# Percolation threshold, Fisher exponent, and shortest path exponent for four and five dimensions

Gerald Paul,[1,*] Robert M. Ziff,[2] and H. Eugene Stanley[1]
[1]*Center for Polymer Studies and Department of Physics, Boston University, Boston, Massachusetts 02215*
[2]*Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, Michigan 48109-2136*
(Received 10 January 2001; published 23 July 2001)

We develop a method of constructing percolation clusters that allows us to build very large clusters using very little computer memory by limiting the maximum number of sites for which we maintain state information to a number of the order of the number of sites in the largest chemical shell of the cluster being created. The memory required to grow a cluster of mass $s$ is of the order of $s^{\theta}$ bytes where $\theta$ ranges from 0.4 for two-dimensional (2D) lattices to 0.5 for six (or higher)-dimensional lattices. We use this method to estimate $d_{\min}$, the exponent relating the minimum path $\ell$ to the Euclidean distance $r$, for 4D and 5D hypercubic lattices. Analyzing both site and bond percolation, we find $d_{\min}=1.607\pm0.005$ (4D) and $d_{\min}=1.812\pm0.006$ (5D). In order to determine $d_{\min}$ to high precision, and without bias, it was necessary to first find precise values for the percolation threshold, $p_c$: $p_c=0.196889\pm0.000003$ (4D) and $p_c=0.14081\pm0.00001$ (5D) for site and $p_c=0.160130\pm0.000003$ (4D) and $p_c=0.118174\pm0.000004$ (5D) for bond percolation. We also calculate the Fisher exponent $\tau$ determined in the course of calculating the values of $p_c$: $\tau=2.313\pm0.003$ (4D) and $\tau=2.412\pm0.004$ (5D).

DOI: 10.1103/PhysRevE.64.026115 PACS number(s): 64.60.Ak, 64.60.Fr, 05.45.Df

## I. INTRODUCTION

Percolation is a standard model for disordered systems [1,2]. In percolation systems, sites or bonds on a lattice are populated with probability $p$. The value of $p$ at which infinite clusters are formed is known as the critical probability or percolation threshold $p_c$. The shortest path exponent $d_{\min}$ is defined by the relation [2,4,5]

$$\langle \ell \rangle \sim r^{d_{\min}}, \tag{1}$$

where $r$ is the Euclidean distance between two sites on a cluster and $\ell$ is the length of the shortest path traveling along occupied sites and bonds in the percolation cluster. The length of this path is also known as the ''chemical distance'' between the sites. We can also write

$$\langle r \rangle \sim \ell^{z}, \tag{2}$$

which defines the exponent $z=1/d_{\min}$. With the exceptions of $d\geqslant 6$ (where $z=1/2$) and $d=1$ (where $z=1$), $z$ is not known exactly. The most common method of determining $z$ numerically (and the one we will use) is to grow clusters, calculating the average distance $\langle r \rangle$ of sites in the cluster from the seed of the cluster as a function of chemical distance $\ell$ from the seed. In order that finite size effects do not play a role, the lattice must be large enough such that the clusters that are grown do not reach the boundaries of the lattice.

Because corrections to scaling decrease with increasing $\ell$, the larger the value of $\ell_{\max}$ (the value of $\ell$ at which we stop the growth), the more accurately we can estimate $z$. The limitations on the size $\ell_{\max}$ to which the clusters can be grown have been the computer memory available for the simulation and the computer processing power needed to build these clusters. The method of ''data blocking'' [6,7] has helped ameliorate the need for large amounts of memory. In this method, the lattice is logically divided into blocks; memory for a block is not allocated until the lattice grows into that block. The data blocking method has been used recently to obtain precise estimates for the percolation threshold and associated exponents for bond and site percolation on a number of lattices [7,8]. Ultimately, however, although sufficient computer power is available to build larger clusters, the cluster size is limited by the amount of memory available. This becomes particularly true as the dimension of the lattice $d$ increases since at criticality the cluster becomes less dense as $d$ increases [9]. To reach the same cluster mass or $\ell_{\max}$, we must have larger lattices.

In this paper we describe a method of constructing clusters that dramatically reduces the memory requirements needed to grow large clusters relative to previous methods. Using this method of building large clusters, we estimate $z$ for hypercubic lattices in four and five dimensions. The study of critical properties in higher dimensions is important because one can use the results to test relations, which are conjectured to hold in all dimensions (hyperscaling relations) and exponents that are believed to be the same in all dimensions (superuniversal exponents). The current best estimates of $d_{\min}$ for four and five dimensions, $1.63\pm0.03$ [3] and 1.8 [1], respectively, are of relatively low precision compared to the estimates available in two and three dimensions $1.1307\pm0.0004$ and $1.374\pm0.006$, respectively [2,4].

## II. CLUSTER GENERATION

One method of cluster generation is the Leath method [10]. In this method a site is chosen as the seed site of the cluster. Using a random number generator and a given bond occupation probability, one determines whether the bonds connected to the seed site are occupied or not [11]. If a bond is occupied, the site to which this bond connects is consid-

*Electronic address: gerryp@bu.edu

 ©2001 The American Physical Society

ered to be part of the cluster and becomes a ''growth site.'' These sites are at chemical distance of unity from the seed site; all sites at the same chemical distance $\ell$ from the seed site are considered to be in ''chemical shell $\ell$.'' The process is then repeated for each of these growth sites with the next set of growth sites being at chemical distance 2 from the seed site. The cluster continues to grow until the growth stops naturally, the growth is terminated by the sides of the $d$-dimensional lattice of edge $L$, or the maximum chemical distance $\ell_{\max}$ is reached.

We use the Leath method to construct clusters, but we keep track of which bonds are occupied and which sites have been visited by a method different from that traditionally used. Traditionally, this state information is stored in an array of size equal to the number of lattice sites. In the data blocking method, memory usage can be improved by allocating blocks in this array dynamically. Vollmayr [12] eliminated the use of this array, storing status of visited sites in a data structure thus reducing memory requirements to grow a cluster of mass $s$ to $O(s)$. We extend the approach of Ref. [12] further, reducing the memory required to $O(s^{\theta})$ where $\theta$ ranges from 0.4 for two-dimensional lattices to 0.5 for six (or higher)-dimensional lattices.

To see how this can be done, we first consider the uses of this state information.

(a) *Occupancy status*. Information concerning whether a site/bond is occupied is maintained so that it is the same, independent of when it is accessed in the growth process. For example, we would not generate a cluster with the proper statistics if we treated a bond as occupied during one stage of the cluster growth and then treated it as empty during a later stage.

(b) *Visited status*. Information concerning whether a site has been visited or not is maintained in order that (a) we do not multiply count the presence of a site in the cluster and (b) we do not retrace our steps during cluster generation, causing the growth process to never end.

### A. Occupancy status

We address the need to maintain information about whether a bond is occupied or not by using a random number generation scheme in which the random number associated with a bond is determined by the location of the bond in the lattice [12] and the orientation of the bond. This is done by first assigning a unique number $n$ to any *site* in the lattice as follows. Let $(x_1,x_2,x_3 \ldots x_d)$ be the coordinates of the site in the lattice, and let $(L_1,L_2,L_3,\ldots,L_d)$ be the lengths of the sides of the lattice. Then

$$n(x_1,x_2,x_3,\ldots,x_d) = [(\{[(x_1L_2)+x_2]L_3\}+x_3,\ldots)L_d+x_d] \tag{3}$$

assigns a unique number to any site in the lattice. We assign a unique number $n'$ to any *bond* in the lattice by defining

$$n'(x_1,x_2,x_3,\ldots,x_d,o)=[n(x_1,x_2,x_3,\ldots,x_d)d]+o, \tag{4}$$

where $o$ is the orientation of a bond attached to site $x$ (assuming values 0 to $d-1$).

Furthermore we want to assign unique numbers to bonds over many different realizations. We then define

$$n''(x_1,x_2,x_3,\ldots,x_d,o,m) = [n'(x_1,x_2,x_3,\ldots,x_d,o)M]+m, \tag{5}$$

where $m$ is the number of the realization and $M$ is the maximum number of realizations we plan to create.

We then generate a 64-bit random number $R$ using an encryptionlike algorithm $f(n'')$ [13] using $n''$ as its input,

$$R=f(n''). \tag{6}$$

A bond is occupied if $R>2^{64}p$. In practice, because for large lattices and a large number of realizations $n''$ is greater than $2^{64}$, the maximum size of the input to the random number algorithm, we actually determine the random number in two steps,

$$\overline{R}=f(\{[f(n)d]+o\}M+m). \tag{7}$$

That is, we first create an intermediate random number based only on the coordinates of the bond and then create the final random number based on the intermediate random number, the orientation of the bond and the realization number. Using the test described in [14], we confirm that, within statistical error, our algorithm generates unbiased random numbers. This test is important because there is only a small difference between the inputs to the random number generator for neighboring sites. Any correlations between the outputs would cause incorrect results [14]. The generation of random numbers using Eq. (7) is slower than congruence or shift register techniques [14] but is somewhat compensated by eliminating the processing done to store and access bond state when maintained in an array. In any case, the net effect of using this approach is about a factor of 5 increase in calculation time because of the slowness of the encryption-like random number generator that we used.

### B. Visited status

We address the need to maintain information about whether a site has been visited or not by storing information about visited sites in a data structure. Each entry in the data structure contains the coordinates of the site, the chemical shell of the site, and a bit map with one bit for each direction from which the site can be visited. The data structure can be accessed as a ''circular list'' (first-in-first-out queue) so entries can be added and deleted. Since a site can be visited from different directions, we must ensure that a site is counted only once and that backtracking does not occur. To accomplish this, before adding a site to the list of growth sites, we first check to see if it is already in the list.

1. If it is already on the list, we do not add a new entry but, in the entry for the site already in the list, we do set the bit corresponding to the orientation of the connected bond which was traversed to visit the site.

2. If it is not in the list, we add it (storing the coordinates and chemical length and setting the bit corresponding to the direction from which the site was visited).

When we are about to process the entry for a growth site, we only count the site once in the mass of the cluster, and only attempt to grow the cluster in directions other than those from which the site was visited. In this way we avoid backtracking along already traveled paths. If the data structure had to be searched sequentially every time we were about to create a growth site, the time needed would make this approach impractical. In Ref. [12], the data structure was maintained as a binary tree in order to reduce search time. We use the faster ''hash table'' method [15] to access entries for the visited sites.

The hashing technique works as follows: A key $K$ is associated with each entry of the data structure. We use a function $h(K)$ to map the key into a ''slot'' at offset $h(K)$ in a ''hash table.'' If the slot in the table is not already used, we store the number or address of the entry in this slot; if the slot is used (this is referred to as a ''collision'') we add the entry to a chain of entries all of which map to the same value $h(K)$. Ideally the function $h(K)$ maps the keys uniformly over the slots in the table so we obtain few long chains. If we use a hash table of size $M=2^m$, where $m$ is an integer and choose $K$ as the unique number $n$ of the site, an effective hashing function is [15]

$$h(n)=\frac{1}{2^{w-m}}[(nC)\bmod 2^w], \tag{8}$$

where $w$ is the word size(in bits) of our computer and the hash constant, $C$ is the least significant $w$ bits of the product of $2^w$ and the ''golden ratio,'' $(\sqrt{5}-1)/2$. Thus $h(n)$ yields the upper $m$ bits (shift right $w-m$ bits) of the result of taking the lower $w$ bits of the product of the unique site number and the hash constant $C$. We implement the ability to chain entries in the data structure by defining another field in the data structure entry that serves as a chain pointer field. To find an entry in the data structure for a site, we calculate the unique site number using Eq. (3), find the offset in the hash table using Eq. (8), and then walk the chain of entries to find the entry with the desired coordinates. If we make the size of the hash table equal to the size of the site data structure, we find the average number of hash ''collisions'' to be less than two so we can determine if a site has been visited very efficiently.

This approach of keeping the status of visited sites in a special data structure (not in the lattice array) applies to any lattice model. In the case of growing percolation clusters we can further reduce the amount of memory needed significantly. This is accomplished by recognizing that a site which is multiply visited is done so during the growth of a single chemical shell. This is the key insight that allows us to reduce the memory requirement and can be confirmed by considering the bonds adjacent to a site in a lower chemical shell: (i) an occupied bond adjacent to a site in a lower chemical shell cannot be a path to revisit that site because we do not backtrack and (ii) an unoccupied bond adjacent to a site in a lower chemical shell cannot be on the path to revisit that site. Sites in the same chemical shell can, however, be visited by multiple paths as shown in Fig. 1(a). Thus we need only keep state information about growth sites, which themselves have not yet been used to create entries for the next chemical shell. The number of such sites at any point in the growth process will be of the order of the size of the current chemical shell.

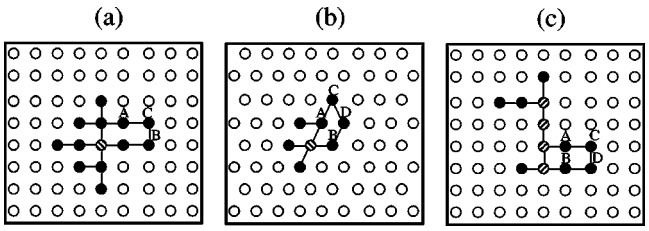


FIG. 1. Examples of cluster growth at the beginning of the population of sites at chemical distance 3 from the seed site. The seed sites are denoted by striped circles. (a) Example of a square lattice in which a site $C$ is multiply visited from sites $A$ and $B$. (b) Example of a triangular lattice in which site $C$ can be multiply visited from sites $A$ and $D$ and in which site D can be multiply visited from sites $B$ and $C$. (c) Example in which the cluster is grown from multiple seeds. Site $C$ can be multiply visited from sites $A$ and $D$; site $D$ can be multiply visited from sites $B$ and $C$.

The discussion so far has been for hypercubic lattices. For these lattices, we ensure that we did not double count site or backtrack by maintaining information about growth sites, which themselves have not yet been used to create entries for the next chemical shell and then checking for duplicates. More generally (e.g., for triangular lattices), the situation is a little more complicated as shown in the example in Fig. 1(b). A similar situation is shown in Fig. 1(c), where we grow a cluster from multiple seeds. To treat both types of situation, we must maintain (i) state information about growth sites that themselves have not yet been used to create entries for the next chemical shell and (ii) state information about all sites in the chemical shell previous to the one being built. Before we add a site to the list of growth sites, we check if it is already present in the previous shell; if it is, we do not add it.

The size of a chemical shell can be estimated as follows. The chemical distance $\ell$ scales with the mean Euclidean radius of the cluster $r$ as

$$\ell \sim r^{d_{\min}}, \tag{9a}$$

while the cluster mass (the number of sites in the cluster) $s$ scales as

$$s \sim r^{d_f}, \tag{9b}$$

where $d_{\min}$ has values 1.13 and 2 for $d=2$ and 6, respectively [2,4,16,17]; $d_f$, the fractal dimension of the cluster mass, has the exact values $91/48=1.89$ and 4 for $d=2$ and 6, respectively [1,2]. Then

$$s \sim \ell^{d_f/d_{\min}}, \tag{10}$$

and

TABLE I. Simulation parameters and results for $p_c$ and the Fisher exponent $\tau$.

| Dimension | Type | No. of realizations | $s_{\max}$ | $p_c$ | $\tau$ |
|---|---|---|---|---|---|
| 4 | Bond | $10^8$ | 131 073 | $0.160130 \pm 0.000003$ | $2.313 \pm 0.003$ |
| | Site | $10^8$ | 131 073 | $0.196889 \pm 0.000003$ | |
| 5 | Bond | $10^8$ | 16 383 | $0.118174 \pm 0.000004$ | $2.412 \pm 0.004$ |
| | Site | $10^8$ | 16 383 | $0.14081 \pm 0.00001$ | |

$$ds \sim \ell^{(d_f/d_{\min})-1} d\ell = (s^{d_{\min}/d_f})^{(d_f/d_{\min})-1} d\ell = s^{1-(d_{\min}/d_f)} d\ell. \tag{11}$$

Setting $d\ell = 1$, we find the size of the outermost chemical shell of a cluster of mass $s$ scales as

$$s_{\text{shell}}(s) \sim s^{\theta}, \tag{12}$$

where $\theta = 1 - d_{\min}/d_f$.

The values of $\theta$ range from $\approx 0.4$ to 0.5 for $d=2$ to $d=6$. Thus the size of the data structure to contain the visited status is only of the order of the square root of the size of the cluster size because we only store status for the largest chemical shell.

## III. PERCOLATION THRESHOLD AND FISHER EXPONENT

In order to determine $d_{\min}$ to high precision and without bias, it is necessary to first find values of the percolation threshold substantially more precise than previously known (in most cases). To determine these thresholds we used the method of measuring cluster-size statistics of individual clusters grown on large virtual lattices as described in [7]. The data-blocking method [6] used involves assigning memory to parts of the lattice only when the cluster grows into it. With the data-blocking method, like the hashing method, a table is used to access a data structure but in this case, the data structure entries represent blocks of sites instead of individual sites; there are no collisions, but some memory is wasted. The advantage of using the data-blocking method as opposed to the one proposed in this work is that the states of all sites are recorded (as described in the Appendix), so it allows using a fast random number generator. The data-blocking method allows lattices of sufficient size to keep finite-size effects under control, with sufficient speed to achieve good statistics. (The hashing method described in this paper could also have been used for this calculation.)

In four dimensions (4D), we use a virtual lattice of $512^4$ sites, broken up into blocks of $16^4$ sites each. In 5D, the virtual lattice of size $128^5$ is divided into blocks of size $8^5$. The cluster-size cutoff $s_{\max}$ is $2^{17} = 131{,}072$, and $2^{14} = 16\,384$ for 4D and 5D, respectively. The threshold is determined as the value of $p$ that leads to the cluster size distribution $n_s$ best following a power law $n_s \sim s^{-\tau}$. Simulating about $10^8$ clusters for each case, and using the data analysis techniques employed in [7], we find

$$p_c = \begin{cases} 0.196889 \pm 0.000003 & [\text{4D site}] \\ 0.160130 \pm 0.000003 & [\text{4D bond}] \\ 0.14081 \pm 0.00001 & [\text{5D site}] \\ 0.118174 \pm 0.000004 & [\text{5D bond}]. \end{cases} \tag{13}$$

Also, for $\tau$ we find the values

$$\tau = \begin{cases} 2.313 \pm 0.003 & [\text{4D}] \\ 2.412 \pm 0.004 & [\text{5D}]. \end{cases} \tag{14}$$

These results are more precise than some of the published values for $p_c = 0.16013 \pm 0.00012$ [3], $0.1407 \pm 0.0003$ [18], and $0.11819 \pm 0.00004$ [19] for 4D bond, 5D site, and 5D bond percolation, respectively, and for $\tau = 2.41$ for 5D percolation; for 4D site percolation, Ballesteros *et al.* [20] found the comparably precise value $p_c = 0.196901 \pm 0.000005$ (just slightly higher than ours) and $\tau = 2.3127 \pm 0.0007$. The analytic $\epsilon$-expansion method has also been used to estimate critical exponents [21–25]. Using the third-order $\epsilon$-expansion for $\eta$ of Ref. [21] and the scaling relations $\tau = d/d_f + 1$ and $d_f = (d + 2 - \eta)/2$, we find $\tau = 2.348$ and 2.421 for 4D and 5D, respectively. These values are fairly close to the values we measured.

All simulation parameters and our results are summarized in Table I. The precision of our results is sufficiently high that we expect that statistical errors in $p_c$ will not have an effect on our value of $d_{\min}$.

## IV. SHORTEST PATH EXPONENT

To calculate the shortest path exponent, we ran simulations at the percolation thresholds found above. We stopped cluster growth at $\ell_{\max} = 2048$ for 4D bond and site percolation and $\ell_{\max} = 1024$ for 5D bond and site percolation. We simulated $73 \times 10^6$, $39 \times 10^6$, $105 \times 10^6$, and $20 \times 10^6$ realizations for 4D bond, 4D site, 5D bond, and 5D site percolation, respectively. During our simulations, we kept track of the maximum and minimum lattice points to which our clusters extended. Using this information, we determined the size of the lattice that we would have needed to build if we had been using conventional memory techniques. For $d=5$ the lattice would have had sides of length $L = 245$ resulting in approximately $900 \times 10^9$ lattice sites ($\approx 1$ TB memory); the actual memory used was less than $\approx 10^6$ (1 MB), six orders of magnitude smaller.

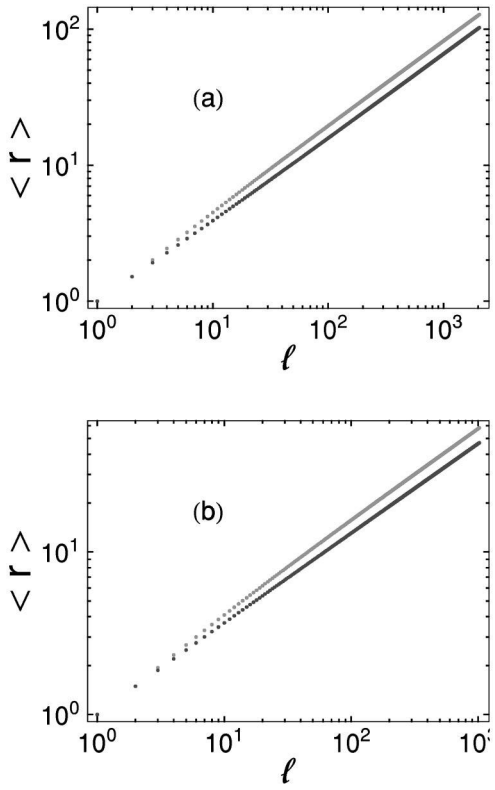


FIG. 2. Euclidean distance $\langle r \rangle$ versus chemical distance $\ell$ for site percolation (upper line) and bond percolation (lower line) for (a) 4D and (b) 5D. The slightly different apparent slopes of the plots for bond and site cases can be attributed to different values of the correction-to-scaling parameters.

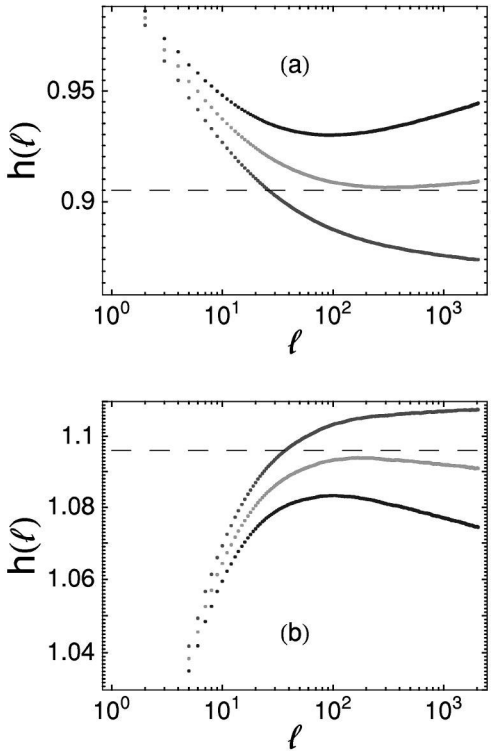


FIG. 3. $h(\ell) \equiv \langle r \rangle / \ell^z$ versus $\ell$ for (a) 4D bond percolation for values of (from top to bottom) $z' = 0.615$, 0.620, and 0.625 and (b) 4D site percolation for values of (from top to bottom) $z' = 0.623$, 0.625, and 0.627. The dashed horizontal lines are provided as guides to the eye to allow one to better see that, for large $\ell$, the middle plots of $h(\ell)$ in (a) and (b) are increasing and decreasing, respectively.

Figure 2(a) shows plots of $\langle r \rangle$ for 4D site and bond percolation, while Fig. 2(b) shows plots of $\langle r \rangle$ for 5D site and bond percolation. While the plots resemble straight lines, the effects of corrections to scaling are, in fact, considerable. One customarily assumes that corrections to scaling have the functional form [1,2,4,5]

$$\langle r \rangle \sim \ell^z (1 + A\ell^{-\Delta} + \cdots), \tag{15}$$

where the constant $A$ depends on the dimension, lattice type and percolation type (bond or site) but the exponent $\Delta$ depends only on dimension. Let

$$h(\ell) \equiv \frac{\langle r \rangle}{\ell^{z'}} \sim \ell^{z-z'} (1 + A\ell^{-\Delta} + \cdots), \tag{16}$$

where $z'$ is an estimated value of $z$. If $\ell_{\max}$ were infinitely large, we could determine $z$ as the value of $z'$, which results in a plot of $h(\ell)$ that asymptotically approaches a constant (i.e., has zero slope as $\ell \to \infty$); however, since $\ell_{\max}$ is finite, we may obtain misleading results if we determine $z$ in this manner. Nevertheless, we can use this approach to determine bounds on $z$.

To see how this is accomplished, first consider Fig. 3(a), in which we plot $h(\ell)$ for 4D bond percolation for various values of $z'$. From this figure and Eq. (16) it is clear that $A$ is positive. Hence, we know that if for large $\ell$ the slope of $h(\ell)$ becomes an increasing function, the leading power-law term $\ell^{z-z'}$ will dominate because $z > z'$. Thus a lower bound on $z$ is that value of $z'$ at which $h(\ell)$ asymptotically becomes an increasing function. From Fig. 3(a) this value is 0.620.

We can proceed similarly by considering site percolation in 4D, plotting $h(\ell)$ for 4D site percolation for various values of $z'$ in Fig. 3(b). From these plots it is clear that $A$ for bond percolation is negative. Hence we know that if for large $\ell$ the slope of $h(\ell)$ becomes a decreasing function, we are seeing the leading power-law term $\ell^{z-z'}$ dominate because $z < z'$. Thus an upper bound on $z$ is that value of $z'$ at which $h(\ell)$ asymptotically becomes a decreasing function. From Fig. 3(b) this value is 0.625.

Proceeding in the same manner for site and bond percolation in 5D [see Fig. 4(a,b)], we find that the constant $A$ is positive for both bond and site percolation, allowing us to determine only an upper bound of $z = 0.5515$ (the lower of the upper bounds for site and bond percolation). While this method of finding bounds on $z$ by identifying the value of $z'$ at which the slope of $h(\ell)$ changes sign does not always yield both upper and lower bounds, it has the advantage that it does not require any estimation of the parameters $A$ and $\Delta$ in Eq. (12) and, in fact, is somewhat insensitive to the exact form of the the corrections-to-scaling terms.

We also analyze our data using another more commonly used method [4–6]. That method is to plot the effective ex-

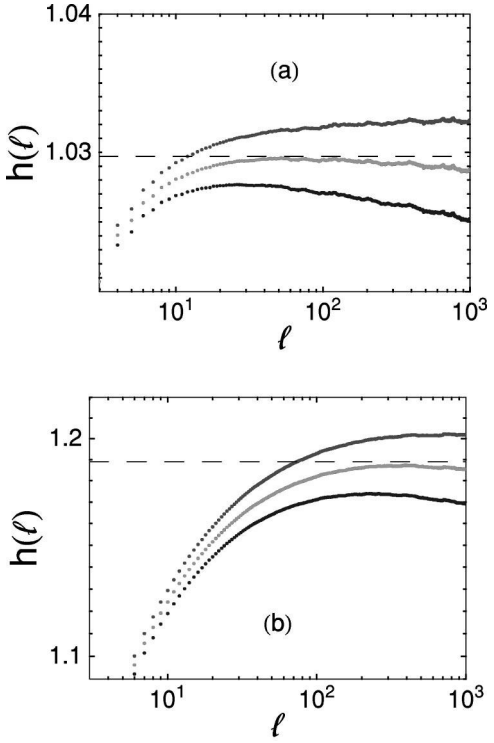


FIG. 4. $h(\ell) \equiv \langle r \rangle / \ell^z$ versus $\ell$ for (a) 5D site percolation for values of (from top to bottom) $z' = 0.5510$, 0.5515, and 0.5520, and (b) 5D bond percolation for values of (from top to bottom) $z' = 0.5595$, 0.5615, and 0.5635. The dashed horizontal lines are provided as guides to the eye to allow one to better see that, for large $\ell$, the the middle plots of $h(\ell)$ in (a) and (b) are decreasing.

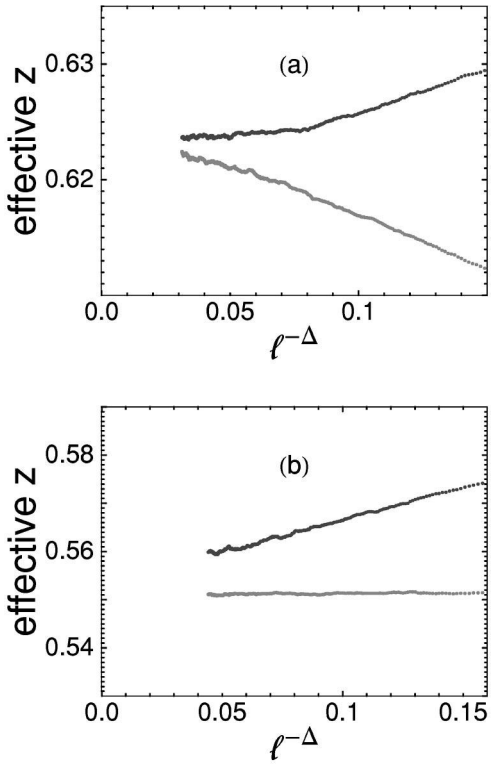


FIG. 5. Effective exponent $z$ versus $1/\ell^{-\Delta}$ with $\Delta = 0.5$ for bond percolation (upper line) and site percolation (lower line) for (a) 4D and (b) 5D.

ponents $z(\ell)$ between points $\ell$ and $2\ell$ versus $\ell^{-\Delta}$ using an estimated value of $\Delta$ that yields the straightest line. The effective exponent $z(\ell)$ between two points at $\ell$ and $2\ell$ is the value of the slope between these points in a log-log plot of $\langle r(\ell) \rangle$

$$z(\ell) = \frac{\ln[\langle r(2\ell) \rangle] - \ln[\langle r(\ell) \rangle]}{\ln[2\ell] - \ln[\ell]} = \frac{\ln[\langle r(2\ell) \rangle / \langle r(\ell) \rangle]}{\ln[2]}. \tag{17}$$

The $\ell = 0$ intercept of a plot of $z(\ell)$ will be an estimate for $z$, and the slope will be proportional to $A$. Our best estimate for $\Delta$ for $d = 4$ and $d = 5$ is $0.4 < \Delta < 0.6$, so we use a value of $\Delta$ of 0.5 and plot $z(\ell)$ for 4D site and bond percolation in Fig. 5(a) and 5D site and bond percolation in Fig. 5(b). In Fig. 5(a), the fact that the slopes of the lines change suggests that we are seeing the effects of both the correction-to-scaling term in Eq. (15) as well as higher order terms, which become less significant at larger values of $\ell$. In general, it is more efficient to generate smaller clusters and more of them, rather than fewer, larger ones. However, if the corrections to scaling are not well understood or large, then one must build the largest clusters possible. As we see here, strong corrections to scaling are present in 4D percolation where the plots of effective slope change at large $\ell$. If we had used smaller clusters using traditional memory-management techniques we would have obtained incorrect results. In Fig. 5(a), the almost horizontal plot for site percolation indicates that the amplitude, $A$, of the correction-to-scaling term is very small. From these plots and our estimates above of bounds on $z$, we estimate

$$z = \begin{cases} 0.622 \pm 0.002 & [4\mathrm{D}] \\ 0.552 \pm 0.002 & [5\mathrm{D}]. \end{cases} \tag{18}$$

In terms of $d_{\min}$, this corresponds to

$$d_{\min} = \begin{cases} 1.607 \pm 0.005 & [4\mathrm{D}] \\ 1.812 \pm 0.006 & [5\mathrm{D}]. \end{cases} \tag{19}$$

The previously published values for $d_{\min}$ obtained by numerical methods are $1.63 \pm 0.03$ [3] and 1.8 [1] for 4D and 5D. Thus our estimates of $d_{\min}$ are of higher accuracy than the existing ones and have accuracy comparable to that for the estimate of $d_{\min}$ in 3 dimensions, $1.374 \pm 0.006$ [5]. Using the rational form of the second order $\epsilon$-expansion, of Ref. [25] [in which a cubic term was added so that $d_{min}(d = 1) = 1$] we find $d_{\min} = 1.568$ and 1.803 for 4D and 5D, respectively. The $\epsilon$-expansion result for 5D is just a small amount outside the error bar of our result. The agreement can be improved by making a [2,1] Pade approximation to the $\epsilon$-expansion series; using this technique, we find $d_{\min} = 1.614$ and 1.814 for 4D and 5D, respectively, in good agreement with our results.

Our results and all simulation parameters are summarized in Table II.

TABLE II. Simulation parameters and results for the spreading exponent $z$ and shortest path exponent $d_{\min}$.

| Dimension | Type | $p_c$ | No. of realizations | $\ell_{\max}$ | $z$ | $d_{\min}$ |
|---|---|---|---|---|---|---|
| 4 | Bond | 0.160130 | $73\times10^6$ | 2048 | $0.622\pm0.002$ | $1.607\pm0.005$ |
| | Site | 0.196889 | $39\times10^6$ | 2048 | | |
| 5 | Bond | 0.118174 | $105\times10^6$ | 1024 | $0.552\pm0.002$ | $1.812\pm0.006$ |
| | Site | 0.14081 | $20\times10^6$ | 1024 | | |

## V. DISCUSSION

We have developed a technique that allows us to build very large percolation clusters using very little memory. In fact, using the method described here, relative to computer processing power available today and in the foreseeable future, computer memory is no longer a constraint on building percolation clusters near the percolation threshold. The critical computer resource thus becomes solely processing power. For example, by extrapolating from our simulations, we find that with our method, with less than $10^8$ bytes of memory, we could build a 5D cluster of $10^{12}$ sites, which would have required a lattice of $10^{17}$ sites, and reach a value of $\ell_{\max}$ of $10^7$ (versus the 1024 cutoff we used in our simulations). But the time to build a single trillion-site cluster would be about 2000 h on current workstations. As processor speeds increase, our technique for reducing memory usage should allow critical exponents and constants to be determined with greater precision. Current techniques of growing clusters, including the one described in this paper, require computer processing resource of $O(s)$, where $s$ is the size of the cluster grown.

We note that the technique we have developed is useful when we can count the quantities in which we are interested as we build the cluster (e.g., cluster mass, average distance to sites in a chemical shell). On the other hand, it is not clear how we could calculate the mass of the backbone, for example, using our method because current methods of determining backbone mass require knowing all the sites in the cluster, not just those in the current chemical shell. To obtain backbone properties one could, however, reduce memory required to $\sim s$ (versus $L^d$) by maintaining information about all visited sites (not just those in the current chemical shell) in a data structure as opposed to maintaining the full lattice data structure [12]. In the Appendix, we describe an alternative method of cluster generation that can be used when information about all visited sites must be maintained.

Finally, it is useful to compare our method with the Hoshen-Kopelman method [26], which constructs *all* clusters in a $d$-dimensional lattice by successively populating $d-1$ dimensional slices of the lattice. Memory is used to store the last and current slice of the lattice so the memory needed scales as $L^{d-1}$. The Hoshen-Kopelman method is much less memory efficient than the method presented here, and becomes less effective as the dimension increases since $L^{d-1}/L^d \to 1$ with increasing $d$. Also, the Hoshen-Kopelman method cannot be used to calculate $d_{\min}$. On the other hand, the Hoshen-Kopelman method is better suited to other problems, such as calculating the number of clusters that span across a rectangular system, than our method, based on the Leath algorithm.

## ACKNOWLEDGMENTS

We thank Y. Ashkenazy, S. V. Buldyrev, P. Grassberger, S. Havlin, and D. Stauffer for helpful comments and discussions, and BP Amoco and NSF for financial support.

## APPENDIX: ALTERNATIVE METHOD OF CLUSTER GROWTH

We discuss a variant of our approach in which we still store information concerning which sites are visited in a data structure and access the entries using a hash table. However, if one stores information about all visited sites, not just for those in the last shell(s), then a traditional random number generator (one which does not take the coordinates/orientation of the bond as input) can be used. Let us first consider the case where we have no need for occupied bond information (e.g., we are simply counting the number of sites in the cluster). When considering a growth site, if: (i) an adjacent site is vacant we determine whether the bond connected to that site is occupied or not; (ii) an adjacent site is not vacant we simply do not make a determination of whether the bond is occupied.

In this way we make a determination about whether a given bond is occupied no more than once.

Now consider the case in which we do have a need to know whether a bond is occupied or not (e.g., we are counting the number of bonds in the cluster or we will be determining the backbone of the cluster). In this case, when considering a growth site, if:

(i) An adjacent site is vacant, we determine whether the bond connected to that site is occupied or not.

(ii) An adjacent site is not vacant and is in a higher chemical shell, we also determine whether the bond is connected to that site is occupied or not.

(iii) An adjacent site is not vacant and is in the same chemical shell as the growth site, we make a determination about whether the bond is occupied only if the direction from the growth site to the adjacent site is positive. In this way, the determination about whether the bond is occupied is done only once. This situation arises in noncubic (e.g., triangular)

lattices and when we start cluster growth with multiple seeds.

(iv) An adjacent site is not vacant and is in a lower chemical shell than the growth site, we make no determination about whether the bond to that site is occupied; whether the bond is occupied has been determined earlier in the growth process. In fact, the bond must be unoccupied because if it were occupied we would have reached the growth site earlier directly from the adjacent site.

Thus we ensure that we determine whether a bond is occupied once and only once. If one needs to keep a record of whether a given bond is occupied (e.g., to later determine the backbone) this information can be stored in the entry in the data structure for the site with which the bonds are associated along with the coordinates of the site, etc.

This method trades off memory (we keep state for all visited sites) versus performance (we can use the faster traditional random number generators as opposed to the encryptionlike random number generator). Also, in cases where we, for some other reason, must keep state information about all the sites, we can obtain the benefit of the using a faster random number generator.


[1] D. Stauffer and A. Aharony, *Introduction to Percolation Theory*, 2nd ed. (Taylor and Francis, London, 1994).

[2] A. Bunde and S. Harlin, in *Fractals and Disordered Systems*, 2nd ed., edited by A. Bunde and S. Havlin (Springer, New York, 1996).

[3] P. Grassberger, J. Phys. A **19**, 1681 (1986).

[4] P. Grassberger, J. Phys. A **25**, 5475 (1992).

[5] P. Grassberger, J. Phys. A **25**, 5867 (1992).

[6] R. M. Ziff, P. T. Cummings, and G. Stell, J. Phys. A **17**, 3009 (1984).

[7] C. D. Lorenz and R. M. Ziff, Phys. Rev. E **57**, 230 (1998).

[8] D. Stauffer and R. M. Ziff, Int. J. Mod. Phys. C **11**, 205 (2000).

[9] The density of a cluster of size $L$ is $(1/L)^{d-d_f}$ where $d_f$ is the fractal dimension. Since the codimension, $d-d_f$, increases with $d$, the density decreases with $d$.

[10] P. L. Leath, Phys. Rev. B **14**, 5046 (1976).

[11] We will use bond percolation in our discussion here. Application to site percolation is straightforward.

[12] H. Vollmayr, J. Stat. Phys. **74**, 919 (1994).

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. (Cambridge University Press, Cambridge, 1992), p. 300.

[14] R. M. Ziff, Comput. Phys. **12**, 385 (1998).

[15] D. E. Knuth, *Sorting and Searching*, The Art of Computer Programming Vol. 3 (Addison-Wesley, Reading, MA, 1973), p. 513.

[16] R. Pike and H. E. Stanley, J. Phys. A **14**, L169 (1981).

[17] H. J. Herrmann and H. E. Stanley, J. Phys. A **21**, L829 (1988).

[18] S. C. van der Marck, J. Phys. A **31**, 3449 (1998).

[19] J. Adler, Y. Meir, A. Aharony, and A. B. Harris, Phys. Rev. B **41**, 9183 (1990).

[20] H. G. Ballesteros, L. A. Fernández, V. Martín-Mayor, A. Muñoz Sudupe, G. Parisi, and J. J. Ruiz-Lorenzo, Phys. Lett. B **400**, 346 (1997).

[21] O. F. de Alcantara Bonfim, J. E. Kirkham, and A. J. McKane, J. Phys. A **13**, L247 (1980); **14**, 2391 (1981).

[22] J. L. Cardy and P. Grassberger, J. Phys. A **18**, L267 (1985).

[23] H. K. Janssen, Z. Phys. B: Condens. Matter **58**, 311 (1985).

[24] H. K. Janssen, O. Stenull, and K. Oerding, Phys. Rev. E **59**, R6239 (1999).

[25] H. K. Janssen and O. Stenull, Phys. Rev. E **61**, 4821 (2000).

[26] J. Hoshen and R. Kopelman, Phys. Rev. B **14**, 3438 (1976).