the integrated autocorrelation times for algorithm B grow roughly like $n^{2.2}$ in two dimensions and like $n^{1.8}$ in three dimensions; the exponent decreases (to 1, we conjecture) as the dimension increases. For comparison, in the case of the pivot algorithm applied to self-avoiding walks, the integrated autocorrelation times are believed to be proportional to n in *every* dimension (Madras and Sokal 1988). The pivot algorithm for polygons in three dimensions fared slightly worse: the integrated autocorrelation times were found to grow roughly as $n^{1.1}$ (Janse van Rensburg *et al* 1990). However, our algorithm B was found to be much better than algorithm A, whose integrated autocorrelation times were found to grow roughly like n^3 in both two and three dimensions. This may improve in higher dimensions, but it can never be better than n^2, by the following argument. We can think of algorithm A as 'picking' leaves from the tree and appending them elsewhere on the tree. Consider the correlation between two edges on the longest path of the tree. To destroy this (conformational) correlation, algorithm A must remove edges from an endpoint of the longest path until it has deleted at least one of these edges. Let the set of edges to be removed be coloured red, and let the rest of the tree be coloured blue. If an elementary transition removes a red edge and appends it to the blue side, then this edge becomes blue, and vice versa. The number of red edges is $O(n)$ (see the arguments in section 2.3). Since the algorithm moves edges between the red and blue sets, and within the red and blue sets, we can think of the number of red edges performing a random walk on positive integers. Therefore, the number of transitions required to remove all the red edges will be $O(n^2)$. Since each transition takes $O(1)$ CPU seconds, we find

$$\tau_e(A) \sim n^2 \quad \text{CPU seconds.} \tag{5.2}$$

This heuristic argument supports our belief that algorithm B is numerically superior to algorithm A.

The grand canonical Monte Carlo algorithms for trees are necessary to estimate the exponent θ (equation (1.2)), which we cannot extract from our data. For walks, the introduction of pivots into a grand canonical algorithm, such as the BFACF algorithm (Berg and Foerster 1981) improved the performance of the Monte Carlo simulation (see for example Caracciolo *et al* 1989). The way is now open for a similar development in the simulation of lattice trees; a hybrid algorithm consisting of the algorithm of Glaus (1985) supplemented with large non-local moves from algorithm B should perform much better than previous algorithms.

Table 8. Best estimates for the exponents ν and ρ. The error bars are calculated by adding the 95% confidence interval to a systematic error.

Dimensions	ν	ρ
2	0.6371 ± 0.0119	0.736 ± 0.013
3	0.4960 ± 0.0052	0.6523 ± 0.0059
4	0.4200 ± 0.0162	0.6081 ± 0.0066

Table 8 contains our best estimates of exponents in two, three, and four dimensions. For ν, we take the estimates obtained from the mean square radius of gyration data as our best estimates, and for ρ we take the average of the estimates from the mean longest path data and the mean branch size data (assuming of course that $\rho = \epsilon$). The error bars given are taken to be the maximum error in each of the two estimates. To give a single error bar we took the sum of the systematic and the statistical errors. Note that the exponent ρ is a new, intrinsic exponent for trees, measuring the mean path length between vertices in the tree. The fact that $\rho < 1$ is interesting; it implies that in the scaling limit, the distance between two nearest vertices of degree other than two will go to zero. A tree will be a highly branched object, with an arbitrary number of branching points in every open subset of the tree.

Table 9. The estimates for ν from the literature. (a = Series expansion, b = Monte Carlo calculation, c = Exact enumeration of animals, d = Renormalisation group, e = Flory exponents, f = Dimensional reduction, g = Exact enumeration, h = Scanning Method). The error bars are those given by the authors.

Reference	2 dimensions	3 dimensions	4 dimensions
Kurze and Fisher (1979)^a	—	0.5	0.425
Redner (1979)^b	0.57 ± 0.06	0.45 ± 0.06	—
de Alcantara Bonfim *et al* (1980)^c	—	0.55 ± 0.05	0.450 ± 0.035
Family (1980)^d	0.637	—	—
Isaacson and Lubensky (1980)^e	0.625	0.5	0.417
Gould and Holl (1981)^b	—	0.53 ± 0.02	—
Parisi and Sourlas (1981)^f	—	0.5	—
Parisi and Sourlas (1981)^d	—	—	0.42
Seitz and Klein (1981)^b	0.615	0.46	—
Gaunt *et al* (1982)^g	—	0.55 ± 0.05	0.45 ± 0.05
Derrida and de Seze (1982)^d	0.6408 ± 0.0003	—	—
Dhar (1983)^f	—	—	0.417
Bovier *et al* (1984)^b	0.6402 ± 0.0084	—	—
Margolina *et al* (1984)^g	0.640 ± 0.004	—	—
Privman (1984)^g	0.6394 ± 0.0067	—	—
Alexandrowicz (1985)^b	0.64 ± 0.03	0.50 ± 0.03	0.42 ± 0.03
Caracciolo and Glaus (1985)^b	0.635 ± 0.015	—	—
Glaus (1985)^b	—	0.495 ± 0.013	—
Duarte (1986)^b	0.650 ± 0.015	—	—
Meirovitch (1987)^h	0.640 ± 0.004	—	—
Adler *et al* (1988)^a	—	0.500 ± 0.010	0.425 ± 0.015
Ishinabe (1989)^c	0.644 ± 0.004	—	—
This paper (1991)^b	0.637 ± 0.012	0.4960 ± 0.0052	0.420 ± 0.017

Finally, we provide a detailed comparison with results of previous estimates of the exponent ν in table 9. In two dimensions the best estimates are from renormalization group calculations: Derrida and de Seze (1982) found $\nu = 0.6408 \pm 0.0003$ while Kertész (1986) estimated ν to be 0.6406 ± 0.0002. From a numerical point of view we expect exact enumeration to provide the best estimates for ν (in analogy with the situation for walks). This is indeed the case, as we note in table 9. An exact enumeration by Margolina *et al* (1984) estimates $\nu = 0.640 \pm 0.004$ and a study by Ishinabe (1989) finds $\nu = 0.644 \pm 0.004$, while Meirovitch (1987) uses the scanning method (which can be considered a hybrid of exact enumeration and the Monte Carlo method of Rosenbluth and Rosenbluth (1968) (for walks) generalized to trees) to find that $\nu = 0.640 \pm 0.004$. Of the Monte Carlo simulations for trees the best estimate was produced by a simulation done by Bovier *et al* (1984) who found $\nu = 0.6402 \pm 0.0084$. The other Monte Carlo studies performed in the last ten years all have results of comparable accuracy, these are the simulations by Caracciolo and Glaus (1985) ($\nu = 0.635 \pm 0.015$), Duarte (1986) ($\nu = 0.650 \pm 0.015$) and the results in this paper ($\nu = 0.637 \pm 0.012$). (As we noted in section 2.2, the algorithm used by Duarte fails to satisfy detailed balance; however, his result includes the accepted value. It is difficult to guess what systematic error would be present in his data.) The total computing time involved in estimating ν using algorithm B was about 11 hours of CPU time on a DEC5000 workstation. If we compare this fact to the results of Bovier *et al* (1984) and Caracciolo and Glaus (1985) we note a big improvement in performance, even if we take into account the slower computers used in those studies. (Bovier *et al* (1984) performed a run of 180 hours on a CDC-174/720 while Caracciolo and Glaus (1985) performed a run of 380 hours on a VAX 11/780). In defense of the grand canonical algorithms we should note that they can, in addition to ν, also estimate the growth constant and the specific heat exponent (θ) *in the same run.*

Studies of trees in three dimensions are not as common as in two dimensions. The best estimate is given by the Monte Carlo simulation in this paper ($\nu = 0.4960 \pm 0.0052$), while a heroic effort by Glaus (1985) estimates that $\nu = 0.495 \pm 0.013$ (using a grand canonical algorithm). In four dimensions a similar situation is found. The best result is given by the simulation in this paper ($\nu = 0.420 \pm 0.017$), the only other Monte Carlo result being that of Alexandrowicz (1985) who found that $\nu = 0.42 \pm 0.03$. We are not aware of any exact enumeration results in three and four dimensions. These Monte Carlo results are close to the 'exact' value $5/12$ of ν, which resulted from Dhar's identification of directed animals (in d dimensions) and a lattice gas with extended hard cores in $(d-1)$ dimensions. Numerically, it is possible that even better results in three and four dimensions could be found by an exact enumeration study; however, with improving technology, the best numerical results will inevitably come from Monte Carlo studies since the effort in exact enumeration grows exponentially with increasing n.

Acknowledgments

The authors acknowledge numerous discussions with S G Whittington. N Madras was partially supported by a University Research Fellowship and an operating grant from NSERC of Canada.

Appendix A. Mean field theory for ϵ

Let t_n be the number of trees (unrooted) with n vertices. Then it is believed that

$t_n \sim n^{-\theta}\lambda^n$, with θ given in equation (1.2). Pick an edge in a tree. This roots the tree at this edge. If we delete the edge, then we find two subtrees, one with (say) k vertices and the other with $n-k$ vertices, *rooted* at the vertices incident on the deleted edge. The mean number of vertices in the smaller subtree (assume that $k < n-k$ without loss of generality) is then given by

$$\langle b_n \rangle = \frac{1}{t_n}\sum_{k=1}^{n/2} k^2 t_k (n-k) t_{n-k}. \tag{A.1}$$

We can now evaluate this expression to find

$$\langle b_n \rangle \sim n^{3-\theta}. \tag{A.2}$$

The mean field value of θ is $5/2$ (Bovier *et al* 1984), so we have

$$\epsilon = 1/2 \tag{A.3}$$

in the mean field approximation.

References

Adler J, Meir Y, Harris A B, Aharony A and Duarte J A M S 1988 *Phys. Rev.* D **38** 4941
Alexandrowicz Z 1985 *Phys. Rev. Lett.* **54** 1420
Berg B and Foerster D 1981 *Phys. Lett.* **106B** 323
Bovier A, Fröhlich J and Glaus U 1984 *Critical Phenomena, Random Systems, Gauge Theories (Les Houghes, Session XLIII)* ed K Osterwalder and R Stora (Amsterdam: Elsevier)
Caracciolo S and Glaus U 1985 *J. Stat. Phys.* **41** 95
Caracciolo S, Pelissetto A and Sokal A D 1989 *Nucl. Phys.* B *(Proc Suppl)* **9** 525
Carmesin I and Kremer K 1988 *Macromolecules* **21** 2819
Dhar D 1983 *Phys. Rev. Lett.* **51** 853
de Alcantara Bonfim O F, Kirkham J E, McKane A J 1980 *J. Phys. A: Math. Gen.* **13** L247
Derrida B and de Seze L 1982 *J. Physique* **43** 475
Dewdney A K 1985a *Sci. Am.* (6) 18
—— 1985b *Sci. Am.* (9) 24
Duarte J A M S 1986 *J. Phys. A: Math. Gen.* **19** 1979
Duarte J A M S and Cadilhe A M R 1989 *J. Stat. Phys.* **56** 951
Duarte J A M S and Ruskin H J 1981 *J. Physique* **42** 1585
Family F 1980 *J. Phys. A: Math. Gen.* **13** L325
Fisher D S, Fröhlich J and Spencer T 1984 *J. Stat. Phys.* **34** 863
Fisher M E 1978 *Phys. Rev. Lett.* **40** 1610
Gaunt D S, Sykes M F, Torrie G M and Whittington S G 1982 *J. Phys. A: Math. Gen.* **15** 3209
Glaus U 1985 *J. Phys. A: Math. Gen.* **18** L609
Gould H and Holl K 1981 *J. Phys. A: Math. Gen.* **14** L443
Imbrie J Z 1984 *Phys. Rev. Lett.* **53** 1747
Isaacson J and Lubensky T C 1980 *J. Physique Lett.* **41** L469
Ishinabe T 1989 *J. Phys. A: Math. Gen.* **22** 4419
Janse van Rensburg E J, Whittington S G and Madras N 1990 *J. Phys. A: Math. Gen.* **23** 1589
Kemeny J G and Snell J L 1976 *Finite Markov Chains* (Berlin: Springer)
Kertész J 1986 *J. Phys. A: Math. Gen.* **19** 599
Knuth D E 1973 *The Art of Computer Programming* vol 3 (Reading, MA: Addison-Wesley)
Kremer K and Binder K 1988 *Comput. Phys. Rep.* **7** 259
Kurze D A and Fisher M E 1979 *Phys. Rev.* B **20** 2785
Lipson J E G, Gaunt D S, Wilkinson M K and Whittington S G 1987 *Macromolecules* **20** 186

Lubensky T and Isaacson J 1979 *Phys. Rev.* A **20** 2130
Madras N 1989 Unpublished
Madras N and Sokal A D 1988 *J. Stat. Phys.* **50** 109
Margolina A, Family F and Privman V 1984 *Z. Phys.* B **54** 321
Meirovitch H 1987 *J. Phys. A: Math. Gen.* **20** 6059
Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H and Teller E 1953 *J. Chem. Phys.* **21** 1087
Parisi G and Sourlas N 1981 *Phys. Rev. Lett.* **46** 871
Peters H P, Stauffer D, Hölters H P and Loewenich K 1979 *Z. Phys.* B **34** 399
Privman V (1984) *Physica* **123A** 428
Rosenbluth M N and Rosenbluth A W 1968 *J. Chem. Phys.* **49** 648
Redner S 1979 *J. Phys. A: Math. Gen.* **12** L234
Seitz W A and Klein D J 1981 *J. Chem. Phys.* **75** 5190
Silvey S D 1975 *Statistical Inference* (London: Chapman and Hall) ch 3
Whittington S G, Lipson J E G, Wilkinson M K and Gaunt D S 1986 *Macromolecules* **19** 1241
Wilson R J and Watkins J J 1990 *Graphs: An Introductory Approach* (New York: Wiley)