Exact and efficient discrete random walk method for time-dependent environments

J. Asikainen[1], J. Heinonen[1] and T. Ala-Nissila[1,2]
[1] *Helsinki Institute of Physics and Laboratory of Physics, Helsinki University of Technology, P.O. Box 1100, FIN–02015 HUT, Espoo, Finland*
[3] *Department of Physics, Brown University, Providence, RI 02912–1843*
(August 2, 2002)

We present an exact method for speeding up random walk in complicated lattice environments. To this end, we derive the discrete two-dimensional probability distribution function for a diffusing particle starting at the center of a square of linear size s. This is used to propagate random walkers from the center of the square to sites which are nearest neighbors to its perimeter sites, thus saving $\mathcal{O}(s^2)$ steps in numerical simulations. We discuss in detail how this method can be implemented efficiently. We examine its performance in the diffusion limited aggregation model which produces fractal structures, and in a one-sided step-growth model producing compact, finger-like structures. We show that in both cases, the square propagator method reduces the computational effort by a factor proportional to the linear system size as compared to standard random walk.

PACS number(s):

I. INTRODUCTION

Random walk (RW) based methods constitute one of the most useful tools in the study of various physical systems [1]. Such methods have been employed to study random materials, including glasses, polymers, and amorphous materials, and on larger scales porous media, composites and suspensions (a thorough review of the applications of RW is given by Weiss [1]). In particular, many transport processes in disordered media can be analyzed with RW models [2,3]. Examples include problems in polymer physics [4], various diffusion processes [5,6], crystallography [7], and reaction kinetics [1]. RW techniques have also been applied to the study of biological phenomena, such as modeling of bacterial motion [8].

A particularly useful application of RW methods in continuum is based on the analogy of the diffusion equation to an electrostatic problem where one wants to know the charge density on a boundary surrounding a point charge. Namely, the Green's function of the Laplace's equation equals the probability density of diffusing random walkers at the boundary [9,10]. When the underlying geometry is simple, standard numerical or analytical methods for electromagnetic problems are applicable. However, for complicated geometries or in particular when the boundary evolves in time, using a RW algorithm may turn out to be much more efficient.

An important example of a problem with complicated, time dependent boundary conditions is Diffusion Limited Aggregation (DLA) which is a model of irreversible growth to generate fractal structures [11]. It has been used to study a great variety of processes, including dendritic growth, viscous fingering in fluids, dielectric breakdown, and electrochemical deposition [12]. The DLA is a model defined conventionally on a lattice, where fractal structures are grown by particles that diffuse and stick to an initial seed particle (or a line of seed particles). In standard simulations of DLA lattice models, one releases a particle far away from the seed, lets it walk until it sticks to the structure, and continues then with a new particle. The main technical problem in this approach is that the random walkers will occasionally escape the finite simulation box. In this case if the particle steps beyond a large, predefined distance from the seed, it is usually rejected, and a new particle is released. To avoid this problem, more sophisticated methods have been developed in the continuum limit [13]. The basic idea is that instead of the particle stepping only a short (constant) distance at a time, a random step as large as possible is taken. Also, if the particle exits a circle around the seed (in the case of circular DLA), it is projected back with the "first hit" probability distribution (see the Appendix of Ref. [13]).

Similar methods have also been used in lattice simulations. Meakin and Tolman [14] used a method, where one takes the largest empty hypersphere around the diffusing particle, and moves the particle to a randomly chosen position on its surface. Ball and Bray [10] calculated the Laplacian Green's function for a hypercube, and projected the particle to the boundary of it. However, both of these methods introduce approximations. In the first case, there is a systematic error in mapping of a hypersphere onto the lattice sites, and in the latter case error is induced by mapping continuous coordinates onto lattice sites. Such errors are largest for small steps, and may influence some properties of the growing structures in an uncontrollable way.

In this work, our aim is twofold. We will derive and present an *exact* method for speeding up random walkers in complicated lattice environments. To this end, we calculate the complete *discrete* two-dimensional probability distribution function for a diffusing particle around a square of linear size s. This is used to propagate random walkers starting from the center of the square, to

Accepted for publication in Phys. Rev. E. (Manuscript number: EV8006).

any of the $4s$ sites that are nearest neighbors (NN) to the $4s-4$ perimeter sites (see Fig. 1), thus saving $\mathcal{O}(s^2)$ steps in the simulation. In the limit of large s, our solution agrees with the known continuum case [15]. Second, we discuss in detail how this discrete square propagator method can be implemented efficiently and examine its performance in two growth models. The first one is the DLA lattice model. In spite of its apparent simplicity, an analytic solution is still unavailable, and thus numerical work provides most of the current understanding of the model. The model exhibits very slow approach to the asymptotic limit [13,16], and efficient simulation algorithms are crucial in the study of DLA. The second model is that introduced by Heinonen *et al.* [17] (HBAK) to study kinetic roughening in one-sided, anisotropic step growth. The HBAK model produces compact finger-like structures, with growth rules exactly the same as in DLA, except for a sliding-down condition at the boundary of the growing aggregate. For both models, we demonstrate that the square propagation method reduces the computational effort by a factor proportional to the linear system size L as compared to the standard RW approach. We note that the implementation and conclusions presented here can also be applied to the continuum case [15] for off-lattice problems.

II. PROPAGATORS

We consider here systems on a 2D rectangular geometry, where initially the bottom of the system at $y=0$ comprises the growth (sticking) sites, and diffusive particles are released from $y=\infty$. The system size in the lateral x direction is L, with periodic boundary conditions. Such systems were also considered by Heinonen *et al.* [17] who devised an exact half-plane propagation (HPP) method to speed up diffusion limited growth. It is based on releasing each walker at height $y_{\rm HPP}$ which is defined to be one lattice site higher than the highest point of the growing substrate. If the particle crosses the line again in the $+y$ direction while performing RW, it is immediately returned to the line $y=y_{\rm HPP}$ with a new x coordinate chosen from the appropriate spatial distribution. The Fourier transform of the corresponding spatial probability distribution function $P_{\rm HPP}(x)$ is given exactly by [17]

$$\hat{P}_{\rm HPP}(q) = 2-\cos q - \sqrt{(2-\cos q)^2-1}, \qquad (1)$$

where $q=2\pi n/L$, with $n=0,1,\ldots,L-1$. The general formula for non-zero drift can be found in Ref. [17].

Although the HPP method makes it possible to simulate systems of infinite size in the y direction exactly, with growing spatial structures the random walkers spend more and more time in trying to find available growth sites. Indeed, we expect that the time grows typically as L^2 in self-similar systems of size L. The additional idea here is to speed up the RW process at every possible step. This can be achieved with the so-called square propagator (SQP) which we here define to be the probability distribution for a particle released at the center of a square of linear size s, to appear at any site which is a NN to the perimeter sites of the square. By choosing the square to be as large as possible at every diffusion step, the particles can be very efficiently propagated to available growth (absorbing) sites, as shown schematically in Fig. 1.

To define the SQP, we consider a square of size $s=2\ell+1$, where the *jump index* $\ell=0,1,2,\ldots$, and where the particle initially sits at the center row $\ell+1$. Let T_n be the transfer matrix that gives the probability to move from the $n^{\rm th}$ row to the $(n+1)^{\rm th}$ row (see Fig. 1). Thus, the element $T_n(i,j)$ of this matrix is the probability to jump from the site i on row n, to site j on row $n+1$. In Appendix A we derive a recursion relation for the matrix T_n and solve it using the discrete z transform. What follows is the probability distribution $P_{\rm SQP}$ for the walker to enter the $k^{\rm th}$ site at, for example, the bottom row just outside to the bottom perimeter of the square (see Fig. 1), and it is given by

$$P_{\rm SQP}(k) = \sum_{n=1}^{s} x_n^k f(\lambda_n) x_n^{\ell+1}, \qquad (2)$$

where $f(\lambda)=(\lambda_+^{\ell+1}+\lambda_-^{\ell+1})^{-1}$, $\lambda_\pm=\frac{\lambda}{2}\pm\sqrt{(\frac{\lambda}{2})^2-1}$, $x_n^k=\sqrt{\frac{2}{s+1}}\sin(\frac{\pi nk}{s+1})$, and $\lambda_k=4-2\cos\frac{\pi k}{s+1}$, with $k=1,2,...,s$.

In analogy with the HPP distribution, it is fast to calculate and tabulate the required SQP at the beginning of the simulation [17].

The displacement k can be found efficiently from the probability distribution $P(k)$ using two arrays with indices $n=0,1,\ldots,N-1$, where the length $N=1/\Delta_\xi$ is chosen such that $\Delta_\xi<\min_k[P(k)]$. With such a small Δ_ξ, each interval of the random number $n\Delta_\xi\le\xi<(n+1)\Delta_\xi$ belongs to the range of one or two displacements at most. To distinguish between the two displacements in the latter case, we find the smallest displacement k in the cumulative distribution which is still larger than the lower boundary $n\Delta_\xi$. For this purpose, we define two arrays as follows

$$\begin{aligned} A(n) &= k_n; \\ B(n) &= \sum_{z=0}^{k_n} P(z), \end{aligned} \qquad (3)$$

where $k_n=0,1,\ldots,k_{\rm max}$ is the smallest k for which $n\Delta_\xi<\sum_{z=0}^{k}P(z)$. For each random number $0\le\xi<1$, the displacement k is found as

$$k=\begin{cases} A(\mathrm{int}[\xi N]), & \xi<B(\mathrm{int}[\xi N]); \\ A(\mathrm{int}[\xi N])+1, & \xi\ge B(\mathrm{int}[\xi N]), \end{cases} \qquad (4)$$

where int[y] is the largest integer which is smaller than y. This is a fast method for any fixed probability distribution $P(k)$ such as SQP and HPP here.

When using the square propagator, we choose to tabulate only one half of the probability distribution $P_{\rm SQP}(z)$ because the size of the search arrays grows rapidly with the range of the distribution. We utilize the symmetry by dividing the square into eight segments and use two random numbers to find the final position on the perimeter of the square. The first random number is used to find the displacement $k = 1, 2, \ldots, \ell + 1$ in Eq. (2) using Eq. (4). The second random number is used to choose one of the eight segments.

To illustrate the large distances that can be covered within a single propagation step to bring the particle to the NN sites of the growing aggregate we show in Fig. 2 sample configurations of both the DLA and HBAK models in the early time regime.

III. INDEX SEARCH ALGORITHMS

While the use of the square propagator clearly gives a significant advantage in simulation efficiency, it is of no use if finding the proper square size takes too much time. Thus, the crucial aspect in the actual implementation of the SQP method is how to find the jump index ℓ corresponding to the largest possible free square around the position of the walker. To this end, we have compared two different algorithms in the DLA simulations, which we call the Array Index search, and the Multigrid Index search. For the case of the HBAK model, the search can be further simplified using a Linear Index search algorithm. In this section we discuss all these algorithms in detail, and present theoretical arguments for the performance of their implementations.

A. Array index search

The first algorithm that we implemented is based on storing the indices in a 2D array corresponding to the sites of the underlying lattice. After the walker sticks to one of the possible growth sites of the substrate, one has to update the index configuration around the sticking site.

Let us define a shell around an occupied site as the set of neighbors of the sticking site which posses the same jump index ℓ (in the absence of other occupied sites) as shown in Fig. 3. Now the local, incremental index updating is performed as follows.

First, the sticking site is set to $\ell = -2$ and its vacant NN's are set to $\ell = -1$. The corresponding negative square sizes $s = 2\ell + 1 < 0$ indicate that no jump is possible from these sites. Then, a directed walk is started on the shell with $\ell = 0$. At each site the walker steps on the index ℓ' of the site is checked against the shell number ℓ. If $\ell < \ell'$, the index is changed to ℓ. If any of the indices on shell ℓ is changed, a walk is started on the next shell $\ell + 1$. This process is continued until no sites are updated on shell ℓ, or until $\ell = \ell_{\rm max}$, the largest index value that is allowed. A further constraint for the indices to be taken into account is that direct jump to sites for which $y > y_{\rm HPP} + 1$ is not allowed to ensure that no approximations are made with the half-plane propagator. This algorithm needs about $\log_2(\ell_{\rm max})$ bits for each site but in practice, 32-bit integers are an appropriate choice.

Similar idea has been used by Meakin [18], but in an approximate way combined with off-lattice walks. We denote this the Array Index search (AI) algorithm.

In the case of the DLA clusters, the free space between the growing structures can become very large. In fact, since the empty area grows proportional to the square of the system size, one expects that the index updating with no constraints can for large enough systems become the dominant part of simulation. This could be circumvented by setting a constraint to the maximum step size. However, in this case additional time must be spent due to the larger number of diffusion steps. Since it is not obvious which way is better, we present here a detailed analysis of these two cases.

The total CPU time per particle $t_{\rm tot,AI}$ can be split into three parts: Let $t_{\rm SQP,AI}$ denote the average time per particle needed to perform the accelerated SQP walk with the AI algorithm and let $t_{\rm u,AI}$ denote the index updating time between the walkers. Then $t_{\rm tot,AI}$ can be written as

$$t_{\rm tot,AI} = n_{\rm SQP} t_{\rm SQP,AI} + t_{\rm u,AI} + n_{\rm HPP} t_{\rm HPP}, \qquad (5)$$

where $n_{\rm SQP}$ is the number of SQP leaps (We will discuss the behavior of $n_{\rm SQP}$ in Sec. IV), $t_{\rm SQP,AI}$ is the average time needed to take a single SQP leap and the number of HPP steps is expected to scale roughly as $n_{\rm HPP} = a_{\rm HPP} L$ ($a_{\rm HPP} = const.$). The time $t_{\rm SQP,AI}$ is composed of two parts: First, the index ℓ is found and then, using ℓ and the current position of the particle, the final position is calculated. The time needed for a single HPP leap $t_{\rm HPP}$ is found to be constant with L (see Sec. IV).

Next, we consider the behavior of the first two terms in the R.H.S. of Eq. (5) for two possible implementations. First, we examine the case where all the possible indices in the lattice are updated. As noted above, the updating time $t_{\rm u,AI}$ will in this case be proportional to L^2. Second, we impose a constraint on the indices so that no indices larger than a pre-defined value $\ell_{\rm max}$ are updated. This reduces the contribution of index updating time to $\mathcal{O}(L)$ (see Sec. IV). Below we show how the average SQP walk time $t_{\rm SQP,AI}$ behaves in the two cases.

Let $P(\ell, L)$ denote the probability of using the SQP corresponding to jump index ℓ in the simulation of a lattice of size L. We have calculated the index distribution numerically from our simulation, and the result is shown in Fig. 4. From the figure we see that $P(\ell) \sim \ell^{-2}$. The cutoff visible in Fig. 4 is due to the finite system size and

it scales linearly in L. Thus, $P(\ell, L)$ can be approximated as

$$P(\ell, L) = C\ell^{-2}\theta(rL - \ell), \tag{6}$$

where θ is the Heaviside step function, the constant r is close to $1/2$, and the normalization factor C is practically independent of L. Then, $t_{\mathrm{SQP,AI}}$ can be written as

$$t_{\mathrm{SQP,AI}}(L) = \sum_{\ell=1}^{\infty} P(\ell, L) t'_{\mathrm{SQP,AI}}(\ell), \tag{7}$$

where $t'_{\mathrm{SQP,AI}}(\ell)$ is the time needed to take a step of size ℓ.

Let us consider the first case where no restrictions are imposed on the indices. We denote the constant time $t'_{\mathrm{SQP,AI}}$ by $a_{\mathrm{SQP,AI}}$ which is independent of ℓ, and Eq. (7) gives simply that $t_{\mathrm{SQP,AI}} = a_{\mathrm{SQP,AI}}$ is constant in L.

In the second case let us impose a restriction to the indices in the following way: let $\ell_{\max}$ be the largest value that will be updated and used. The average number of steps n_{SQP} remains the same if we consider fictitious steps for larger indices. Now a SQP step with $\ell > \ell_{\max}$ actually consists of successive SQP steps of size $\ell \leq \ell_{\max}$. The time needed to take a step of size ℓ can thus be given as

$$t'_{\mathrm{SQP,AI}}(\ell) = \begin{cases} a_{\mathrm{SQP,AI}}, & \text{if } \ell \leq \ell_{\max}; \\ a_{\mathrm{SQP,AI}}(\ell/\ell_{\max})^2, & \text{if } \ell > \ell_{\max}. \end{cases} \tag{8}$$

Substituting $t'_{\mathrm{SQP,AI}}(\ell)$ into Eq. (7) and approximating the summation with an integral yields

$$\begin{aligned} t_{\mathrm{SQP,AI}}(L, \ell_{\max}) = a_{\mathrm{SQP,AI}} C[(1 - 1/\ell_{\max}) \\ + (rL - \ell_{\max})/\ell_{\max}^2], \end{aligned} \tag{9}$$

The idea now is to minimize the dominant scaling exponent of $t_{\mathrm{tot,AI}}$ in Eq. (5) with the system size L. As will be discussed in Sec. IV, $n_{\mathrm{SQP}} = a_n L$ ($a_n = const.$) quite accurately. Since $t_{\mathrm{u,AI}} = a_{\mathrm{u,AI}}\ell_{\max}^2$ ($a_{\mathrm{u,AI}} = const.$) one obtains the optimal dominant scaling of Eq. (5) if one chooses $\ell_{\max} \propto \sqrt{L}$, and then

$$t_{\mathrm{SQP,AI}} = \frac{3}{2} a_{\mathrm{SQP,AI}} C(1 - 1/\sqrt{L}) \tag{10}$$

becomes constant $(3/2)a_{\mathrm{SQP,AI}}$ with large L. The same scaling of $\ell_{\max}$ is obtained by minimizing the total CPU time with fixed L. In addition, a prefactor comes out from the minimization. For cases $\ell_{\max} \ll L$ we get

$$\ell_{\max} \approx \left(\frac{a_n a_{\mathrm{SQP,AI}}}{a_{\mathrm{u,AI}}} \right)^{1/4} L^{1/2}. \tag{11}$$

In terms of Eq. (21), the optimal maximum index can be given as $\ell_{max} \approx (b_3/b_4)^{1/4}\sqrt{L}$. The simulation results in Sec. IV show that $(b_3/b_4)^{1/4}$ is close to unity and the minimum CPU time can therefore be obtained with $\ell_{max} = \sqrt{L}$.

Collecting these results together, we thus find that the total simulation time behaves as

$$t_{\mathrm{tot,AI}} = \begin{cases} n_{\mathrm{SQP}} a_{\mathrm{SQP,AI}} + \frac{1}{4} a_{\mathrm{u,AI}} L^2 + a_{\mathrm{HPP}} L, & \text{if } \ell_{\max} = L/2; \\ \frac{3}{2} n_{\mathrm{SQP}} a_{\mathrm{SQP,AI}} + a_{\mathrm{u,AI}} L + a_{\mathrm{HPP}} L, & \text{if } \ell_{\max} = \sqrt{L}, \end{cases} \tag{12}$$

where $a_{\mathrm{SQP,AI}}$, $a_{\mathrm{u,AI}}$ and a_{HPP} are constants. The RW algorithm is actually identical to the AI algorithm with $\ell_{\max} = 0$ although we have implemented a separate RW algorithm for efficiency. In Sec. IV we show that n_{SQP} scales essentially linearly with L, which means that using the index cutoff, which is between the RW and the unrestricted AI algorithms, is the preferred way here.

We also note that simulation of, e.g., diffusion in static porous structure with arbitrary distribution of pore sizes [15] can be efficiently done by applying the AI method. One first calculates the index configuration and then uses the SQP for speed-up of the diffusion process (no updating of the indices needs to be done here).

B. Multigrid index search

A more sophisticated way to update the jump indices is to coarse grain the lattice in such a way that one divides the lattice into 2×2 blocks that are mapped onto one site at a higher level, continuing the mapping to as high level as needed (this hierarchical mapping algorithm was also used in Ref. [10]). If one of the four sites is occupied by the substrate, the site at the higher level becomes occupied, otherwise it becomes vacant. At each step, when one wants to know the largest empty square around a given site, one checks from the hierarchical structure level by level if the site has occupied neighbors. Updating in this way after the walker sticks is fast and simple, and the time scales as $t_{\mathrm{u,MG}} \propto \mathcal{O}(\log L)$ at most. Another advantage of this approach is that at each level one can describe the state of 32 sites using one 32-bit integer only. At the lowest level, each lattice sites requires one bit but only 1/4 bits at the next level *etc.* Therefore, each site requires about 4/3 bits in total. This bit-packing saves a lot of memory, and in the large system size limit increases efficiency due to reduction of cache mismatches. We refer this approach as the multi-grid (MG) algorithm here. One should note that also the length of the HPP search arrays in Eq. (3) is of the order of L^2. However, when the interest is in the large time properties of the aggregate as in the present study, vertical lattice size is very large as compared to the horizontal lattice size L, and thus the memory requirement of the lattice dominates.

To analyze the performance of the MG algorithm let $t_{\mathrm{SQP,MG}}$ denote the average time per particle to perform the accelerated SQP walk within MG. Similarly to Eq. (7), this can be written as

$$t_{\mathrm{SQP,MG}}(L) = \sum_{\ell=1}^{\infty} P(\ell, L) t'_{\mathrm{SQP,MG}}(\ell), \tag{13}$$

where $t'_{\mathrm{SQP,MG}}(\ell)$ now is the time needed to take a step of size ℓ. Most of the time is consumed in finding the jump index ℓ corresponding to the largest available empty square around the present position. The larger ℓ is, the higher level of the hierarchical tree must be checked. By construction, $t'_{\mathrm{SQP,MG}}(\ell)$ scales as

$$t'_{\mathrm{SQP,MG}}(\ell) \propto \log \ell + const., \tag{14}$$

where the constant term counts for the time needed to calculate the final position in the step of size ℓ. Taking this into account, the average step time can be written as

$$t_{\mathrm{SQP,MG}}(L) = \sum_{\ell=1}^{\infty} \ell^{-2} \log(\ell)\theta(rL-\ell) + const. \tag{15}$$

This sum can be approximated by an integral, whose upper limit is $t_{\mathrm{SQP,MG}}(L) < t_{\mathrm{SQP,MG}}(\infty) = a_{SQP,MG}$ is constant. Thus we get the final scaling of $t_{\mathrm{tot,MG}}$ as

$$\begin{aligned} t_{\mathrm{tot,MG}} &= n_{\mathrm{SQP}} t_{\mathrm{SQP,MG}} + t_{\mathrm{u,MG}} + n_{\mathrm{HPP}} t_{\mathrm{HPP}} \\ &= n_{\mathrm{SQP}} a_{\mathrm{SQP,MG}} + a_{\mathrm{u,MG}} \log L + a_{\mathrm{HPP}} L, \end{aligned} \tag{16}$$

where $a_{\mathrm{SQP,MG}}$, $a_{\mathrm{u,MG}}$ and a_{HPP} are constants. Comparison with Eq. (12) shows that both the optimal AI and the MG algorithms have the same asymptotic scaling for large systems, and they both reduces the CPU time by a factor proportional to the system size L as compared to the RW algorithm.

C. Linear index search

In the HBAK case the index search can be further simplified since only a one dimensional vector $h(x)$ is needed to keep track of the occupied sites at each x. When the walker is at point (x, y), the proper jump index ℓ can be found using simple linear search. One starts at the point x with initial index set to zero and checks if the corresponding square at height y fits between $h(x)$, $h(x+1)$, $h(x-1)$, and the line $y = y_{\mathrm{HPP}}$. This procedure is continued by increasing ℓ at each step by one until the square thus drawn hits (at least) one of the constraints. This will be referred to as the Linear Index (LI) algorithm. Within it, the scaling of the total time per particle behaves as

$$t_{\mathrm{tot,LI}} = n_{\mathrm{SQP}} t_{\mathrm{SQP,LI}} + n_{\mathrm{HPP}} t_{\mathrm{HPP}}, \tag{17}$$

where $n_{\mathrm{SQP,LI}}$ is the number of SQP steps per particle, $t_{\mathrm{SQP,LI}}$ is the average time needed to take a single SQP step within LI, and we expect that $t_{\mathrm{SQP,LI}}$ depends only weakly on L (see Sec. IV).

IV. RESULTS

As a first check on our algorithm, we calculated the fractal dimension of a DLA cluster in a cylindrical geometry. The initial configuration was such that the bottom line of a lattice of width L was filled by the aggregate particles, while the rest of the lattice was empty. Periodic boundary conditions were imposed in the horizontal direction.

The scaling of the mass M of the DLA structure was measured in the saturated regime, *i.e.* in the regime where the surface roughness of the growing fractal has saturated in time. This was done in order to avoid the influence of the initial line source. The surface roughness, or global width of the surface, is measured as the standard deviation of single-valued interface height $h(x,t)$ [19]:

$$w(t,L) = \langle \overline{[h(x,t) - \overline{h}(t)]^2} \rangle^{1/2}, \tag{18}$$

where the overbar denotes spatial averaging over the system of size L and angular brackets denote configuration averaging. The global width satisfies the Family-Viscek (FV) scaling ansatz [20] $w(t,L) \sim t^{\beta} f(t/L^z)$, where the scaling function $f(u \to 0) = const.$ and $f(u \to \infty) \propto u^{-\beta}$. Here, β defines the (global) growth exponent. The dynamic exponent z describes the scaling of saturation time t_s with system size, $t_s \sim L^z$, and β and z are connected through the global roughness exponent χ as $\beta = \chi/z$ [21]. The regime of our measurements corresponds to surface roughness $w(t,L) \propto L^{\chi}$ independent of time.

The lateral system sizes used in the calculation ranged from $L = 16$ to 512. A fit to the data on a logarithmic scale yielded the estimate $D = 1.66 \pm 0.01$ for the fractal dimension, which is in excellent agreement with previous studies [22–25]. We have also measured other quantities, and the results can be found in Ref. [19].

A. Application to DLA

To quantitatively compare the performance of the different algorithms in the DLA case, we measured their performance in the saturated regime. This is the worst case scenario in the sense that the fractal structure is sparse, and the empty area where diffusion occurs is large. In particular, we concentrate on the scaling of the relevant parts of the different algorithms with the system size. We note that within the SQP method, the scaling of $n_{\mathrm{SQP}}(L)$ does not depend on the algorithm used (AI, MG, LI), and that $n_{\mathrm{HPP}}(L)$ always scales as L.

1. Simple random walk method

In this case, only thing to update after the walker has reached the aggregate is the status of the corresponding lattice site. Since we also utilize the use of HPP in all of our simulations, it is included in the scaling of the total CPU time per particle $t_{\mathrm{tot,RW}}$:

$$t_{\mathrm{tot,RW}} = n_{\mathrm{RW}}(L)t_{\mathrm{RW}}(L) + n_{\mathrm{HPP}}(L)t_{\mathrm{HPP}}(L)$$
$$= b_1 L^{\alpha_{\mathrm{RW}}} + b_2 L^{\alpha_{\mathrm{HPP}}}, \quad (19)$$

where b_1 and b_2 are constants. The subscripts RW and HPP refer to (simple) random walk and half-plane propagator, respectively. Our numerical estimates for the scaling exponents are $\alpha_{\mathrm{RW}} = 2.00 \pm 0.02$ and $\alpha_{\mathrm{HPP}} = 1.03 \pm 0.01$. The time t_{HPP} is found to be constant in L. The fit to the total CPU time per particle gives

$$t_{\mathrm{tot,RW}} \propto L^{1.94}. \quad (20)$$

In the limit $L \to \infty$ we thus expect $t_{\mathrm{tot,RW}}$ to be proportional to L^2 as also expected from simple scaling arguments for diffusion.

2. Square propagator method with AI algorithm

In the SQP walk with the AI algorithm, the time to take a single SQP leap is essentially constant in L. However, it is on the average about 2.5 times larger than the time needed for taking a single RW step. The index updating after each walker has stuck to the aggregate, on the other hand, increases rapidly with the system size, as the empty area where the change in the indices propagates increases. The total CPU time per particle in Eq. (12) is written as

$$t_{\mathrm{tot,AI}} = n_{\mathrm{SQP}}(L)t_{\mathrm{SQP,AI}}(L) + t_{\mathrm{u,AI}}(L)$$
$$+ n_{\mathrm{HPP}}(L)t_{\mathrm{HPP}}(L)$$
$$= b_3 L^{\alpha_{\mathrm{SQP}}} + b_4 L^{\gamma_{\mathrm{AI}}} + b_5 L^{\alpha_{\mathrm{HPP}}}, \quad (21)$$

where b_3, b_4 and b_5 are constants, and α_{SQP} is the scaling exponent for the number of step. Here we have to include the time $t_{\mathrm{u,AI}} \propto L^{\gamma_{\mathrm{AI}}}$ to describe the scaling of index updating between walkers. Numerical estimates give $\alpha_{\mathrm{SQP}} = 0.97 \pm 0.01$ and $\alpha_{\mathrm{HPP}} = 1.03 \pm 0.01$. Again, t_{HPP} does not depend on L. When ℓ_{max} is of the order of the system size, $\gamma_{\mathrm{AI}} = 1.94 \pm 0.07$. This clearly shows that in this case index updating between the walkers will eventually dominate the scaling. On the other hand, as also expected from theoretical considerations of Sec. III A since $t_{\mathrm{u,AI}} \propto \ell_{\mathrm{max}}^2$, we measure $\gamma_{\mathrm{AI}} = 0.97 \pm 0.06$ when $\ell_{\mathrm{max}} = \sqrt{L}$.

As discussed in Section III, this bottleneck can be avoided by imposing a maximum index ℓ_{max} so that indices larger than ℓ_{max} are not updated. Taking $\ell_{\mathrm{max}} \propto \sqrt{L}$ yields theoretically optimal behavior. Our final numerical numerical estimates for the effective scaling of our algorithm (based on fits to total CPU time per particle) with index cutoff ℓ_{max} in both cases is given by

$$t_{\mathrm{tot,AI}} \propto \begin{cases} L^{1.26}, & \text{when } \ell_{\mathrm{max}} = L/2; \\ L^{1.07}, & \text{when } \ell_{\mathrm{max}} = \sqrt{L}, \end{cases} \quad (22)$$

which shows that, in the unrestricted case $\ell_{\mathrm{max}} = L/2$, we are not yet in the scaling limit $L \to \infty$ of Eq. (12) where $t_{\mathrm{tot,AI}} \propto L^2$. The latter case $\ell_{\mathrm{max}} = \sqrt{L}$ is close to the predicted linear scaling behavior of Eq. (12) where $t_{\mathrm{tot,AI}} \propto L$.

However, even if one uses the AI algorithm without a cutoff ℓ_{max} in which case it scales as L^2, its prefactor b_4 in Eq. (21) is much smaller than that of the simple RW algorithm, b_1 in Eq. (19), thus yielding better efficiency.

3. Square propagator method with multi-grid algorithm

Within the multi-grid version of the SQP method the time needed to update the structure is negligible between each walk. On the other hand, the time to take a step is larger due to the greater effort needed in finding the proper square index. The CPU time given in Eq. (16) scales as

$$t_{\mathrm{tot,MG}} = n_{\mathrm{SQP}}(L)t_{\mathrm{SQP,MG}}(L) + t_{\mathrm{u,MG}}(L)$$
$$+ n_{\mathrm{HPP}}(L)t_{\mathrm{HPP}}(L)$$
$$= b_6 L^{\alpha_{\mathrm{SQP}}+\beta_{\mathrm{SQP}}} + b_7 L^{\gamma_{\mathrm{MG}}} + b_8 L^{\alpha_{\mathrm{HPP}}}, \quad (23)$$

where b_6, b_7, b_8 are constants. Here, there is an additional scaling term $t_{\mathrm{SQP,MG}}(L) \propto L^{\beta_{\mathrm{SQP}}}$ describing the time needed for finding the proper index using MG. In this case, we expect it to depend on L rather weakly due to the efficiency of the index search, and indeed we find numerically that $\beta_{\mathrm{SQP}} = 0.37 \pm 0.04$. Theoretically, one expects this dependence to be bounded from above by $\log(L)$ by construction. The same conclusion applies to the updating between walkers $t_{\mathrm{u,MG}}(L)$ that depends logarithmically on L, so that $t_{\mathrm{u,MG}}(L) \sim \mathcal{O}(\log(L))$, although in practice it is negligible when compared to the other terms. The other estimates for the scaling exponents are $\alpha_{\mathrm{SQP}} = 0.89 \pm 0.01$ and $\alpha_{\mathrm{HPP}} = 1.02 \pm 0.01$, and t_{HPP} is again independent of L.

To summarize, the asymptotic scaling in the $L \to \infty$ limit from our numerical estimate by fitting to $t_{\mathrm{tot,MG}}$ is given by

$$t_{\mathrm{tot,MG}} \propto L^{1.12}. \quad (24)$$

When compared to our theoretical discussion in Sec. III B it is apparent that we are not yet in the asymptotic scaling regime. We expect the scaling of $t_{\mathrm{u,MG}}$ to be a weak bounded function of L so that the asymptotic dependence of L would be linear.

Our numerical results for the scaling of the total simulation time per particle t_{tot} for the different algorithms (RW, AI, MG) are shown in Fig. 5. Both the optimal AI and the MG algorithm scale almost with the same exponent but in the former case, the prefactor is smaller. In our implementation, the AI algorithm is roughly twice as fast as the MG algorithm.

B. Application to HBAK

In the HBAK model, we examined performance of the SQP method in the growth regime. The HBAK model is implemented with a 1D vector $h(x,t)$, where $h(x,t)$ is the height of column x at time t. The diffusing particle walks until it steps on a site that is a nearest neighbor to an occupied site. Then the height of the corresponding column is increased by one unit. Note that if the particle arrives to the side of the step, this definition implies that the particle instantaneously slides down until it reaches a corner site. This "sliding-down" rule guarantees that the height profiles $h(x,t)$ obey the solid-on-solid restriction, and the steps form compact structures.

1. Simple random walk method

The simulation time can be expressed solely in terms of the number of walks and the time needed to take a step. Since we also utilize the use of HPP in our simulations, it is included in the scaling:

$$\begin{aligned} t_{\mathrm{tot,RW}} &= n_{\mathrm{RW}}(L)t_{\mathrm{RW}}(L) + n_{\mathrm{HPP}}(L)t_{\mathrm{HPP}}(L) \\ &= c_1 L^{\alpha_{\mathrm{RW}}} + c_2 L^{\alpha_{\mathrm{HPP}}}, \end{aligned} \tag{25}$$

where c_1 and c_2 are constants, and the time needed to perform a single step is independent of L. Our estimates for the scaling exponents are $\alpha_{\mathrm{RW}} = 2.10 \pm 0.02$ and $\alpha_{\mathrm{HPP}} = 1.08 \pm 0.02$. In the infinite system size limit the fit to $t_{\mathrm{tot,RW}}$ gives the scaling

$$t_{\mathrm{tot,RW}} \propto L^{2.07}. \tag{26}$$

We note that theoretically, $\alpha_{\mathrm{RW}} = 2$ for simple random walk. The leading term of Eq. (25) gives somewhat larger value 2.10 for the dominant behavior. However, both agree well with theoretical expectations.

2. Square propagator method

For the HBAK model there is no need for the jump index updating schemes or hierarchical structures used in the DLA simulations, and no additional updating is needed after the walk is finished. Since the growing aggregate forms compact structures with no overhangs, the proper index for taking a SQP step can be found using simple linear search (LI). After finding the proper index ℓ, taking a step using the square propagator is independent of L. The simulation time can now be expressed as follows:

$$\begin{aligned} t_{\mathrm{tot,LI}} &= n_{\mathrm{SQP}}(L)t_{\mathrm{SQP,LI}}(L) + n_{\mathrm{HPP}}(L)t_{\mathrm{HPP}}(L) \\ &= c_3 L^{\alpha_{\mathrm{SQP}}+\beta_{\mathrm{SQP,LI}}} + c_4 L^{\alpha_{\mathrm{HPP}}}, \end{aligned} \tag{27}$$

where c_3, c_4 are constants, and α_{SQP} and $\beta_{\mathrm{SQP,LI}}$ are the scaling exponents for the number of steps and the time needed to take a step, respectively. Our numerical estimates for the exponents are $\alpha_{\mathrm{SQP}} = 1.02 \pm 0.01$, $\beta_{\mathrm{SQP,LI}} = 0.13 \pm 0.03$, and $\alpha_{\mathrm{HPP}} = 1.06 \pm 0.01$. The leading behavior in the large L limit obtained by a fit to $t_{\mathrm{tot,LI}}$ is given

$$t_{\mathrm{tot,LI}} \propto L^{1.06}. \tag{28}$$

The dominant term of Eq. (27) gives $t_{\mathrm{tot,LI}} \propto L^{1.15}$. The weak dependence of the index search time $t_{\mathrm{SQP,LI}}$ is seen here as the final behavior slightly exceeds the linear scaling.

This means that one essentially saves a whole factor of L in the CPU time per particle by utilizing the use of the square propagator. The scaling of the total CPU time using the simple RW and SQP methods is illustrated in Fig. 6.

V. CONCLUSIONS AND DISCUSSION

To summarize, we have presented an exact method to speed up discrete random walk in complicated environments of absorbing sites. This is based on the discrete 2D probability distribution, which can be used to propagate walkers to (absorbing) sites outside of the periphery sites of a square of linear size s in one step, with no approximations. The method generalizes the previously used continuum propagation method [15] which is not exact for lattice systems. We have presented a detailed analysis of the performance of the square propagator (SQP) method as compared to traditional simulations of random walks. Two different types of growth models were considered to test the scaling of the CPU time consumption with system size, namely the fractal diffusion limited aggregation model [11] and a model for anisotropic growth of isolated steps [17] which produces compact finger-like structures. For both cases we have shown that with proper implementation of the SQP method, the speedup with respect to straightforward random walk approach is of the order of the linear system size L.

We also want to emphasize that the speed-up methods described here can be easily extended for other cases. The half-plane propagator can be calculated for an anisotropic case, including a drift term in the diffusion field [17]. This can also be done for the square propagator. In addition, one can calculate the probability for a particle starting at an arbitrary site within a rectangle of size $s_x \times s_y$ to enter a given site outside the rectangle. In the limit where s_x is fixed and s_y goes to infinity, the average number of steps needed to exit the strip increases only by a factor of two as compared to the isotropic square case. Thus, the use of a rectangle instead of square does not affect the scaling, but merely the prefactor. Finally, it should also be mentioned that time dependence of the random walk process can be easily extracted from the square propagator distribution, if it is needed for the problem at hand.

Acknowledgments: This work has been supported in part by the Academy of Finland through its Center of Excellence program. J. A. wishes to thank the Vaisala foundation for financial support.

APPENDIX A: SQUARE PROPAGATOR

Let ℓ denote the jump index corresponding a square of size $s = 2\ell + 1$. Let n be the index of a row, starting from the top, as illustrated in Fig. 1. Let T_n be the the transition matrix whose (i, j) element gives the probability for the walker at site i on n^{th} row to enter site j at the next row $n + 1$. All the sites at rows 0 and $s + 1$, as well as the sites at columns 0 and $s+1$ are considered to be absorbing sites here.

Let E be the escape matrix, which gives the probability to exit the n^{th} row and enter the $(n + 1)^{\text{th}}$ row. This matrix is independent of n. Clearly, $T_0 = 0$. At each of the s sites within the n^{th} row, the walker can step off the row, or step within the row to one of the neighboring sites. Thus T_n can be decomposed as the sum of probability two terms as follows:

$$T_n = E + T_n T_{n-1} E. \tag{A1}$$

Let D be the direct transition matrix $\frac{1}{4}I$ (I is the identity matrix), whose $(i, i)^{\text{th}}$ element is the probability to step from site i at the n^{th} row to site i at the $(n+1)^{\text{th}}$ row. In addition, let S be the transition matrix for the walker to step within the row. Then, $(I - S)$ is a tri-diagonal matrix, having entries 1 at the diagonal and entries $-\frac{1}{4}$ at off-diagonals, and it gives the probability to exit the n^{th} row (up or down).

Now, the particle can enter the $(n + 1)^{\text{th}}$ row directly, or it can take one or more steps within the row and then step the $(n + 1)^{\text{th}}$ row. Thus the escape matrix E can be written as a geometric series

$$E = D \sum_{i=0}^{\infty} S^i = D(I - S)^{-1}. \tag{A2}$$

We denote $(I - S) = \frac{1}{4}K$. Thus, we can write the recursion relation for the transition matrix T_n as

$$T_n = K^{-1} + T_n T_{n-1} K^{-1}. \tag{A3}$$

This equation can be formally solved as

$$T_n = (K - T_{n-1})^{-1}. \tag{A4}$$

Using our initial value $T_0 = 0$, we immediately get $T_1 = K^{-1}$, $T_2 = (K - K^{-1})^{-1}$, *etc.*

The probability distribution for the walker to exit the square from the last row $n = 2\ell+1$ starting from the row $n = \ell + 1$ can be obtained as the product of the T_n's as

$$P_{\text{SQP}} = T_{2\ell+1} T_{2\ell} \cdots T_{\ell+2} T_{\ell+1}, \tag{A5}$$

because we consider the situation where the walker starts at the center from the $(\ell + 1)^{\text{th}}$ row. Let us now define matrices f_n by [26]

$$T_n = f_{n+1}^{-1} f_n. \tag{A6}$$

Again, using the initial value for T_0 we get $f_0 = 0$. In addition, we choose $f_1 = I$. Writing Eq. (A3) in terms of the f_n's, we obtain the recursion relation

$$f_{n+2} - f_{n+1} K + f_n = 0. \tag{A7}$$

It is convenient to use the discrete z transform for such recursion relations. The z transform $\tilde{f}(z)$ of the sequence f_k $(k = 0, 1, 2, \ldots)$ is defined as

$$\tilde{f}(z) = \sum_{k=0}^{\infty} f_k z^{-k}. \tag{A8}$$

In particular, the transform for $f_k = c^k$ is needed here as well as for solving for the eigenvalues and eigenvectors. This is given by

$$\tilde{f}(z) = \sum_{k=0}^{\infty} c^k z^{-k} = \frac{z}{z - c}. \tag{A9}$$

Transforming both sides of Eq. (A7), we get

$$\tilde{f}(z) = (z^2 - zK + I)^{-1} zI. \tag{A10}$$

Denoting the roots of the denominator in Eq. (A10) by $K_\pm$, we have

$$K_\pm = \frac{K}{2} \pm \sqrt{(\frac{K}{2})^2 - I}. \tag{A11}$$

Next, we write the solution in terms of partial fractions to get

$$\tilde{f}(z) = (K_+ + K_-)^{-1}[(z - K_+)^{-1} - (z - K_-)^{-1}]. \tag{A12}$$

The inverse z transform can be found using Eq. (A9), and thus we have

$$f_n = (K_+^n - K_-^n)(K_+ - K_-)^{-1}. \tag{A13}$$

Now we can write the probability distribution P_{SQP} in terms of the matrices $K_\pm$:

$$\begin{aligned} P_{\text{SQP}} &= T_{2\ell+1} T_{2\ell} \cdots T_{\ell+2} T_{\ell+1} \\ &= f_{2\ell+2}^{-1} f_{2\ell+1} f_{2\ell+1}^{-1} f_{2\ell} \cdots f_{\ell+2}^{-1} f_{\ell+1} \\ &= f_{2\ell+2}^{-1} f_{\ell+1} \\ &= (K_+^{\ell+1} + K_-^{\ell+1})^{-1}. \end{aligned} \tag{A14}$$

This equation could already be used to numerically calculate the desired probability distribution P_{SQP}, but one can do better. Extracting the full solution in closed form saves a lot of computational effort for large matrices and increases numerical accuracy.

To this end, one can easily write recursion relation for the characteristic function of the eigenvalues of the matrix K. This can be solved using the z transform again. The result for the $k^{\rm th}$ eigenvalue is

$$\lambda_k = 4 - 2\cos\frac{\pi k}{s+1}, \text{ with } k = 1, 2, ..., s. \qquad \text{(A15)}$$

The same technique can be used to obtain the eigenvectors of K yielding for the $n^{\rm th}$ component of the eigenvector corresponding to the $k^{\rm th}$ eigenvalue, the result

$$x_n^k = \sqrt{\frac{2}{s+1}}\sin(\frac{\pi nk}{s+1}). \qquad \text{(A16)}$$

Having now calculated the eigenvectors and eigenvalues of the matrix K, we can obtain the solution for $P_{\rm SQP}$ in a closed form. The matrix K can be diagonalized, reducing significantly the elements needed in calculating $P_{\rm SQP}$. What follows is that starting from the center at the $(\ell+1)^{\rm th}$ column, the probability to enter the site at the $k^{\rm th}$ column just outside the square equals

$$P_{\rm SQP}(k) = \sum_{n=1}^{s} x_n^k f(\lambda_n) x_n^{\ell+1}. \qquad \text{(A17)}$$

where $f(\lambda) = (\lambda_+^{\ell+1} + \lambda_-^{\ell+1})^{-1}$, and $\lambda_\pm = \frac{\lambda}{2} \pm \sqrt{(\frac{\lambda}{2})^2 - 1}$. This is the desired SQP corresponding to Fig. 1.

[1] G. H. Weiss, *Aspects and Applications of the Random Walk*, (North Holland, 1994).
[2] J. P. Bouchaud and A. Georges, Phys. Rep. **195** (1990).
[3] A. Bunde and S. Havlin, *Fractal and Disordered Media*, vol. 1 (Springer-Verlag, Berlin, 1991).
[4] *Polymer Physics: 25 Years of the Edwards Hamiltonian*, edited by S. Bhattacharjee, World Scientific, Singapore (1992); *The Monte Carlo method in condensed matter physics*, edited by K.Binder, Springer, Heidelberg (1995).
[5] I. Goldhirsch and Y. Gefen, Phys. Rev. A **33**, 2583 (1986).
[6] G. H. Weiss and s. Havlin, Physica A **134**, 474 (1986).
[7] M. Woolfson, *An introduction to X-ray Crystallography* (Cambridge University Press, Cambridge, 1979).
[8] E. Ben-Jacob, O. Shochet, A. Tenenbaum, I. Cohen, A. Czirk, and T. Vicsek, Nature **368**, 46 (1994).
[9] R. L. Gibbs, C. W. Beason, and J. Beason, Am. J. Phys. **43**, 782 (1975).
[10] R. C. Ball and R. M. Bray, J. Phys. A **18**, L809 (1985).
[11] T. A. Witten and L. M. Sander, Phys. Rev. Lett. **47**, 1400 (1981).
[12] T. Vicsek, *Fractal Growth Phenomena* (World Scientific, Singapore, 1992); For a review, see *Fractals in Natural Science*, edited by T. Vicsek, M. Schlesinger, and M. Matsuhita (World Scientific, Singapore, 1993).
[13] H. Kaufman, A. Vespignani, B. B. Mandelbrot, and L. Woog, Phys. Rev. E **52**, 5602 (1995).
[14] S. Tolman, and P. Meakin, Phys. Rev. A **40**, 428 (1989).
[15] S. Torquato and In Chan Kim, J. Appl. Phys. **85**, 1560 (1999).
[16] B. B. Mandelbrot, B. Kol, and A. Aharony, Phys. Rev. Lett. **88**, 055501 (2002).
[17] J. Heinonen, I. Bukharev, T. Ala-Nissila, and J. M. Kosterliz, Phys. Rev. E, **57**, 6851 (1998).
[18] P. Meakin, J. Phys. A **18**, L661 (1985).
[19] J. Asikainen, J. Heinonen, S. Majaniemi, M. Dubé, and T. Ala-Nissila, unpublished.
[20] F. Family and T. Viscek, J. Phys. A **18** L75 (1985).
[21] A.-L. Barabási and H. E. Stanley, *Fractal Concepts in Surface Growth* (Cambridge University Press, Cambridge, 1995).
[22] P. Meakin, and F. Family, Phys. Rev. A **34**, 2558 (1986).
[23] P. Meakin, in *Fractals in Physics*, Proceedings of the Sixth Trieste International Symposium on Fractals in Physics, Trieste, Italy, 1985, edited by L. Pietronero and E. Tosati (North-Holland, Amsterdam, 1986).
[24] L. A. Turkevich and H. Scher, Phys. Rev. Lett. **55**, 1026 (1986).
[25] R. C. Ball, R. M. Brady, G. Rossi, and B. R. Thompson, Phys. Rev. Lett **55**, 1406 (1985).
[26] Y. Stroganov, private communication; J. Heinonen, *Ph. D. Thesis*, http://lib.hut.fi/Diss/2001/isbn9512252864.

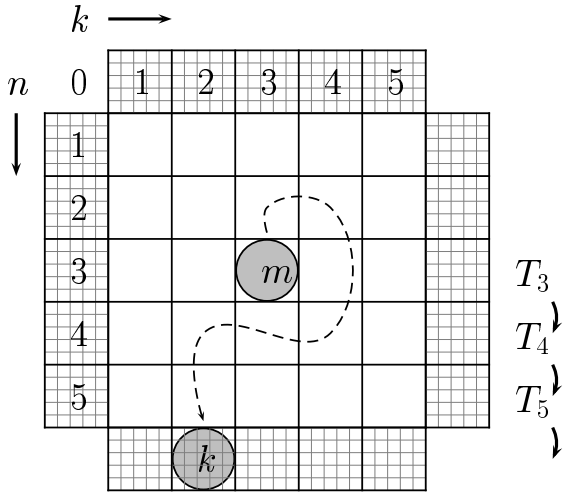
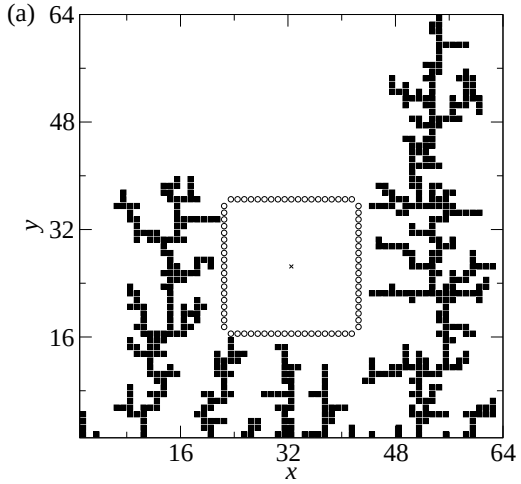


FIG. 1. Illustration of the transition matrix T_n for a square of size $s = 5$. Absorbing sites are shown as shaded. The product $(T_5T_4T_3)_{m,k}$ gives the probability for the walker starting at the center row site $(m, 3)$ to end up in any of the absorbing sites at the bottom $(k, 6)$. The random path (denoted by the dashed line) is replaced by a single SQP leap.



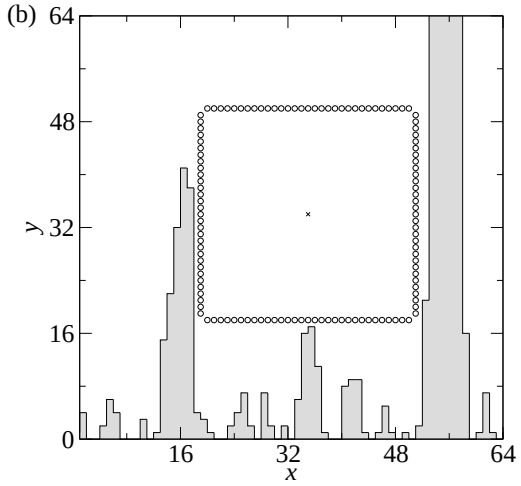


FIG. 2. (a) A sample DLA cluster in the growth regime. (b) A sample configuration generated by the HBAK model. In both figures, the diffusing particle denoted by a cross can arrive at any of the sites denoted by open circles by a single SQP leap.

	1	1	1	1	1	
1	1	0	0	0	1	1
1	0	0	−1	0	0	1
1	0	−1	−2	−1	0	1
1	0	0	−1	0	0	1
1	1	0	0	0	1	1
	1	1	1	1	1	

FIG. 3. Illustration of the shells used in the incremental index updating. The grey occupied site at the center has the index $\ell = -2$ and its vacant neares neighbors posses the value $\ell = -1$, etc.

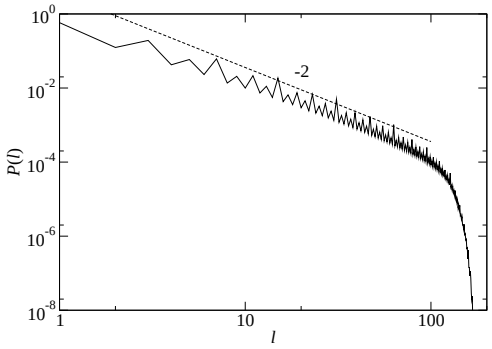


FIG. 4. The distribution $P(\ell)$ giving the probability for using a jump index ℓ in cylindrical DLA. The dashed line indicates the slope -2. System size in the simulation was $L = 512$.

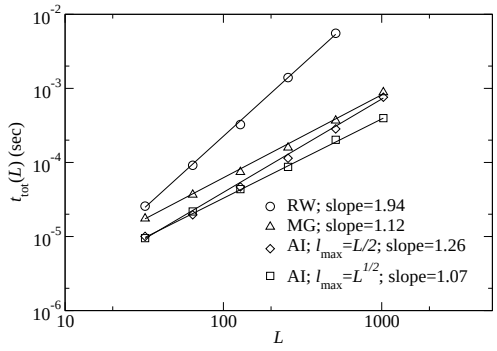


FIG. 5. Scaling of the total CPU time $t_{\rm tot}$ per particle with the system size in the DLA model with different algorithms [see Eqs. (19), (21) and (23)].

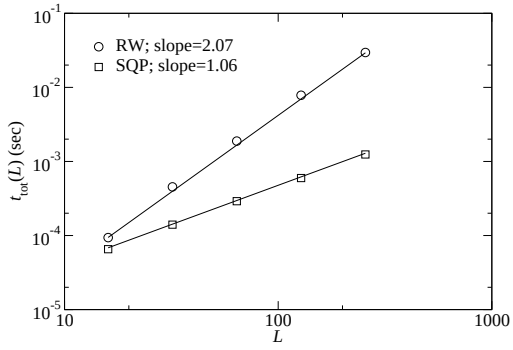


FIG. 6. Scaling of the total CPU time $t_{\rm tot}$ per particle with the system size in the HBAK model with different algorithms [see Eqs. (25) and (27)].