Minimal spanning trees at the percolation threshold: A numerical calculation

Sean M. Sweeney and A. Alan Middleton
Department of Physics, Syracuse University, Syracuse, New York 13244, USA
(Received 9 July 2013; published 19 September 2013)

The fractal dimension of minimal spanning trees on percolation clusters is estimated for dimensions d up to $d = 5$. A robust analysis technique is developed for correlated data, as seen in such trees. This should be a robust method suitable for analyzing a wide array of randomly generated fractal structures. The trees analyzed using these techniques are built using a combination of Prim's and Kruskal's algorithms for finding minimal spanning trees. This combination reduces memory usage and allows for simulation of larger systems than would otherwise be possible. The path length fractal dimension d_s of MSTs on critical percolation clusters is found to be compatible with the predictions of the perturbation expansion developed by T. S. Jackson and N. Read [Phys. Rev. E **81**, 021131 (2010)].

DOI: 10.1103/PhysRevE.88.032129 PACS number(s): 02.50.−r, 64.60.al

I. INTRODUCTION TO THE PROBLEM

The statistical physics and dynamics of disordered physical systems naturally leads to the study of fractal geometric objects. Physical systems with quenched disorder, i.e., those with fixed random heterogeneities, often have power-law correlations at large scales or interfaces that are fractal or self-affine. These structures can result from some global optimization problem that has connections with graph theory. For example, Dijkstra's shortest path algorithm [1,2] can be used to find the lowest energy path of a vortex line in a disordered superconductor [3,4], and these paths are self-affine. Excitations in a disordered XY model in two dimensions [5,6], domain walls in spin glasses [7], and boundaries between drainage basins [8] are examples of physical objects with fractal dimension that are found by global optimization.

A structure with fractal scaling that arises in physical contexts is the minimal spanning tree (MST). One such context is a highly disordered Ising spin glass. Newman and Stein showed that for the strongly disordered limit, the problem of finding a ground state can be directly mapped to finding an MST [9]. This mapping can be used to investigate the multiplicity of ground states in the thermodynamic limit. Minimal spanning trees have other applications such as in transportation networks connecting cities [10], telecommunications networks connecting remote computer terminals [11], efficient circuit design [12], taxonomic reconstruction of evolutionary trees [13], and pattern recognition in image analysis [14].

A minimal spanning tree is a structure that connects a set of nodes with minimum total cost. This structure is defined for a weighted graph $G = (V, E, w)$, where V is a set of vertices (or nodes), E is a set of edges that connect vertices, and w is a weight function, with each edge $e \in E$ having weight $w(e)$. A spanning tree is a loopless connected set of edges that includes all the vertices in V. The minimal spanning tree is the spanning tree T that minimizes the total weight

$$w(T) = \sum_{e \in T} w(e) \,. \tag{1}$$

This is a well-known problem in computer science and combinatorics. See Ref. [15] for a general overview of MSTs and MST-finding algorithms.

A notable fact about the MST is that the minimal tree is determined only by the numerical ordering of the weights, i.e., it is otherwise independent of their value. So there is a large invariance or universality for these structures; their geometry is independent of the disorder distributions. As long as the weights $w(e)$ are independently drawn from the same distribution (independent identically distributed or i.i.d. weights), the edges can be sorted in order of increasing weight. This ordering alone determines the tree. Two physical problems with wholly different distributions of quenched disorder have MSTs with equivalent statistics.

Recently, Jackson and Read carried out an analytical calculation for the fractal dimension d_s of paths in MSTs [16,17]. They developed a perturbation expansion for d_s for MSTs on critical percolation clusters in d dimensions, obtaining the result

$$d_s = 2 - \frac{\epsilon}{7} + O(\epsilon^2) \,, \tag{2}$$

where $\epsilon = 6 - d$ and $d = 6$ is the upper critical dimension [17]. In general, disordered systems are difficult to analyze and rarely yield quantitative analytic results. This prediction therefore provides a strong motivation for more precise computation of fractal dimensions in spanning trees, in particular, dimensions of the trees on spanning percolation clusters. We note that it is unclear whether the fractal dimension of these trees is affected by being constructed on spanning percolation clusters as opposed to a whole lattice. The work presented in this paper seeks to numerically compute d_s in dimensions $2 \leqslant d \leqslant 5$ for comparison with Eq. (2). This calculation employs a combination of memory-saving techniques to simulate large systems as well as careful data fitting procedures to obtain precise estimates for d_s in the limit of infinite system size.

Our final calculations for d_s yield values of 1.216(1) for $d = 2$, 1.46(1) for $d = 3$, 1.65(2) for $d = 4$, and 1.86(4) for $d = 5$ (refer to Table I). We develop and utilize a χ^2 test that accounts for the scale-invariant correlations found in the data, allowing for an improved χ^2 measure and robust estimates of the uncertainty in the effective exponent at scale L, $d_s(L)$. We then use linear and nonlinear least squares fitting to extrapolate to the infinite system size limit. We find the numerical results to be of higher precision than previous calculations and compatible with Eq. (2), though

TABLE I. d_s calculated using MST algorithms.

Dimension d	Two-step	Prim's (no trimming)
2	1.216(1)	1.216(1)
3	1.46(1)	1.46(1)
4	1.65(2)	1.66(2)
5	1.86(4)	1.86(4)

more conclusive confirmation requires improved numerical statistics and higher order analytic work. We emphasize that the analysis procedure used here is generalizable and could be useful for other work dealing with disordered systems.

II. MODEL AND ALGORITHMS

To model MSTs on percolation clusters, we simulated hypercubic lattices with L vertices per side with periodic boundary conditions, giving L^d total vertices and dL^d total edges. Edges are independently given a weight randomly distributed on the interval (0,1), where $w(e)$ is represented by a double precision number. Very rarely two edges are assigned the same weight. In this case, a new random weight is generated for one of these edges until a weight is generated that does not match any previously assigned weight. An important distinction to note is that rather than seeking to find a tree that spans all of the vertices in these hypercubic lattices, we seek to find the MST on a percolation cluster that wraps around the periodic lattice (in any of the d directions) at the threshold of percolation. Thus the MST-finding algorithms are stopped when the tree contains a subset of vertices that wraps around the system instead of including all L^d vertices of the graph. The final object of interest, the MST on a critical percolation cluster, contains only a small subset of the set of vertices V in the lattice.

In order to avoid confusion, we first note a dual usage of the term spanning. For an MST, spanning means that the tree includes all vertices in the graph for which the MST is found. In the context of percolation theory, the term refers to a cluster that is spanning or percolating around the lattice. We use spanning in both senses, with the correct sense implied by the context.

One naive approach to finding the MST on a graph is to iterate through the list of all spanning trees and select the tree with the lowest total weight. This approach might work on a small finite graph, but the number of trees grows exponentially with L^d. In order to analyze properties of MSTs in the thermodynamic limit of infinite system size, a more efficient MST-finding algorithm (both in terms of running time and maximum memory requirements) is needed.

As background for the algorithms used to find MSTs, it might be helpful to briefly review the relationship between invasion percolation and Bernoulli percolation. In Bernoulli (bond) percolation [18,19], edges in a random graph have a probability p to be occupied, and a probability $1-p$ to be unoccupied. After determining the occupation of each edge independently, one inspects the graph to check for long-range connectivity in the form of a cluster of connected vertices that percolates, i.e., spans the graph. On a periodic hypercubic lattice, one definition of a percolating or spanning cluster is a cluster that wraps around the lattice along one or more of the d spatial dimension axes. Such a cluster contains a loop that cannot be contracted to a point. Note that Bernoulli percolation is equivalent to assigning weights $w(e)$ on the interval (0,1) and occupying only those edges with weights $w(e) \leqslant p$. We will refer to this as the alternative definition of Bernoulli percolation.

Examining larger and larger systems on a macroscopic scale, this percolation transition becomes clear; below some critical percolation probability p_c only small clusters are seen, but at $p = p_c$ large clusters that span the system begin to emerge. Aizenman refers to these large critical clusters as incipient spanning clusters (ISCs), a term which is closely related to, but distinct from, the incipient infinite cluster (IIC) [20,21]. In the exploration of Bernoulli percolation, this occupation probability p is finely tuned in order to observe this critical transition and the clusters that are formed at criticality.

An alternate approach to percolation is invasion percolation [22]. Invasion percolation is a procedure that greedily occupies edges of low weight w and has a termination condition. Invasion percolation consists of a growing cluster C that is a subset of E. This cluster has a set of adjacent edges ∂C. The cluster C begins as a single occupied seed site. Additional sites are "invaded" by choosing from ∂C the lowest weight edge, $e_l(\partial C)$, and expanding the invaded region to include this edge, by adding $e_l(\partial C)$ to C. This invasion percolation process can be repeated until long-range connectivity is observed (i.e., until the invaded region percolates across the system).

There has been much discussion [23–25] on how to relate the clusters of invasion percolation to those of ordinary Bernoulli percolation. There seems to be a strong reason to believe that in the limit of infinite system size, infinite connected clusters created using both of these percolation methods should obey the same scaling relations, meaning that the fractal path length dimension d_s should be the same for both methods. Invasion percolation is extremely useful because it allows for the simulation of critical percolation systems without having any knowledge of what the value of p_c is for a particular system. Thus invasion percolation is an example of self-organized criticality [23,24].

In our particular case, we're interested in analyzing the MSTs that lie on an ISC. In other words, we want to find the MST for the ISC. We'll refer to this final object as the MTISC (minimal tree for an incipient spanning cluster). Here we use two algorithms, Kruskal's algorithm [26] and Prim's algorithm [10]. While Prim's algorithm is similar to the invasion percolation process described above [27], Kruskal's algorithm is related to Bernoulli percolation due to the global nature of the algorithm. However, neither algorithm actually requires the choice of an occupation probability p as a parameter, and consequently both processes exhibit self-organized critical behavior, as the growth is stopped when a tree is found that wraps [28]. For a more precise definition of these algorithms, refer to Appendix A. We next present less formal descriptions of these algorithms.

A. Kruskal's algorithm

Kruskal's algorithm is an MST-finding algorithm that considers all edges of a graph. In Kruskal's algorithm, we grow a forest of many small trees, merging small groups of trees into larger trees and eventually identifying one of these

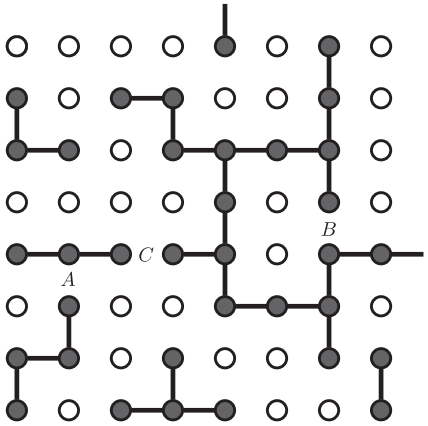

FIG. 1. A sample iteration of Kruskal's algorithm on an eight by eight periodic square lattice ($d = 2$). At a given step, the state is a forest of trees, which includes isolated sites (open circles) and larger trees (connected solid circles). During each step of the algorithm, the edge with the lowest weight is selected from the remaining unselected edges. For example, if edge A is selected, the trees containing either endpoint of edge A are merged into a single tree. If edge B is selected, its addition would form a nonwrapping loop (forbidden cycle), so edge B is not added to any tree and is removed from future consideration. If edge C is selected, its addition would form a wrapping loop (allowed cycle), so the algorithm is terminated. The tree containing the endpoints of edge C is then the Kruskal's MTISC, T_K.

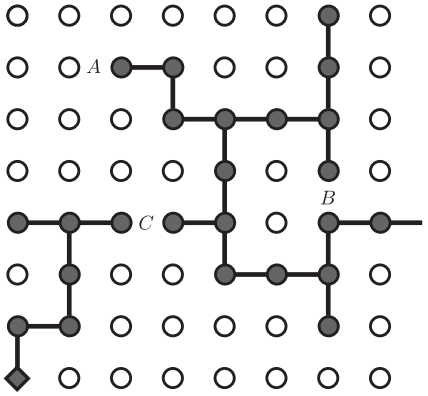

FIG. 2. A sample iteration of Prim's algorithm on an eight by eight periodic square lattice ($d = 2$). The tree is represented by connected solid circles and lines, while unoccupied sites are shown as open circles. The diamond-shaped vertex at the bottom left of the lattice is the initial seed site v_0. During each step of the algorithm, the edge adjacent to the current Prim's tree with the lowest weight is selected. If edge A is selected, it is added to the Prim's tree, along with the endpoint of A that is not already in the Prim's tree. If edge B is selected, its addition to the Prim's tree would form a nonwrapping loop (forbidden cycle), so edge B is not added to the tree and is removed from future consideration. If edge C has the lowest weight and is selected, its addition would cause the Prim's tree to wrap around the periodic lattice (adding an allowed cycle), so the algorithm is terminated, leaving the current Prim's tree as the Prim's MTISC, T_P.

large trees as the MTISC, terminating the algorithm. At the start of Kruskal's algorithm, each vertex is its own isolated tree. All of the edges that are not yet part of a tree are sorted, and the edge with minimal weight is selected, excluding edges that would form nonwrapping loops. When an edge is selected, the trees containing each of its vertices are joined into a single tree. This process continues until a tree grows big enough that it wraps around the periodic lattice. At this time, this tree is identified as the Kruskal's MTISC, T_K. This process is illustrated in Fig. 1.

Kruskal's algorithm bears similarity to Bernoulli percolation in that it is a nonlocal algorithm that requires information about all edges of the graph. If we consider a graph with edges whose weights are distributed evenly on $(0,1)$ and run Kruskal's algorithm until we first examine an edge with weight $w > p_c$, the sites of T_K are identical to those obtained from performing Bernoulli percolation on the same graph with occupation probability p_c. Thus T_K is the MST on the Bernoulli percolation cluster. Given the Bernoulli percolation cluster, this MST is that formed by using the weights $w(e)$ in the alternative definition of Bernoulli percolation.

B. Prim's algorithm

Prim's algorithm is equivalent to algorithms for loopless invasion percolation [27]. At the start of the algorithm, the MST consists of a single seed site. This tree grows outward from this vertex through the examination and conditional addition of adjacent edges, as edges adjacent to the growing tree are examined. At any given iteration, the minimal weight adjacent edge is selected and incorporated into the tree, excluding edges that would lead to nonwrapping loops. The check for which edges form a loop is simple: if both ends of the edge are in the current tree, a loop would be formed. To check whether a loop is wrapping or nonwrapping, the algorithm assigns to each vertex a a displacement $\vec{r}_a$ from the origin. This displacement is found for a newly added vertex b by adding a vector displacement $\vec{e}_{ab}$ for the edge to the vector displacement of the end of the edge (site) that is already in the tree $\vec{r}_a$. Thus for a newly added site b we have $\vec{r}_b = \vec{r}_a + \vec{e}_{ab}$. When a loop is encountered, this new site b will already have a vector displacement $\vec{r}'_b$ defined, since this site is already in the tree. If the relative displacement $|\vec{r}_b - \vec{r}'_b|$ between these two labelings is greater than L, the loop formed is a wrapping loop. When this wrapping edge is examined, the algorithm is terminated, and this final tree is identified as the Prim's MTISC, T_P, as shown in Fig. 2. T_P is thus the MST on the invasion percolation cluster, given an initial seed site. Note that this MST or invasion cluster may depend on the seed site.

To increase efficiency when generating MTISCs, Kruskal's, and Prim's algorithms can be engineered to save memory by creating edges and weights only as needed during the execution of the growth algorithm rather than at the start [29,30]. The memory saved in the Prim's method is much larger and leads to an overall more memory efficient method because Prim's algorithm only requires the growing of a single tree rather than a large forest of trees. Due to this decreased memory usage, it is possible to simulate larger systems with Prim's algorithm than with Kruskal's algorithm.

C. Two-step method

We used a two-step method, combining Prim's and Kruskal's algorithms in order to take advantage of the in-

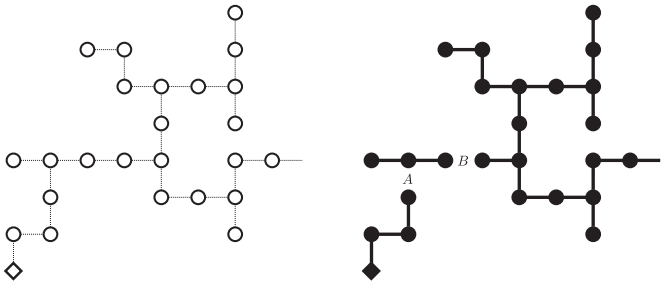

FIG. 3. Illustration of the two-step method. This method grows a candidate MTISC using the Prim's method and then trims the tree using Kruskal's algorithm. (a) depicts T_P (plus the final wrapping edge) which is given as input to Kruskal's algorithm. During construction of this tree, only edges adjacent to the tree are tested. The potential connectivity of the unoccupied sites is explicitly shown using open circles and thin lines, and for reference the Prim's seed site v_0 at the bottom left of the lattice is represented by a diamond. (b) shows the state of the graph after running 23 steps of Kruskal's algorithm on T_P, with trees being represented by connected solid circles. If edge B is selected before edge A, i.e., $w(B) < w(A)$, the Prim's seed site v_0 will not be part of T_K, and the disconnected portion of the graph containing the Prim's seed site v_0 will be trimmed off.

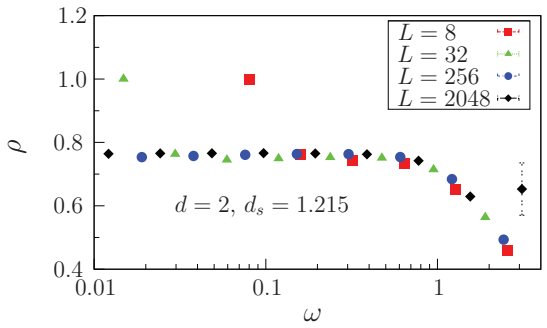

FIG. 4. (Color online) Scaling collapse for $d = 2$ with $d_s = 1.215$, which appears to be moderately close to the true value of d_s judging by eye from the goodness of the collapse. Note that error bars are smaller than the symbol size for all points except the right-most point.

creased efficiency afforded by Prim's algorithm when selecting a typical MTISC. We first generate a tree T_P using Prim's algorithm and then apply Kruskal's algorithm to this tree, ensuring that the MTISC obtained has the same scaling as the MTISC that would be obtained by increasing p large enough to form a forest, one of whose trees is the final tree T_K. The data presented in this paper comes from analysis of MTISCs generated by the two-step method, as well as comparisons with the intermediate data from T_P, before Kruskal's algorithm is applied to T_P.

While Prim's algorithm in general takes less time and memory to find an MTISC than Kruskal's algorithm, the two algorithms do not find exactly the same MTISC [31]. T_K and T_P are not completely equivalent because T_K is an MST on the Bernoulli percolation cluster, while T_P is an MST on an invasion percolation cluster. As shown in detail in Appendix B, either T_K is a subset of T_P, or they do not intersect. We are interested in constructing an MTISC that is either identical to T_K or scales the same as T_K. The nonlocal greedy edge selection of Kruskal's algorithm guarantees T_K to have the same sites as a Bernoulli percolation cluster.

A technical detail to note about the two-step method is that in the final stage of Prim's algorithm when a wrapping edge is selected, we add this edge to the final Prim's tree T_P. This is done so that when Kruskal's algorithm is applied to T_P, there will be a wrapping edge to satisfy the termination condition of Kruskal's algorithm. While the inclusion of this edge temporarily destroys the tree-like nature of T_P, its inclusion is essential.

Because Prim's growth begins at a random seed site that does not necessarily belong to T_K, often the T_K is a subset of T_P. This of course depends which vertex v_0 is used as the seed site for Prim's growth, as the Prim's MTISC is a function of its seed site, $T_P(v_0)$. Executing Kruskal's algorithm in the second step of this method serves to "trim off" the part of $T_P(v_0)$ that includes the seed site v_0 and that does not belong to T_K. The tree derived from this procedure is termed the two-step MTISC, T_2. This method is illustrated in Fig. 3.

We show in Appendix B that this two-step procedure yields the Kruskal's MTISC, i.e., $T_2 = T_K$, in all cases except when there naturally arise multiple disjoint ISCs on the lattice at criticality. By direct simulation of these systems, we observed that $T_2 \neq T_K$ about 0.2% of the time in two dimensions, 1% in three dimensions, and 5% in four dimensions. Five dimensional samples were not compared due to the large memory demands of simulating minimal spanning forests in five dimensions.

We suppose by standard scaling that in the case where $T_2 \neq T_K$, T_2 and T_K should have the same path length fractal dimension d_s. We simulated samples in this case for systems up to size 512^2, 64^3, and 32^4. Examining scaling collapses as in Fig. 4, we observed similar values of d_s for T_2 and T_K in two, three, and four dimensions. Data for T_K in five dimensions was not obtained due to the large memory demands of simulating minimal spanning forests in five dimensions. The memory requirements for the two-step method allowed us to simulate systems of size 2048^2 within roughly 2 GB of memory, 256^3 within 1.5 GB, 64^4 within 1 GB, and 48^5 within 2 GB.

III. ANALYSIS/METHODS

A. Methods for scaling analysis

To find the fractal dimension of the minimal trees on spanning percolation clusters, we compute the Euclidean distance r and path length s between some origin on the MTISC and other points on the MTISC. Accurately determining the scaling in the limit of large clusters requires taking into account lattice effects, finite size effects, and statistical uncertainties. Given any two points on a tree, there is a unique path between the two points, so that the path length is easily defined. The choice of endpoints for the paths is chosen in a natural fashion for each tree.

For trees constructed using Prim's algorithm, the origin is taken to be the seed site where cluster growth begins. The set of paths connecting the origin to all other points in the tree

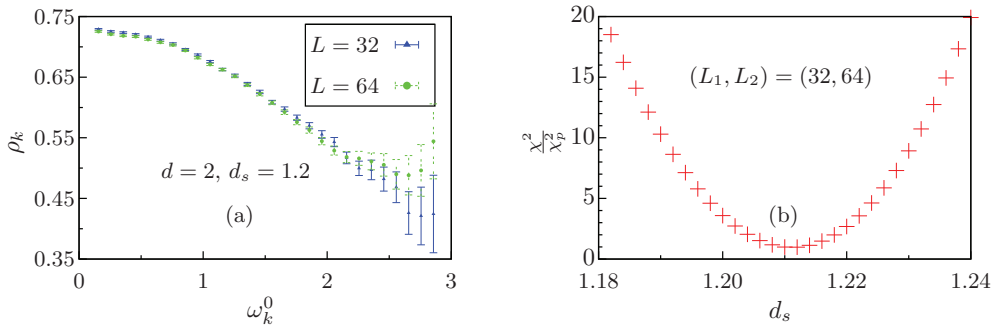

FIG. 5. (Color online) A sample collapse for two systems of size $L = 32$ and $L = 64$ in dimension $d = 2$. (a) Shows the comparison of the two systems at interpolated points ω_k^0 for a value of $d_s = 1.2$. Comparing the value of $\rho_k(L_1 = 32)$ indicated by the blue (dark gray) triangles and $\rho_k(L_2 = 64)$ indicated by the green (light gray) circles at each of these points allows for the calculation of $\chi^2(d_s; L_1, L_2)$. (b) Shows χ^2 compared with an expected estimate χ_p^2 as a function of the fitting parameter d_s for the same pair of systems. Though we determine final error bars by resampling, the χ^2 model is shown for comparison.

is used in the statistics. For each tree T_P, we find $n_P(s)$, the number of paths of length s that start at the origin. We use $r_P(s)$ to indicate the average over all paths of length s of the Euclidean distance $|\vec{r}|$.

After then running Kruskal's algorithm on the Prim's tree to find the trimmed MTISC T_2, a random origin is chosen on the trimmed tree. All paths from this new origin are used to find both $r(s)$ and $n(s)$, the averaged Euclidean distance and number of paths. In order to reduce the amount of data stored, the full set of N samples is grouped into sets of N_b batches of uniform size N/N_b. Thebatch-averaged quantities of $r(s)$ and $n(s)$ for these trimmed trees are calculated and stored. Here we use $r(s)$ and $n(s)$ to refer to data generated by the two-step algorithm. As a comparison, we also looked at the averages for Prim's trees, before trimming.

We assume that the paths on the tree are well described as fractal. The scaling of the sample-averaged $r(s)$ will then follow the relation

$$r(s) \sim s^{1/d_s} . \tag{3}$$

If we write L as the length of one side of a hypercubic system, then s/L^{d_s} is a natural scaling parameter, and we expect to see finite size effects near $s/L^{d_s} = 1$, as paths of this length approach the size of the system. The standard one parameter finite size scaling assumption is then that $r(s)$ will scale like s^{1/d_s} multiplied by some unknown function of the argument $s^{1/d_s}L^{-1}$. The form of the scaling hypothesis is that

$$r(s) \approx s^{1/d_s} f\left(\frac{s^{1/d_s}}{L}\right) \approx s^{1/d_s} g\left(\frac{s}{L^{d_s}}\right), \tag{4}$$

where the scaling functions $f(\omega)$ and $g(\omega)$ behave as $f(\omega) \approx c_1$ for some constant c_1 for $\omega \ll 1$, $f(\omega) \approx 0$ as $\omega \to \infty$, $g(\omega) \approx c_2$ for some constant c_2 for $\omega \ll 1$, $g(\omega) \approx 0$ as $\omega \to \infty$. For more compact formulas, we define the scaled dimensionless variables

$$\rho = \frac{r(s)}{s^{1/d_s}} , \tag{5}$$

$$\omega = \frac{s}{L^{d_s}} . \tag{6}$$

Using this scaling hypothesis, we can make estimates for d_s by plotting data for multiple system sizes on the scaled axes of ρ vs. ω. If we tune the parameter d_s, we see that the curves for various sizes L_i can be made to collapse well near an estimated best value of d_s (see Fig. 4). While this method allows us to get a fair idea of this exponent d_s, it relies on subjective estimations of curve collapse and for that reason is not ideal when attempting precise estimates.

To have a better estimate for d_s and to estimate the uncertainty in this estimate, we have implemented an automated fitting procedure. This procedure determines an effective exponent $d_s(L)$ derived from data for samples of size L and of larger size $2L$ by minimizing a "goodness of fit" parameter χ^2 at each scale L. The only input to this procedure is an estimate of s_l, the small path data cutoff, which is discussed in Appendix C. We then extrapolate $d_s(L)$ for $L \to \infty$ to get our best estimate for d_s.

The key part of this calculation is to choose a robust and reliable measure for χ^2. This allows us to quantitatively measure how well the data for a given pair of system sizes collapses as a function of the parameter d_s. We seek a value of d_s for which this χ^2 is minimized (Fig. 5), though the magnitude of our final error bars are determined by resampling. Our definition of χ^2 must allow for nonuniform correlations in fluctuations of $r(s)$ between different values of s, discrete lattice effects (small s lower data cutoff), and statistical uncertainties (large s upper data cutoff).

B. Correlations in $r(s)$

In order to define a useful χ^2 statistic, we first focus on the correlations we observed in the $r(s)$ data for the spanning trees. In summary, we find that these correlations have a range in s that grows linearly with increasing path length s, in dimensions $d = 2,3,4,5$. We then modify the standard χ^2 test for uncorrelated data to account for these correlations.

To describe correlations in the averaged $r(s)$ data, it will be helpful to first define how our data is averaged over samples, since we use multiple groupings of data for calculating averages. Let $r_i(s)$ denote an average of Euclidean distance $r = |\vec{r}|$ over all points on tree i that are at a chemical (path length) distance s from the origin for each tree $i = 1, \ldots, N$ in the N samples. For faster analysis, the N samples are organized into N_b batches. The batch index α ranges over $1 \leqslant \alpha \leqslant N_b$.

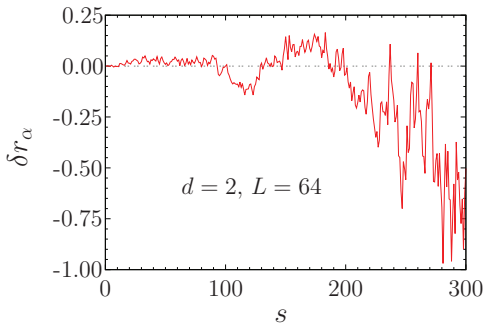

FIG. 6. (Color online) Variations from the mean for a typical batch α of data for a system of size $L = 64$ in dimension $d = 2$. The difference between the mean Euclidean distances $\delta r_\alpha(s) = r_\alpha(s) - \overline{r(s)}$ is plotted vs. path length s.

We will use $r_\alpha(s)$ to denote an average over the $m = N/N_b$ samples in batch α:

$$r_\alpha(s) = \frac{1}{m}\sum_{i\in\alpha} r_i(s)\,. \tag{7}$$

The global average over all N samples will be represented by

$$\overline{r(s)} = \frac{1}{N_b}\sum_{\alpha=1}^{N_b} r_\alpha(s) = \frac{1}{N}\sum_{i=1}^{N} r_i(s)\,. \tag{8}$$

To initially examine correlations over s of $r_i(s)$ within a single tree, we plot the fluctuations of the batch averages $r_\alpha(s)$ about the global average $\overline{r(s)}$. That is, we plot the variations of the average $\delta r_\alpha(s)$, where

$$\delta r_\alpha(s) = r_\alpha(s) - \overline{r(s)}\,, \tag{9}$$

vs. path length s. Fig. 6 displays these correlations for a typical batch of data in a system of size 64^2.

Note that the form of the correlations should be independent of batch size, up to a multiplicative scaling factor. We can see this explicitly by examining $\overline{\delta r_\alpha(s)\delta r_\alpha(t)}$ for path length values s and t. For $i \neq j$, i and j are independent samples, so we can use the relation

$$\overline{\delta r_i(s)\delta r_j(t)} = \overline{\delta r_i(s)}\ \ \overline{\delta r_j(t)} \tag{10}$$

and of course

$$\overline{\delta r_i(s)} = \overline{r_i(s) - \overline{r(s)}} = 0\,. \tag{11}$$

By standard computation all of the $i \neq j$ cross terms are zero, and we are left with

$$\overline{\delta r_\alpha(s)\delta r_\alpha(t)} = \frac{1}{m}\overline{\delta r_i(s)\delta r_i(t)}\,, \tag{12}$$

showing that the choice of grouping the data into batches should not affect the form of the correlations in $r(s)$.

To quantitatively examine these correlations we compute the correlation matrix $c_{s,t}$ defined as

$$c_{s,t} = \frac{1}{N_b}\sum_{\alpha=1}^{N_b}\frac{\delta r_\alpha(s)\delta r_\alpha(t)}{\gamma(s)\gamma(t)}\,. \tag{13}$$

Here $\gamma(s)$ is the root mean square fluctuation in $\delta r_\alpha(s)$ computed over the N_b batches of data and is used to normalize

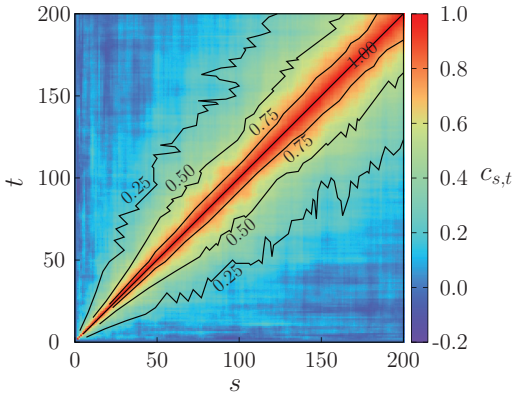

FIG. 7. (Color online) Correlation matrix $c_{s,t}$ averaged over all data for systems of size $L = 64$ in dimension $d = 2$. Selected contours are shown.

the entries $c_{s,t}$ of the correlation matrix:

$$\gamma^2(s) = \frac{1}{N_b}\sum_{\alpha=1}^{N_b}\delta r_\alpha^2(s)\,. \tag{14}$$

A sample correlation matrix for $L = 64$ is plotted in Fig. 7. The data suggest that the correlation length increases with increasing s. To construct a model for the scaled correlation length, we measured the width along the diagonal of the peak in the correlation matrix for various values of s. We used different measures for measuring this width, including a measure of the full width at half maximum (how many "steps" away from the diagonal before $c_{s,t}$ falls to a value of $1/2$), a measure of one decay length (how many steps from the diagonal until $c_{s,t}$ drops to a value of $1/e$), and a measure using an exponential decay model $c_{s,t} = \exp(-|s-t|/\ell')$ assuming an unscaled correlation length ℓ'. We calculate ℓ' using the zeroth moment (integral) of $c_{s,t}$, allowing us to then obtain the scaled correlation length $\ell = \ell'/L^{d_s}$, given a rough estimate for d_s.

We plot in Fig. 8 the scaled correlation length ℓ vs. ω for multiple system sizes for dimension $d = 2$, using the zeroth moment of $c_{s,t}$ to estimate the unscaled correlation length

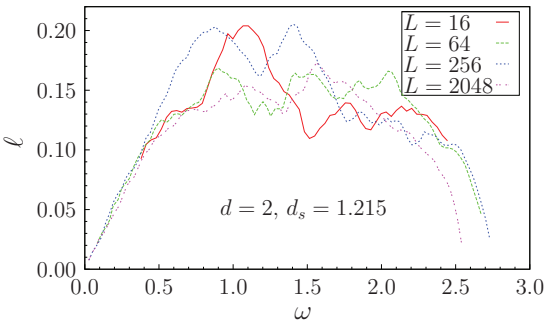

FIG. 8. (Color online) Scaled correlation length ℓ vs. ω in dimension $d = 2$ with $d_s = 1.215$ for systems $L = 16,64,256,2048$ (red solid, green dashed, blue dotted, and magenta dash-dotted lines, respectively). In this case ℓ was measured assuming exponential decay $c_{s,t} = \exp(-|s-t|/\ell')$ and integrating $c_{s,t}$ to obtain the unscaled correlation length ℓ', with the relation to the scaled correlation length ℓ being $\ell = \ell'/L^{d_s}$.

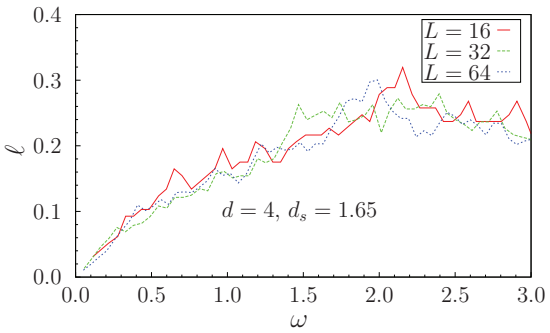

FIG. 9. (Color online) Scaled correlation length ℓ vs. ω in dimension $d = 4$ with $d_s = 1.65$ for systems $L = 16,32,64$ (red solid, green dashed, and blue dotted lines, respectively). In this case the unscaled correlation length ℓ' was measured using the full width at half maximum (how many "steps" in s away from the diagonal before $c_{s,t}$ falls to a value of $1/2$). To obtain the scaled correlation length ℓ, we use the relation $\ell = \ell'/L^{d_s}$.

ℓ'. We see that ℓ increases linearly with ω up until roughly $\omega = 1$, at which point the scaled correlation length levels off to a constant value. An alternate example of a measure of correlation length ℓ' for $d = 4$ is displayed in Fig. 9. Here we used the full width at half maximum to measure correlation length. There are larger fluctuations in the curve. Although the proportionality constant differs depending on which method is used to measure the correlation length, we observed a relationship between ℓ and ω consistent with linear behavior for ω up to about 1 in all cases for dimensions $2 \leqslant d \leqslant 5$. The crossover generally appears broader when measuring correlation length with the full width at half maximum than when measuring using the integral of $c_{s,t}$. The broadness of this crossover does not seem to depend on dimensionality.

Based on this empirical observation, as well as recent results [32] showing that invasion percolation has strong mixing over geometrically separated scales, we choose an ansatz for the scaled correlation length:

$$\ell(\omega) \propto \begin{cases} \omega : \omega \leqslant 1 \\ 1 : \omega > 1. \end{cases} \tag{15}$$

The $\omega = 1$ cutoff for the linear regime in this model is a rough estimate based on the empirical data as seen in Figs. 8 and 9, and we find this model to work well with our data. Furthermore, this cutoff is unimportant to some extent since high ω (large path length) data points are weighted less overall, as long paths are less frequent. For the exponential decay model used in Fig. 8, the proportionality is $\ell \approx 0.24\omega$. Physically, this means that on average, for any given growing path in a spanning tree, the path must grow about 24% longer than its current length before the correlations in the $r(s)$ data for this path decay, using the exponential decay model $c_{s,t} = \exp(-|s-t|/\ell')$ for the unscaled correlation length ℓ'.

Now that we have a consistent model for the correlation length, we incorporate this model into a χ^2 goodness of fit measure. We use subscripts 1 and 2 to indicate data sets for systems of size L_1 and L_2 being compared. We use $L_2 = 2L_1$. A general goodness of fit measure for quantifying how well two curves collapse in variables ρ vs. ω starts with choosing a set of independent points ω_k and at each point calculating the difference between $\rho_1(\omega_k)$ and $\rho_2(\omega_k)$, $\Delta_{12}(\omega_k) = \rho_1(\omega_k) - \rho_2(\omega_k)$.

For a χ^2 test with uncorrelated data this difference $\Delta_{12}(\omega_k^0)$ at the point ω_k^0 would then be squared and normalized by the sum of the variances, $\sigma_1^2(\omega_k^0)$ for the system of size L_1 and $\sigma_2^2(\omega_k^0)$ for the system of size L_2. This gives

$$\chi_0^2(d_s; L_1, L_2) = \sum_k \frac{\Delta_{12}^2(\omega_k^0)}{\sigma_1^2(\omega_k^0) + \sigma_2^2(\omega_k^0)}\,, \tag{16}$$

where we would sum over discrete points ω_k^0 chosen with uniform spacing [33].

To incorporate the observed structure of the correlations into our definition of χ^2, we compare the spacing of data points with the correlation length. First consider grouping the data points into bins with a representative point ω_k at the center of each bin, with the ω_k points spaced out in such a way that they are effectively independent. To this end, we consider centering bins around the terms of the sequence $\omega, q\omega, q^2\omega, \ldots, 1$, where q is some constant factor. Our assumption is that data points in bins centered at ω and $q\omega$ are correlated to the same extent as data points in bins centered at $q^k\omega$ and $q^{k+1}\omega$. So the correlation length is effectively constant if we choose bins logarithmically spaced, i.e., $\omega_{k+1} = q\omega_k$, for $\omega_k \leqslant 1$ and for some constant factor q. Furthermore, changing the value of q will change the value of this correlation length, and there will be some choice of q for which this correlation length is equal to 1, allowing us to use the form of Eq. (16):

$$\chi^2 = \sum_k \frac{\Delta_{12}^2(\omega_k)}{\sigma_1^2(\omega_k) + \sigma_2^2(\omega_k)}\,. \tag{17}$$

If we consider taking the continuum limit of this equation, we see that

$$\begin{aligned} \chi^2 &= \int \frac{\Delta_{12}^2}{\sigma_1^2 + \sigma_2^2}\, d(\log_q \omega) = \int \frac{\Delta_{12}^2}{\sigma_1^2 + \sigma_2^2} \left(\frac{1}{\ln(q)\omega}\right) d\omega \\ &\propto \int \frac{\Delta_{12}^2}{\ell_q(\sigma_1^2 + \sigma_2^2)}\, d\omega\,, \end{aligned} \tag{18}$$

where we have defined $\ell_q = \ln(q)\omega$. If we revert back to the discrete form, we see that our generalized goodness of fit measure becomes

$$\chi^2 = \sum_k \frac{\Delta_{12}^2(\omega_k^0)}{\ell(\omega_k^0)\left[\sigma_1^2(\omega_k^0) + \sigma_2^2(\omega_k^0)\right]}, \tag{19}$$

where $\ell(\omega_k^0) \propto \ell_q$ is the scaled correlation length at point ω_k^0 [34]. Dividing by the scaled correlation length ℓ will effectively weight each "box" or bin of data points (whose width is equal to ℓ) as one independent data point in the χ^2 sum. We use the form of ℓ given by Eq. (15). The proportionality constant in Eq. (15) affects the scale of χ^2 but not the fitted d_s. See Appendix C for details.

For our analysis, all runs ($N = 4 \times 10^6$ samples per system size) were split into $N_b = 400$ uniform batches (histograms) of $N/N_b = 10^4$ samples apiece. This was done in part because storing data for all 4×10^6 samples was impractical and running the bootstrap analysis over samples individually would be too time consuming to be feasible. To determine a useful batch size, we ran some preliminary tests by varying the

batch size and plotting scaling collapses like Fig. 4. We saw similar estimated values of d_s based on these collapses, independent of batch size. A batch size of 400 seemed balanced because it allowed for error bars of order $1/\sqrt{N_b} \approx 5\%$. As we have shown in Eq. (12), the form of correlations should be independent of batch size chosen.

C. Extrapolation to $L = \infty$

Next we must consider exactly what region in s of the data we want to fit. We address lattice effects by examining a lower (small s) data cutoff. The upper (large s) cutoff is also considered due to low statistics (large uncertainties) for $s \gg L^{d_s}$, though in the end we find that no upper cutoff is necessary. To determine reasonable cutoffs, we examine how the measured χ^2 and d_s respond to changes in these cutoffs. A more detailed discussion is included in Appendix C. Once we've decided upon fair cutoffs, this collapse procedure is run N_b times (once for each batch of data for each pair of sizes L_1, L_2 with $L_2 = 2L_1$). Each time, the value of d_s for which χ^2 is minimized is found, giving N_b independent estimates $d_s(\alpha, L_1, L_2)$ for a given pair of systems L_1 and L_2. The final estimate of d_s for this pair of systems is the average of the N_b independent estimates:

$$\overline{d_s(L_1,L_2)} = \frac{1}{N_b}\sum_{\alpha=1}^{N_b} d_s(\alpha,L_1,L_2)\,. \tag{20}$$

Then the sample variance $S^2(L_1,L_2)$ in these N_b estimates is used to estimate the statistical error bars in the final estimate,

$$\sigma_{d_s}(L_1,L_2) = \frac{S(L_1,L_2)}{\sqrt{N_b - 1}}\,. \tag{21}$$

At this point we have obtained a single estimate of d_s (with error bars) for each consecutive pair of systems simulated.

Next we look to see if this $\overline{d_s(L_1,L_2)}$ converges to some value as L_1 and L_2 tend to infinity. We extrapolate the infinite system size limit using a variety of least squares fitting routines adapted from the GNU Scientific Library [35]. Standard scaling suggests that d_s as a function of system size L will exhibit power law scaling behavior. But since we have no knowledge of the expected value for the exponent in this power law, we can allow this exponent to vary as a parameter in our fit, plotting $\overline{d_s}$ vs. $L^{-\lambda}$, where we take L to be the geometric mean $L = \sqrt{L_1 L_2}$. Allowing λ to vary, fitting the data gives an estimate for d_s as well as the correction to scaling exponent λ. See Fig. 10 for an example of one extrapolation attempt, where λ is seen to be roughly 0.5 and linear least squares fitting for $\overline{d_s}$ vs. $L^{-\lambda}$ is used for the four largest system sizes in $d = 2$ dimensions.

D. Blind test for analysis method

To test this data analysis procedure, we applied the procedure to the similar problem of the uniform spanning tree (UST) in dimension $d = 2$. Whereas a minimal spanning tree seeks to minimize the total weight of a tree that spans all the vertices of the lattice, a uniform spanning tree is simply any tree that spans the vertices of the lattice, chosen with equal weight from all possible spanning trees. It can be thought of as a generalization of the minimal spanning tree problem to a system where all edges have the exact same weight. The reason for using this system as a test case for the analysis method is that one can examine $r(s)$ data to look at d_s for paths on the UST, just as one would examine such paths on an MST. The analysis should be completely analogous, and we observed directly that the correlations in the $r(s)$ data for the UST have a similar structure to that of the MST, allowing us to use the form of Eq. (15) for ℓ. To ensure no bias in this test, one of us was not made aware of the nature of the system at the time of this test but was only given the prepared $r(s)$ data on which to blindly run the analysis. The final result from this blind test, using a range of systems of size 128^2 to 1024^2, was $d_s = 1.2499(4)$ for the UST, which agrees well with the known exact result $d_s = 5/4$ in two dimensions [36,37]. This provides confidence in the data analysis method.

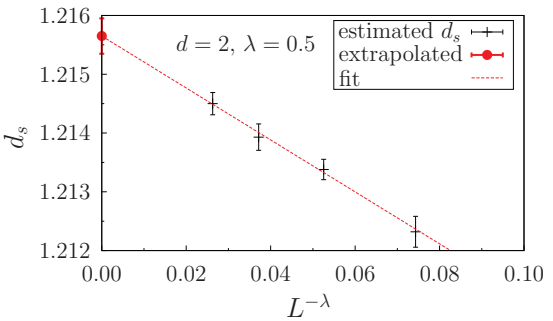

FIG. 10. (Color online) An illustration of a linear least squares fitting method being used to extract the value of d_s in the infinite system size limit in dimension $d = 2$. The fit uses the form $d_s(L) = AL^{-\lambda} + d_s(\infty)$ where λ is a correction to scaling exponent and A and $d_s(\infty)$ are fitting parameters. Here $\lambda = 0.5$, and $L = \sqrt{L_1 L_2}$, where L_1 and L_2 are the two system sizes used to produce a given data point. The fit found gives $d_s(\infty) = 1.216(1)$.

IV. RESULTS

Table I displays our final numerical estimates for d_s. The values in dimensions $d = 2$ and $d = 3$ agree with previous results [38–41] and have error bars that are similar or smaller. Our result for d_s in dimension $d = 4$ is a bit higher than the result 1.59(2) by Cieplak, Maritan, and Banavar [38]. The fractal dimensions computed from the two-step MTISC agree with those of the intermediate, untrimmed Prim's MTISC, to within our error estimates.

Our results for the path length dimension d_s can be directly compared with the $O(\epsilon = 6 - d)$ expansion of Jackson and Read. A graphical comparison of the data is shown in Fig. 11. Our results are not in conflict with the first order perturbation theory calculations. Some previous comparisons of $O(\epsilon)$ calculations with numerical results for disordered materials show differences of similar magnitude [42].

We investigate possible $O(\epsilon^2)$ calculations using nonlinear fitting routines adapted from the GNU Scientific Library [35] to fit d_s vs. ϵ. Using a two parameter fit,

$$d_s = 2 + a\epsilon + b\epsilon^2\,, \tag{22}$$

and allowing a,b to vary, we find a chi-squared of 3.8 for two degrees of freedom (d.o.f.), suggesting consistency with a quadratic fit to within our errors. For this fit, we find

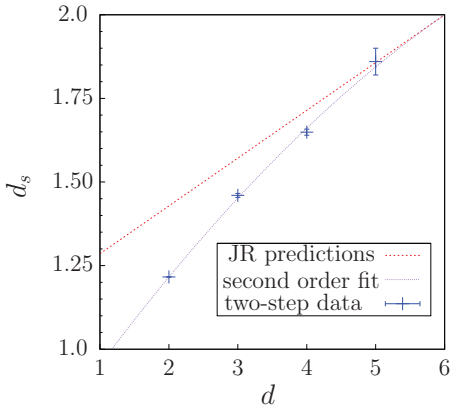

FIG. 11. (Color online) A plot comparing numerical results for d_s using the two-step method (blue points) against predictions from the $O(\epsilon)$ perturbation expansion theorized by Jackson and Read (red dashed line). Also included is an example of a compatible $O(\epsilon^2)$ fit (purple dotted line), $d_s = 2 - \frac{\epsilon}{7} + b\epsilon^2$, with a best fit value $b = -0.0133$.

$a = -0.142(8)$, which is near to the $-1/7$ suggested by Jackson and Read. We find for this fit the value $b = -0.014(2)$ for the second order prefactor.

Fixing $a = -1/7$, a one parameter fit,

$$d_s = 2 - \frac{\epsilon}{7} + b\epsilon^2 \,, \tag{23}$$

gives $b = -0.0133(1)$ with a chi-squared of 3.8 for three d.o.f. Presuming that the Jackson and Read result is correct to first order, this gives us a more precise numerical prediction for the second order term. This fit, Eq. (23), is plotted in Fig. 11.

V. SUMMARY

The intention of the two-step method was to allow the simulation of larger systems than were previously possible, reducing memory requirements of MST-finding algorithms by combining Prim's algorithm with Kruskal's algorithm. In this regard the work was successful, allowing for precise calculations of d_s. The trimming of the Prim's algorithm tree to possibly improve scaling of MTISCs constructed appears to have been unnecessary. Calculations using Prim's algorithm alone yielded almost identical results to those that used the two-step method.

We developed a data analysis method that allowed taking nonuniform correlations into account in order to obtain more accurate estimates for d_s. This analysis method should be applicable to a wide array of disordered systems with scale invariance and may be useful for future work.

The results for d_s calculated in this work are compatible with the perturbation expansion result proposed by Jackson and Read. Fitting $d_s = 2 + a\epsilon + b\epsilon^2$, we find $a = -0.142(8)$, compatible with the predicted $a = -1/7$ [17]. Fixing the first order result to the Jackson and Read result, we used an $O(\epsilon^2)$ fit, $d_s = 2 - \frac{\epsilon}{7} + b\epsilon^2$, yielding $b = -0.0133(1)$. This could be checked if a higher order analytic calculation could be computed.

ACKNOWLEDGMENTS

This work was made possible in part by NSF Grant No. DMR-1006731 and by the Syracuse University Gravitation and Relativity computing cluster, which is supported in part by NSF Grant No. PHY-0600953. This work was carried out largely using the Syracuse University HTC Campus Grid, a computing resource of approximately 2000 desktop computers supported by Syracuse University. Some of this work was discussed at the Aspen Center for Physics (NSF Grant No. 1066293).

APPENDIX A: DEFINITIONS AND ALGORITHMS

In these appendices, we present more precise definitions of the algorithms used to generate MTISCs, as well as details of the connection between T_P, T_K, and T_2. We also discuss the region in s used for the χ^2 fitting procedure to estimate the fractal dimension d_s.

Here we present a more detailed discussion of Kruskal's and Prim's algorithms. We consider these MST-finding algorithms in the context of an undirected weighted graph $G = (V, E, w)$, where V is a set of vertices, E is a set of edges connecting these vertices, and we define a weight function $w : E \mapsto \mathbb{R}$, with each edge $e \in E$ having weight $w(e)$. We consider the case of unique weights; no two edges in E have exactly the same weight.

In order to precisely describe the algorithms, it will be helpful to define some terms. A cycle or loop is a closed path such that the removal of any single edge from this path will result in the path becoming an open path, a connected set of edges with no vertex shared by more than two edges. When constructing trees, the notion of allowed and forbidden cycles is useful. Every cycle in the finite set $\mathcal{C}$, the set of all possible cycles for graph G, is chosen to belong to $\mathcal{C}_F$ (the set of forbidden cycles) or $\mathcal{C}_A$ (the set of allowed cycles), where $\mathcal{C}_F \cup \mathcal{C}_A = \mathcal{C}$ and $\mathcal{C}_F \cap \mathcal{C}_A = \emptyset$. In a typical application on a periodic lattice, the allowed cycles correspond to loops that wrap around the lattice, while forbidden cycles correspond to nonwrapping loops that can be deformed, plaquette by plaquette, to a point. A set of two or more edges is considered connected if every edge in the set shares a vertex with at least one other edge in the set. A cluster T is set of connected edges and the vertices that these edges connect. A cluster may contain cycles, whereas a tree is an acyclic cluster. T_E denotes the set of edges in the cluster and T_V denotes the set of vertices in the cluster.

Consider edge $e = (u, v)$ where $e \in E$ and $u, v \in V$. Adding e to a cluster T means that the edge set for the cluster, T_E, becomes $T_E \cup \{e\}$, and the set of vertices in the cluster, T_V, becomes $T_V \cup \{u, v\}$. A forbidden edge for a cluster T is an edge that, if added to T, would cause T to gain a forbidden cycle. An edge that is not a forbidden edge is an allowed edge. An allowed terminating edge for a cluster T is an allowed edge that, if added to T, would cause T to gain an allowed cycle (a wrapping loop). The addition of this allowed cycle will be used as the termination condition for both Kruskal's and Prim's algorithms, which are described below. ∂T is the set of adjacent edges for a cluster T, consisting of all edges that have one (but not both) endpoints in cluster T. As this set of edges forms a

border or frontier on the outer regions of the cluster, we call ∂T the frontier of cluster T. The forbidden frontier of this cluster, $\partial_F T$, is the subset of the adjacent edges that are forbidden edges for T. The allowed frontier of this cluster, $\partial_A T$, is the subset of the adjacent edges that are allowed edges for T.

To examine an edge in Kruskal's or Prim's algorithm is to select this edge during a step of the algorithm and decide whether to accept or reject this edge into a cluster based on the conditions of the algorithm (usually dealing with the weight of the edge and whether this edge is allowed or forbidden for a particular cluster).

Next we will present Kruskal's and Prim's algorithms, step by step, using the notation we have outlined above.

Kruskal's Algorithm:

(1) Sort E by increasing weight w to form a list of edges L.

(2) Initialize each vertex in the graph to be its own tree, not connected to any other vertices and containing no edges.

(3) Select the first (lowest weight) edge $e = (u,v) \in L$. Remove this edge from L. If e is a allowed edge, join into a single tree the tree containing u with the tree containing v, adding edge e. If e is a forbidden edge (that is, adding edge e to to the tree(s) containing its endpoints u and v would introduce a forbidden cycle), edge e is disregarded and not added to any trees. Edge e has now been examined.

(4) Repeat step (3) until a allowed terminating edge, i.e., one that introduces an allowed (wrapping) cycle, is selected. The tree that contains both the vertices of this edge is identified as the Kruskal's MTISC T_K, and the allowed terminating edge that is examined in this final step is the Kruskal's wrapping edge, $e_K \in E$.

Prim's Algorithm:

(1) Initialize the growing Prim's tree T_g to have one site, called the Prim's origin, $v_0 \in V$. Note that T_g is both a function of the v_0 and time (number of iterations of Prim's algorithm), since T_g "grows" from the origin v_0 by adding edges and vertices as Prim's algorithm proceeds.

(2) Sort the frontier of the current Prim's tree, ∂T_g, by increasing edge weight w to form the Prim's queue (a function of T_g), excluding any edges that have already been examined.

(3) Select the first (lowest weight) edge $e = (u,v)$ in the Prim's queue. If e is a allowed edge, add it to T_g, and if either u or v is not already in T_g, add it to T_g as well. Otherwise, if e is a forbidden edge, disregard the edge and do not add it to T_g. Edge e has now been examined.

(4) Repeat steps (2) and (3) until a allowed terminating edge, i.e., one that introduces an allowed (wrapping) cycle, is selected. At this time, the growing Prim's tree T_g is identified as the Prim's MTISC T_P, and the allowed terminating edge examined in this final step is the Prim's wrapping edge, $e_P(v_0) \in E$. Note that T_P and e_P are both functions of the Prim's origin v_0.

APPENDIX B: PROOF OF VALIDITY FOR TWO-STEP METHOD

The two-step method first applies Prim's algorithm to find $T_P \cup e_P$. Kruskal's algorithm is then executed using the edges in $T_P \cup e_P$, serving as a trimming procedure for T_P and yielding what we will term the two-step MTISC, T_2. Here we will prove that the two-step method for MTISC generation yields $T_2 = T_K$ in all cases except in the case of multiple disjoint ISCs. All results of this method can be divided into three cases. The first is the case where T_P is exactly equal to T_K and no trimming is needed, yielding $T_P = T_2 = T_K$. In the second case, T_K is a subset of T_P, and the trimming procedure of the two-step method yields $T_2 = T_K$. In the third case, there are multiple disjoint ISCs, and T_P and T_K share no common edges or vertices, meaning that $T_2 \neq T_K$. In other words, we will prove that

$$(T_P \supseteq T_K) \vee (T_P \cap T_K = \emptyset) \tag{B1}$$

and subsequently that $T_2 = T_K$ in all cases except when the Prim's MTISC and the Kruskal's MTISC do not intersect. Here it will be useful to establish a few lemmas to aid in verifying these three cases we have outlined.

Lemma 1. Edges in the allowed frontier of T_K have higher weight than edges in T_K.

Proof. Because Kruskal's algorithm examines edges that are monotonically increasing in weight as the algorithm proceeds, at the time Kruskal's algorithm terminates, the edges that are not examined must be higher weight than edges that have been examined. Since edges in the allowed frontier $\partial_A T_K$ cannot be in T_K, this means that edges in T_K have been examined at the time of Kruskal's algorithm termination, whereas edges in $\partial_A T_K$ have not. Thus edges in $\partial_A T_K$ must be higher weight than edges in T_K.

Lemma 2. For any given cycle $c \in \mathcal{C}$ for which all edges of c are examined in Kruskal's algorithm, the edge in this cycle that is examined last must be the highest weight edge in this cycle.

Proof. Because Kruskal's algorithm examines edges that are monotonically increasing in weight as the algorithm proceeds, for the finite set of edges in c, the highest weight edge in this set will be examined last in Kruskal's algorithm.

Corollary 3. As a corollary, the Kruskal's wrapping edge $e_K \in E$ has higher weight than any edge in the Kruskal's MTISC T_K.

Proof. Because e_K is the last edge examined during the running of Kruskal's algorithm, it must have a weight higher than every other edge that is examined during the running of the algorithm, including every edge in T_K.

Lemma 4. From lemma 2, for any edge e in the forbidden frontier of T_K, where adding e to T_K would form a forbidden cycle c_e, e must be the highest weight edge in the forbidden cycle c_e.

Proof. The criteria for an edge e to be in the forbidden frontier $\partial_F T_K$ is that adding e to T_K would form a forbidden cycle c_e. This means that every edge in $c_e \backslash \{e\}$ is in T_K and is examined before edge e during Kruskal's algorithm. Thus, edge e will have higher weight than any other edge in cycle c_e.

Using these lemmas that we have established, we will show that each realization of the two-step method can be categorized into one of three distinct cases, with cases 1 and 2 yielding $T_2 = T_K$. In the third case, we see that there are no common edges or vertices between T_P and T_K, so $T_2 \neq T_K$ in this case.

Case 1. If the Prim's origin v_0 is a vertex in the Kruskal's MTISC T_K, then the Prim's MTISC T_P will be identical to the Kruskal's MTISC; $T_P = T_K$.

Proof. Since Prim's algorithm grows a tree T_g by examining and adding edges adjacent to T_g, it will be crucial for us to pay

close attention to the frontier of T_g, ∂T_g. This frontier is sorted to form the Prim's queue from which edges are selected and considered for addition to T_g. In this first case, as long as we are only adding to T_g vertices that are also in T_K, the Prim's queue will consist entirely of edges that are either in T_K or ∂T_K. The fulfillment of this condition,

$$\partial T_g \subseteq T_K \cup \partial T_K \ , \tag{B2}$$

will be shown in the remainder of the proof.

From lemma 1, when the frontier of the Prim's tree is sorted in Prim's algorithm to form the Prim's queue, any edges that are also in T_K will be placed before (having lower weight than) those edges in the allowed frontier of T_K. This means that all edges in T_K will be examined and added to T_g earlier in Prim's algorithm than edges in the allowed frontier of T_K. As the Prim's origin v_0 is in T_K, we are guaranteed to have at least one edge from T_K added to the Prim's queue at the start of the Prim's growth. Furthermore, since T_K is connected, as edges from T_K are added to T_g through the Prim's growth, more edges from T_K and its frontier ∂T_K will be added to the Prim's queue, ensuring that T_g has more edges from T_K to add during further steps of Prim's algorithm. T_g begins as a single vertex v_0 and grows to include more and more edges and vertices from T_K as Prim's algorithm proceeds.

Though we have shown that edges in T_K will be examined during Prim's algorithm before any edges in the allowed frontier of T_K, we cannot say the same about edges in its forbidden frontier, $\partial_F T_K$, which may also be in the Prim's queue and in consideration for selection during the Prim's growth. If edge $x = (u_x, v_x) \in \partial_F T_K$ is in the Prim's queue, from lemma 4 we know that both vertices u_x and v_x are in T_K. If we call c_x the forbidden cycle that would be created in T_K by adding edge x to T_K, we also know from lemma 4 that x has higher weight than every other edge in the forbidden cycle c_x.

Thus, all edges in $c_x \backslash \{x\}$ will be examined in Prim's algorithm and added to T_g before x is examined. So when x is finally examined in Prim's algorithm, x will be a forbidden edge for T_g and will not be added to T_g. This ensures that any forbidden cycles for T_K will be handled in the same manner in both Kruskal's and Prim's algorithms; no edges in the forbidden frontier of T_K will be added to T_g.

Using corollary 3, we can see that until the Kruskal's wrapping edge e_K is examined in Prim's algorithm, Eq. (B2) will remain satisfied and all of the edges and vertices in T_K will be added to T_g before e_K is examined. Furthermore, as we have shown, none of the edges (allowed or forbidden) in the frontier of T_K will be added to the growing Prim's tree T_g before the Kruskal's wrapping edge e_K is examined. In this case, e_K will also serve as the Prim's wrapping edge e_P. Since the condition (B2) is satisfied at the time $e_K = e_P$ is examined in Prim's algorithm (terminating the algorithm by adding an allowed cycle), $T_P = T_K$. In this case, the Kruskal's trimming step of the two-step method is unnecessary, and the two-step method will yield $T_2 = T_K$.

Case 2. If the Prim's origin v_0 is not in the Kruskal's MTISC T_K but the Prim's tree T_g "grows" to include any vertices of T_K, then the Prim's MTISC T_P will include all of the Kruskal's MTISC T_K, i.e., $T_P \supset T_K$.

Proof. If we start Prim's algorithm from an origin v_0 that is not a vertex in the Kruskal's MTISC T_K, we can define bridge edge $b = (u_b, v_b)$ as the first edge added to the growing Prim's tree T_g that introduces a vertex of T_K into T_g. So $b = (u_b, v_b)$ is the first edge added to T_g for which either u_b or v_b are in T_K.

At the point in Prim's algorithm when b is added to T_g, b has a weight lower than any edge in the Prim's queue. Otherwise, edge b would not have been selected for addition to the Prim's tree at that time.

Examining edge b in the frame of the Kruskal's MTISC T_K, we know that edge b is in the allowed frontier of T_K since only one of u_b or v_b is in T_K. This means that b cannot be in T_K itself, nor can b be in the forbidden frontier of T_K. Because b is in the allowed frontier of T_K, it has a weight higher than any edge in T_K, by lemma 1. Thus, Prim's algorithm will continue as in case 1, with growth continuing from this first vertex of T_K (either u_b or v_b) that is introduced to T_g. Any edges from T_K that are added to the Prim's queue will be added to the front (lower weight end) since these edges will all have a weight lower than $w(b)$, whereas every edge in the Prim's queue thus far has weight higher than $w(b)$.

Prim's algorithm will add edges from the Kruskal's MTISC T_K until reaching termination with the examination of the Kruskal's wrapping edge e_K, which in this case will also be the Prim's wrapping edge e_P. At this point $T_P \supset T_K$. In this case, the Kruskal's algorithm portion of the two-step method will serve to trim from the Prim's tree the region near the origin v_0 up to the bridge b, leaving $T_2 = T_K$.

Case 3. If the Prim's origin v_0 is not in the Kruskal's MTISC T_K and the growing Prim's tree T_g does not "grow" to include any vertices of T_K before Prim's algorithm terminates, then the Prim's MTISC and the Kruskal's MTISC will not intersect. In other words, $T_P \cap T_K = \emptyset$.

Proof. In this case, during Prim's algorithm, the Prim's wrapping edge e_P is examined before any vertices of the Kruskal's MTISC T_K are examined or added to the Prim's tree. The algorithm terminates with $T_P \cap T_K = \emptyset$. In this case, the two-step method will yield $T_2 \neq T_K$. However, this case is uncommon, and it is seen by direct observation that T_2 and T_K have similar scaling properties.

Taking all three cases for the two-step method, we can say that either the Prim's MTISC T_P contains or is equal to the Kruskal's MTISC T_K, or the Prim's and Kruskal's MTISCs do not intersect. Symbolically, $(T_P \supseteq T_K) \vee (T_P \cap T_K = \emptyset)$. In cases 1 and 2, where T_P and T_K do overlap, the two-step MTISC T_2 will be equal to the Kruskal's MTISC T_K. So in these cases the two-step method yields the same result as that obtained from running Kruskal's algorithm on the entire graph G. We see that $T_2 = T_K$ in all cases except the case of multiple disjoint ISCs, where the Prim's MTISC T_P (and by construction the two-step MTISC T_2 as well) does not intersect with the Kruskal's MTISC T_K.

APPENDIX C: FITTING REGION FOR χ^2

Here we will discuss the data cutoffs we impose in order to restrict exactly what region in s of data we want to fit. These data cutoffs allow us to reduce overfitting errors in our χ^2 fitting routine [Eq. (19)] for estimating the fractal dimension d_s. We consider a small s (lower) cutoff by implementing s_l as the minimum s value allowed into the fitted data. We also

consider a large s (upper) data cutoff by enforcing an ω_u as the maximum allowed value for any of the scaled data in the fit. Introducing an ω_u cutoff seemed natural because $\omega \gg 1$ corresponds to paths that are much longer than L^{d_s} and are rare. Since these long paths are less frequent, data points with a large ω value have higher statistical uncertainties than those with smaller ω values. For small s we consider lattice effects. It made sense to use s_l as a low cutoff since we have small discrete s values (steps) in the paths. Using ω_l would also have been a viable option, but it is easier to interpret the physical meaning of s_l, since one unit of s corresponds to the lattice spacing for all system sizes.

To determine reasonable values to use for the upper and lower data cutoffs, we tested our χ^2 fitting routine with various values of these cutoffs and assessed how well the minimum value of χ^2 agreed with a predicted estimate χ_p^2. To estimate χ_p^2, it is instructive to envision comparing data sets from two different system sizes on scaled axes ρ vs. ω, as in Fig. 5(a). We use a discrete set $\{\omega_k^0\}$, consisting of n uniformly spaced values of ω, to calculate a set of $\rho(\omega_k^0)$ values for each of the two data sets. The comparison between these ρ values goes into the calculation of χ_2, as outlined in Eq. (19).

One can imagine breaking the ω axis up into b segments or "boxes" of length equal to the scaled correlation length ℓ. Each box will contain some number n_b of data points from the set of n points in the $\{\omega_k^0\}$ discrete points we use to calculate the goodness of the collapse. In this way, we can estimate χ_p^2 piece by piece, calculating separately the contribution χ_b^2 from each of the b boxes:

$$\chi_p^2 = \chi_b^2 b \,. \tag{C1}$$

Further, we can estimate n_b, the number of data points that fall in each of these boxes. This allows us to subdivide the χ_b^2 into contributions from each individual data point, χ_1^2:

$$\chi_p^2 = \chi_1^2 n_b b \,. \tag{C2}$$

Now that we have partitioned χ_p^2 by this relation, we can calculate χ_1^2, n_b, and b individually. First, we can estimate χ_1^2, the χ^2 contribution from a single data point in the set of $\{\omega_k^0\}$ values, by its expectation value $E(\chi_1^2)$. We can write, using the form of Eq. (19),

$$\begin{aligned}\chi_1^2 &\approx E\left(\chi_1^2\right) \approx \frac{E([\rho_1-\rho_2]^2)}{\ell\left[\sigma_1^2+\sigma_2^2\right]} \approx \frac{E\left(\rho_1^2\right)+E\left(\rho_2^2\right)}{\ell\left[\sigma_1^2+\sigma_2^2\right]} \\ &\approx \frac{\sigma_1^2+\sigma_2^2}{\ell\left[\sigma_1^2+\sigma_2^2\right]} \approx \frac{1}{\ell}\,,\end{aligned} \tag{C3}$$

where $\rho_1(\rho_2)$ is the ρ value taken from data set 1(2), and $\sigma_1^2(\sigma_2^2)$ is the variance at this point for data set 1(2).

Next we can write an expression for the number of data points per box, n_b, by multiplying the length of each box, ℓ, by the density of data points to obtain

$$n_b = \ell\left(\frac{n}{\omega_u-\omega_l}\right)\,, \tag{C4}$$

where again n is the total number of data points in the set $\{\omega_k^0\}$ being used for the χ^2 calculation and $\omega_u - \omega_l$ gives the full allowed range of ω_k^0 values.

Finally, we compute the total number of boxes, b, by dividing the range $\omega_u - \omega_l$ into two regions, $\omega < 1$ and $\omega > 1$. This allows us to separately compute the number of boxes in each of these regions, $b_<$ and $b_>$, respectively. Of course,

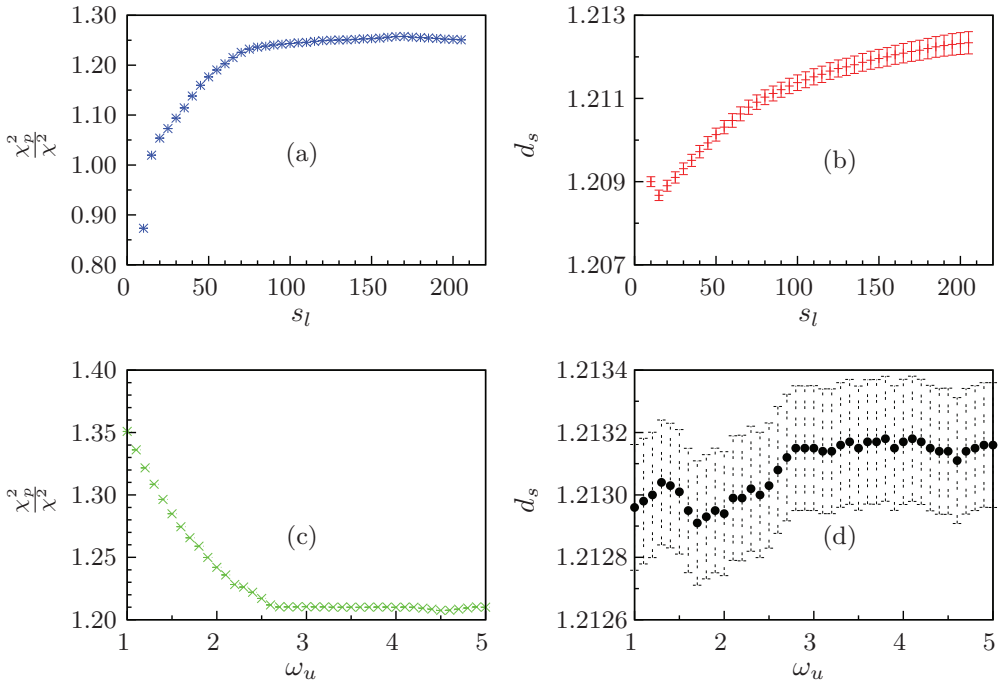

FIG. 12. (Color online) Examining the effects of the lower and upper data cutoffs on a collapse of two systems of sizes $L = 512$ and $L = 1024$ in dimension $d = 2$. (a) and (b) display χ_p^2/χ^2 and d_s as functions of the lower (small s) data cutoff s_l, while (c) and (d) show χ_p^2/χ^2 and d_s as functions of the upper (large s) data cutoff ω_u for $s_l = 100$. Both χ_p^2 and χ^2 are measured at the value of d_s for which χ^2 is minimized.

$b = b_< + b_>$. Expressing $b_>$ is trivial, since in this region the scaled correlation length is constant at $\ell = A$ [as given by our model in Eq. (15)], where A is the proportionality constant relating ℓ and ω by $\ell = A\omega$. Thus we write

$$b_> = \frac{\omega_u - 1}{A} . \tag{C5}$$

Writing $b_<$ is not quite as simple. It may help to begin at $\omega = 1$ and think of stepping left toward lower values of ω and eventually to ω_l, counting up the number of boxes we step across. The boundary of the first box will occur at $\omega = (1 - A)$, since the scaled correlation length ℓ is equal to A around $\omega = 1$. Likewise, the edge of the second box we approach will fall at $\omega = (1 - A)^2$, the ith box will reach to $\omega = (1 - A)^i$, and the final $b_<^{\text{th}}$ box will fall at the lowest allowed ω value, ω_l. So we can write

$$\omega_l = (1 - A)^{b_<} , \tag{C6}$$

from which is follows that

$$b_< = \frac{\ln(\omega_l)}{\ln(1 - A)} \approx \frac{\ln(1/\omega_l)}{A} . \tag{C7}$$

Here an approximation is made in the denominator since A is small.

Combining Eqs. (C2), (C3), (C4), (C5), and (C7), we can write an expression for χ_p^2 by adding the contributions from both of the two regions we examined, $\omega < 1$ and $\omega > 1$:

$$\begin{aligned}\chi_p^2 &= \frac{1}{\ell}\left[\ell\left(\frac{n}{\omega_u - \omega_l}\right)\right]\left[\frac{\ln(1/\omega_l)}{A}\right] \\ &\quad + \frac{1}{\ell}\left[\ell\left(\frac{n}{\omega_u - \omega_l}\right)\right]\left[\frac{\omega_u - 1}{A}\right] \\ &= \frac{1}{A}\left[\frac{n}{\omega_u - \omega_l}\right]\left[\ln\left(\frac{1}{\omega_l}\right) + \omega_u - 1\right] .\end{aligned} \tag{C8}$$

In general, this proportionality constant A will depend on what method is used to measure correlation length. For this reason, we look at the ratio χ_p^2/χ^2. Since both χ_p^2 and χ^2 have a $1/A$ dependence, the ratio of the two is independent of A.

We use this ratio χ_p^2/χ^2 to get an idea of the effect various upper and lower data cutoffs have on our fit. We plot an example case of this analysis for systems of size $L = 512$ and $L = 1024$ in dimension $d = 2$. In Figs. 12(a) and 12(b) it is apparent that below a certain s_l threshold, lattice effects skew the value of χ^2 as well as the value of d_s. Meanwhile, Figs. 12(c) and 12(d) indicate that we need not impose an upper cutoff on the data. The high variance of points in the large ω region of the data already appropriately weights these points. For our final analysis, we chose $s_l = 425$ for two dimensions, $s_l = 375$ for three dimensions, and $s_l = 100$ for four and five dimensions.

[1] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. (MIT Press, Cambridge, MA, USA, 2001).
[2] E. W. Dijkstra, Numer. Math. **1**, 269 (1959).
[3] V. Petäjä, M. Sarjala, M. Alava, and H. Rieger, Phys. Rev. B **73**, 094517 (2006).
[4] D. A. Huse and C. L. Henley, Phys. Rev. Lett. **54**, 2708 (1985).
[5] H. Luo, M. Schulz, L. Schlke, S. Trimper, and B. Zheng, Phys. Lett. A **250**, 383 (1998).
[6] A. A. Middleton, Phys. Rev. B **61**, 14787 (2000).
[7] A. C. Carter, A. J. Bray, and M. A. Moore, Phys. Rev. Lett. **88**, 077201 (2002).
[8] E. Fehr, D. Kadau, N. A. M. Araújo, J. S. Andrade, and H. J. Herrmann, Phys. Rev. E **84**, 036116 (2011).
[9] C. M. Newman and D. L. Stein, Phys. Rev. Lett. **72**, 2286 (1994).
[10] R. C. Prim, Bell Syst. Tech. J. **36**, 1389 (1957).
[11] W. Chou and A. Kershenbaum, in *Proceedings of the Third ACM symposium on Data Communications and Data Networks: Analysis and Design*, *DATACOMM '73* (ACM, New York, 1973), pp. 148–156.
[12] H. Loberman and A. Weinberger, J. ACM **4**, 428 (1957).
[13] A. W. F. Edwards and L. L. Cavalli-Sforza, in *Phenetic and Phylogenetic Classification*, edited by V. H. Heywood and J. Mcneill, No. 6 (Systematics Association Publ. London, 1964), pp. 67–76.
[14] R. Osteen and P. Lin, SIAM J. Comput. **3**, 23 (1974).
[15] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, MA, USA, 1990).
[16] T. S. Jackson and N. Read, Phys. Rev. E **81**, 021130 (2010).
[17] T. S. Jackson and N. Read, Phys. Rev. E **81**, 021131 (2010).
[18] S. R. Broadbent and J. M. Hammersley, Proc. Cambridge Philos. Soc. **53**, 629 (1957).
[19] H. Frisch and J. Hammersley, J. Soc. Ind. Appl. Math. **11**, 894 (1963).
[20] M. Aizenman, Nucl. Phys. B **485**, 551 (1997).
[21] M. Aizenman, IMA Vol. Math. Appl. **99**, 1 (1996).
[22] D. Wilkinson and J. F. Willemsen, J. Phys. A: Math. Gen. **16**, 3365 (1983).
[23] J. Goodman, J. Stat. Phys. **147**, 919 (2012).
[24] M. Damron, A. Sapozhnikov, and B. Vgvlgyi, Ann. Probab. **37**, 2297 (2009).
[25] J. van den Berg, Y. Peres, V. Sidoravicius, and M. E. Vares, Annales de l'Institut Henri Poincar - Probability and Statistics = Probabilities and Statistics **44**, 1173 (2008).
[26] J. B. Kruskal, Proc. Am. Math. Soc. **7**, 48 (1956).
[27] A.-L. Barabási, Phys. Rev. Lett. **76**, 3750 (1996).
[28] J. Machta, Y. S. Choi, A. Lucke, T. Schweizer, and L. V. Chayes, Phys. Rev. Lett. **75**, 2792 (1995).
[29] G. Paul, R. M. Ziff, and H. E. Stanley, Phys. Rev. E **64**, 026115 (2001).
[30] P. Grassberger, Phys. Rev. E **67**, 036101 (2003).
[31] P. Duxbury and R. Dobrin, Physica A (Amsterdam) **270**, 263 (1999).
[32] M. Damron and A. Sapozhnikov, The Annals of Probability **40**, 893 (2012).
[33] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 1st ed. (Cambridge University Press, New York, 1988).

[34] G. E. P. Box and D. A. Pierce, J. Am. Stat. Assoc. **65**, 1509 (1970).
[35] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, Gnu Scientific Library Reference Manual, 3rd ed., Network Theory Ltd., http://www.gnu.org/software/gsl, 2003.
[36] A. Coniglio, Phys. Rev. Lett. **62**, 3054 (1989).
[37] S. N. Majumdar, Phys. Rev. Lett. **68**, 2329 (1992).
[38] M. Cieplak, A. Maritan, and J. R. Banavar, Phys. Rev. Lett. **76**, 3754 (1996).
[39] S. V. Buldyrev, S. Havlin, E. López, and H. E. Stanley, Phys. Rev. E **70**, 035102 (2004).
[40] A. P. Sheppard, M. A. Knackstedt, W. V. Pinczewski, and M. Sahimi, J. Phys. A: Math. Gen. **32**, L521 (1999).
[41] B. Wieland and D. B. Wilson, Phys. Rev. E **68**, 056101 (2003).
[42] K. Dahmen and J. P. Sethna, Phys. Rev. Lett. **71**, 3222 (1993).