*Journal of Statistical Physics, Vol. 74, Nos. 3/4, 1994*

# Cluster Hull Algorithms for Large Systems with Small Memory Requirement

**H. Vollmayr**[1,2]

*Received August 24, 1993; final September 23, 1993*

Improved hull walking algorithms for two-dimensional percolation are proposed. In these algorithms a walker explores the external perimeter of percolation clusters. With our modifications very large systems (size $L$) can be studied with finite and small memory requirement and in computation time $\tau \sim L^{7/4}$. Applications in determining spanning probabilities, continuum percolation, and percolation with nonuniform occupation probability are pointed out.

**KEY WORDS:** Two-dimensional percolation; hull algorithm; maze walker; cluster perimeter; Monte Carlo; random number generator; continuum percolation; gradient percolation; self-organized criticality.

Percolation has been studied extensively in recent years (see ref. 1 for an overview). It provides examples of critical phenomena which are easily accessible to numerical investigation. Applications range from geology (flow in porous media) to ecology (forest fires) to thermal phase transitions[2] and the quantum Hall effect.[3]

One of the most elementary problems in percolation is to determine whether a percolating cluster connecting opposite sides exists in a finite system. Hoshen and Kopelman[4] introduced an algorithm which performs this task in a time of the order of the volume of the system, $\tau \sim L^d$, apart from logarithmic factors. It also yields the probability distribution of clusters with size.

[1] Department of Physics, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801-3080.

[2] Current and permanent address: Institut für Theoretische Physik, 37073 Göttingen, Germany.



0022-4715/94/0200-0919$07.00/0 

Ziff *et al.*[5] proposed an algorithm which explores the hull (the external perimeter) of large clusters. A walker (e.g., a highly trained ant) proceeds on occupied lattice sites along the hull always keeping to the left. The algorithm corresponds to a well-known strategy to avoid getting lost in a two-dimensional maze. A walker who always touches the (locally) left wall is certain to return to the starting point before passing any point in the maze more than twice. Indeed the algorithm has been used in maze walking programs (see, e.g., ref. 6).

In one of the first applications to percolation it was used to determine the fractal dimension $D_h = 7/4$ of the hull of large clusters.[5,7] This result is now known to be exact.[8,9]

The question whether a spanning cluster, connecting top and bottom of a system of $L \times L$ sites, exists can be decided in the following way.[10] The leftmost column of sites is held unoccupied while the lowest row is completely occupied. The walker begins in the lower left corner and walks until it reaches the top row (in which case there is a spanning cluster) or the rightmost column (in which case there is no spanning cluster). The spanning probability $R_L(p)$ has been determined accurately using this algorithm.[11]

The main advantage of hull walking algorithms—besides their simplicity—is that only the sites on the hull and their unoccupied nearest neighbors to the left have to be generated. (To "generate a site" means to decide whether it is occupied by generating a random number. Each site is generated locally, i.e., only at the time when the walker needs to know its state.) For large systems this is only a small fraction of the whole lattice.

However, there is the following problem. If the walker comes back to a previously visited site, it must find the site in the same state (occupied or unoccupied) as before—unless one wants to study a trivial annealed model. We are going to discuss a strategy which has been used before to handle this problem (labeled A below), and then propose two new methods (B and C below).

In ref. 11 every site is in one of three states: unvisited (0), unoccupied (1), occupied (2). Before the walker starts, all sites are set unvisited. Along the run a previously unvisited site is generated and set to the appropriate state, while a previously visited site is not generated again. We call this strategy A. While it solves the problem of multiply visited sites, it introduces limitations for the algorithm. First, memory space of the order $L^2$ is required to store the states of all sites in the system. Second, for very large $L$ the initialization time scales as $\tau_{\text{init}} \sim L^2$, for we need to set all sites unvisited. The actual walk takes time proportional to its length $n_w$, i.e., $\tau_w \sim n_w \sim L^{7/4}$; so in the thermodynamic limit most of the time is used to initialize sites, which will never be visited. However, for practical purposes

it is the memory requirement that will limit the system sizes $L$ that can be studied, because the initialization can be done very efficiently on many machines.

Strategy B consists of storing only the states of sites that have been visited in a "balanced tree."[12,13] Before generating a new site the tree must be searched to find out whether the site has been generated before, and if so, whether it is occupied. The tree requires memory space proportional to the number of steps in the walk $n_w \sim L^{7/4}$. Searching and storing requires CPU-time proportional to $\ln(n_w)$ for one site, thus the total computation time scales as $\tau \sim n_w \ln(n_w) \sim L^{7/4} \ln(L)$. Therefore, apart from the logarithmic factor, the faster scaling $\sim L^{7/4}$ of the hull survives for large $L$. The memory limitation is less severe than for strategy A, but is still present. Searching and sorting in balanced trees is nontrivial, but we can use the corresponding programs from ref. 13 for our purpose with no substantial modification.

Strategy C makes use of a particular random number generator (RNG).[14][3] It generates a random number `r` by applying a very nonlinear function $f$ to an input number `i`, i.e., `r` $= f($`i`$)$. The generated numbers are random in the sense that there are no detectable correlations between input numbers and output ("random") numbers. In particular, very similar input numbers correspond to apparently uncorrelated output numbers. We give a brief sketch of the nonlinear function $f($`i`$)$. Here `i` and `r` are 64-bit words. (Actually both numbers are represented by two 32-bit unsigned integers, which are called "left 32-bit word" and "right 32-bit word.") The function $f$ consists in four iterations of a "cycle," during which the old right 32-bit word becomes the new left word and the new right word is the result of a 32-bit XOR operation with the arguments (left word) and $g$(right word). $g$ is a nonlinear function which includes multiplication, addition, and XOR operations applied to the 16-bit long left and right parts of its argument. For details we refer to ref. 14.

To generate a site at $(x, y)$ we use its address as input to the RNG and obtain a 64-bit random number `r` $= f(x, y)$, which is used to decide whether the site is occupied. In this manner we are able to generate random occupation probabilities that depend only on the site which is currently visited—and not on how many random numbers have been generated before by a (somewhat faster) sequential RNG. Obviously there is no need to store the state of any site. If we confine ourselves to the study of a system with size $L = 2^{20} \approx 10^6$, the unused 12 bits in left and right

[3] The first edition of ref. 14 does not include the RNG, which was used here, but discusses its parent, the "Data Encryption Standard" (DES), which is probably better—i.e., has fewer correlations—but much slower.

words $(x, y)$ can be used to number different runs, which are statistically independent for practical purposes.

Now we discuss some results obtained with strategy B. As mentioned above, the computation time for the spanning algorithm is proportional to the length of the walk $n_w$, apart from logarithmic factors. Two cases must be distinguished. Consider first the critical point $p = p_c$. The total length of the walk $n_w$ can be divided into two parts $n_w = n_w^{(1)} + n_w^{(2)}$. If the system percolates from bottom to top, $n_w^{(1)}$ is defined as the length of the walk until the walker visits the bottom row for the last time. $n_w^{(2)}$ is then the length of the walk on the hull of the spanning cluster. $n_w^{(1)}$ can be interpreted as the length of the walk to find the spanning cluster. If the system does not percolate, $n_w^{(1)}$ is defined as the length of the path until the walker visits the left column for the last time. $n_w^{(2)}$ is proportional to $L^{D_h}$, where $D_h = 7/4$ is the fractal dimension of the hull. $n_w^{(1)}$ is found numerically to be also

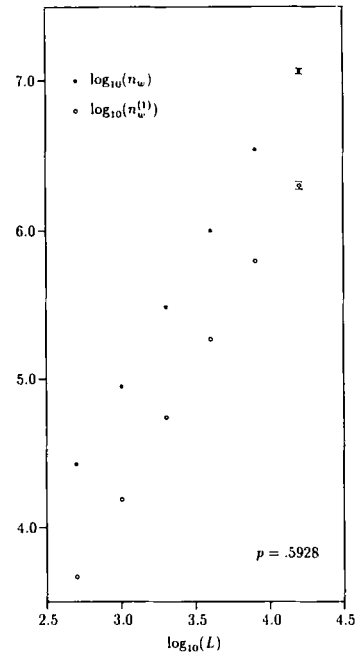


Fig. 1. At the critical point the length of the walk scales as $n_w \sim L^{7/4}$. Here $n_w^{(1)}$ is the time which is needed to find the percolating cluster (or wall). It follows the same scaling law. The data points are averages over 1000 runs for $L = 500$, 1000, 2000, and 4000, over 500 runs for $L = 8000$, and over 100 runs for $L = 16{,}000$. All runs together took about 40 hr on a SPARCstation 1. Where error bars are missing, the statistical error is smaller than the size of the dots.

proportional to $L^{D_h}$ (Fig. 1). So the total walk $n_w$ is also proportional to $L^{D_h}$; numerically we find (Fig. 1)

$$\tau \sim L^{D'_h}, \qquad D'_h = 1.754 \pm 0.007 \tag{1}$$

The computation of the data points in Fig. 2 took about 40 hr on a SPARCstation 1.

If $p > p_c$, the dimension of the percolating cluster is 2 and the walker will essentially stay close to the left wall, its mean distance being of the order of the correlation length $\xi$. If $1 \ll \xi \ll L$, we can divide the left stripe of the system where the walker advances into $L/\xi$ clusters of unoccupied sites with linear extent $\xi$ and hull $\sim \xi^{7/4}$. The walker goes along the hull of these clusters and hits the straight left wall between two clusters. So the length of its total walk scales as

$$n_w \sim \xi^{7/4}(L/\xi) = L\xi^{3/4} \sim L(p - p_c)^{-1} \tag{2}$$

For $p < p_c$ the same argument holds for the dual system, i.e., for non-occupied sites which are defined to be connected if they are nearest or next nearest neighbors. So the walker stays close to the bottom row with mean distance proportional to $\xi$ and the same scaling behavior (2), which is confirmed by Fig. 2. For $p \to p_c$ the computation time is bounded by (1), which leads to a saturation at $n_w^{\max}(L = 16{,}000) \approx 1.15 \times 10^7$. The time is measured in walker steps, so it counts the occupied sites which are visited by the walker, where some sites are counted twice. For the system studied here, $L = 16{,}000$, $L^2/n_w^{\max} \approx 23$.

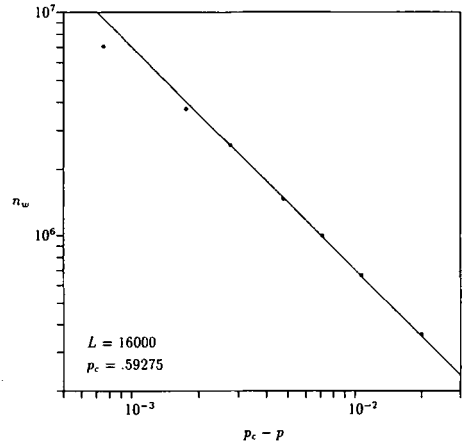


Fig. 2. Off the critical point the length of the walk scales as $n_w \sim |p - p_c|^{-1}$. Close to $p_c$ finite-size effects bound $n_w$. The data points are averages over 100 runs, respectively.

Both strategies B and C can be used for continuum percolation like the two-dimensional Swiss-cheese model, e.g., in Rosso's algorithm.[15]

We applied strategy C to the gradient hull algorithm introduced by Ziff and Sapoval.[16] Consider site percolation on a square lattice where the occupation probability depends on the position $p(x, y)$. To be specific, let $p(x, y) = p_0 - Gy$. Then there is a gradient $G$ in the occupation probability. All sites $(x = 0, y > 0)$ are left unoccupied and all sites $(x = 0, y \leqslant 0)$ are set occupied. The walker starts at the origin heading to the right. If $G$ is small $(G < 10^{-3})$, the walker will essentially go to the right at a height $y$ which fluctuates around $y_c$, defined by $p(x, y_c) = p_c$. The typical spatial fluctuation $\xi_G$ and the corresponding fluctuation $\Delta p_G$ can be estimated by a heuristic argument.[9,16] For $\Delta y \equiv y - y_c > \xi_G$ we have finite clusters with a typical size $\xi \sim |p - p_c|^{-4/3} \sim (G\,\Delta y)^{-4/3}$. The clusters grow together at $\Delta y_G = \xi_G$, which leads to

$$\xi_G \sim G^{-4/7} \qquad \text{and} \qquad \Delta p_G \sim G^{3/7} \tag{3}$$

This is confirmed by Fig. 3. If a very long walk is performed, the mean value $\langle p(x, y)\rangle = p_0 - G\langle y\rangle$ approaches $p_c^{(G)}$, providing an estimate for $p_c$. The largest value of $x$, $x_{\max}$, can be found by dividing the walk into $x_{\max}/\xi_G$ pieces of size $\xi_G$. As the fractal dimension of the hull is $D_h = 7/4$, the walker goes $\sim \xi_G^{7/4}$ steps on each piece, so that $n_w \sim \xi_G^{7/4} x_{\max}/\xi_G \sim x_{\max} G^{-3/7}$. The statistical error for $p_c^{(G)}$ is given by $\Delta p_G$ divided by the number of independent measurements, which is the number of pieces of

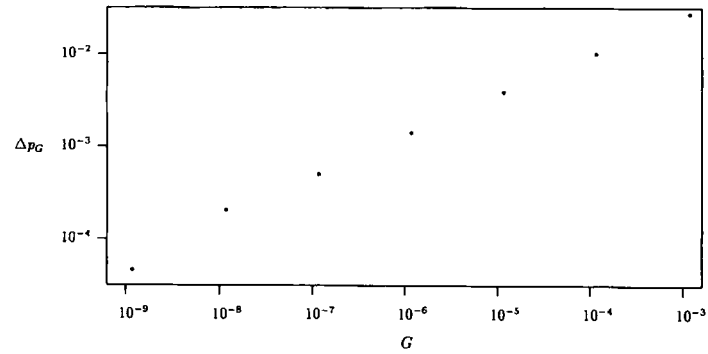


Fig. 3. In the hull gradient algorithm the walker moves in a region where $p(x, y) \approx p_c^{(G)} \approx p_c$. The fluctuations $\Delta p_G$ are proportional to $G^{3/7}$, where $G = |\nabla p|$ is the gradient. For small gradients, fluctuations are small, but the statistical error grows (see text).

size $\xi_G$: $x_{\max}/\xi_G \sim n_w G$. So we find for the statistical error $\Delta p_c^{(G)} \sim \Delta p_G (n_w G)^{-1/2} \sim G^{-1/14} n_w^{-1/2}$, i.e.,

$$n_w \sim G^{-1/7} (\Delta p_c^{(G)})^{-2} \tag{4}$$

This result means that in order to obtain $p_c^{(10^{-7}G)}$ with the same accuracy as $p_c^{(G)}$, we need to run the program ten times longer. Of course, $G$ must be large enough that $x_{\max}/\xi_G > 1$ and one must be certain that $|p_c^{(G)} - p_0| < p_G$. (If not, one has to wait for equilibration, see below.) Because $p_c^{(G)} \xrightarrow{G \to 0} p_c$, we are interested in $p_c - p_c^{(G)}$ for small $G > 0$. In part the finite-$G$ deviation is characterized by $\Delta p_G$. There is no convincing argument which could tell us how the mean $\langle p(x, y) \rangle$ should be taken—as the arithmetic mean $p_0 - G\langle y \rangle$ (which we chose), as a geometric mean, as the mean occupation probability of the sites along the walk,[16,17] or anything else. However, it does not seem unplausible—and simulations strongly indicate this—that for very small $G$ all definitions of $p_c^{(G)}$ approach each other and $p_c$ more rapidly than $\Delta p_G$ approaches zero, i.e., $(p_c - p_c^{(G)})/\Delta p_G \xrightarrow{G \to 0} 0$. This needs some more investigation. Another limitation of the accuracy of $p_c^{(G)}$ is the fact that $p(x, y)$ on a lattice is quantized. It is very surprising that $p_c^{(G)} - p_c$ has been found[16] to be at least two orders of magnitude smaller than $G = 4 \times 10^{-5}$, which is the step $p(x, y) - p(x, y+1)$ for nearest neighbors. We can avoid any uncertainty about the convergence of $p_c^{(G)} - p_c$ by choosing $G$ small enough that the error is dominated by $\Delta p_G$—but at some cost. First the statistical error will increase according to (4), which is not too costly, because 1/7 is a small exponent. Second $\xi_G$ and the necessary size of our system will grow; however, this does not matter if strategy C is used.

$p_c$ can also be measured the following way. First let $G$ be small enough that $\Delta p_G \sim G^{3/7}$ is our desired accuracy. Then start with $p_0$ being the best known value of $p_c$ at (0, 0). After $n_w$ steps the walker has reached the critical region, i.e., $p(x, y) - p_c < \Delta p_G$. The $n_w$ is determined as follows. The walker goes on the hull (length $\sim \xi^{7/4}$) of finite clusters (size $\xi \sim \Delta p^{-4/3}$) which "stick" to the boundary at $x = 0$, until it reaches $\Delta y \approx \xi_G$, when it begins to move to the right. The total length is

$$\begin{aligned} n_w &= \int_{\Delta y = \Delta y_0}^{\Delta y = \xi_G} dn_w \sim -\int_{\Delta y_0}^{\xi_G} dy\, \xi^{7/4}/\xi \\ &\sim -\int_{\Delta y_0}^{\xi_G} dy\, (\Delta p^{-4/3})^{3/4} \sim -G^{-1} \int_{\Delta y_0}^{\xi_G} dy \frac{1}{y} \\ &\sim G^{-1} \ln \frac{\Delta y_0}{\xi_G} \sim G^{-1} \ln \frac{\Delta p_{\text{old}}}{\Delta p_{\text{new}}} \end{aligned} \tag{5}$$

So apart from the logarithmic factor, $n_w \sim G^{-1} \sim \Delta p_c^{-7/3}$. This is again a little slower than the statistical limit $\Delta p_c^{-2}$. It is an interesting question whether there are algorithms to determine $p_c$ which scale faster, i.e., $\tau \sim \Delta p_c^{>-7/3}$, for $\Delta p_c \to 0$. The results in ref. 16 seem to indicate this.

Another application of strategy C is to consider a more complicated function $p(x, y)$. As has been pointed out in ref. 17, for gradient percolation $p_c^{(G)}$ is approached automatically. If we let $p(x, y) = p_0 + ky/x$, the gradient $G \approx k/x$ becomes smaller the farther the walker proceeds. In this case $p_c$ (not only $p_c^{(G)}$) is approached automatically, i.e., with ever increasing accuracy, much like in invasion percolation.[18] We regard this as an instance of self-organized criticality. Scaling arguments similar to those mentioned above lead to $n_w \sim (\Delta p_G)^{-10/3}$, which would be rather slow if $\Delta p_G$ were a typical measure for the accuracy of the determination of $p_c$.

For arbitrary $p(x, y)$ (gradients should be small everywhere) the mean movement of the walker depends on whether or not it is in the critical region, $|p(x, y) - p_c| < \Delta p_{G=|\nabla p|} \sim |\nabla p|^{3/7}$. In the critical region the drift perpendicular to the gradient is dominant and the average velocity is $v \sim |\nabla p|^{3/7}$. This is easily obtained from $x_{\max}/n_w$ (see above). Outside the critical region the average velocity is oriented (anti)parallel to the gradient in the direction of $p_c$ with an absolute value $v \sim \xi^{-3/4} \sim |\nabla p|^{3/7}$. One expects a transition region for $|p(x, y) - p_c| \approx |\nabla p|^{3/7}$.

To conclude, we propose two new strategies for handling the "return problem" in hull walking algorithms for two-dimensional percolation. One of them is based on the use of a particular random number generator.[14] The strategies extend the application of hull algorithms to extremely large systems, taking full advantage of fast scaling with finite (small) memory space. Several applications in determining spanning probabilities, continuum percolation, and generalized gradient percolation have been pointed out.

## ACKNOWLEDGMENTS

We thank D. Stauffer and S. Sondhi for valuable comments on the manuscript. This work has been done during a stay at the University of Illinois at Urbana-Champaign. It is a pleasure to thank A. J. Leggett and the Physics Department for their hospitality. We gratefully acknowledge support by the Deutsche Forschungsgemeinschaft.

## REFERENCES

1. D. Stauffer and A. Aharony, *Introduction to Percolation Theory* (Taylor and Francis, London, 1992).
2. R. H. Swendsen and J. S. Wang, *Phys. Rev. Lett.* **58**:86 (1987); J. S. Wang and R. H. Swendsen, *Physica A* **167**:565 (1990).

3. S. A. Trugman, *Phys. Rev. B* **27**:7539 (1983).
4. J. Hoshen and R. Kopelman, *Phys. Rev. B* **14**:3428 (1976).
5. R. M. Ziff, P. T. Cummings, and G. Stell, *J. Phys. A* **17**:3009 (1984).
6. G. E. Lautenbaugh, Jr., and R. Smedley, *C through DESIGN* (Franklin, Beedle and Associates, Wilsonville, Oregon, 1988).
7. R. F. Voss, *J. Phys. A* **17**:L373 (1984); T. Grossman and A. Aharony, *J. Phys. A* **19**:L745 (1986); R. M. Ziff, *Phys. Rev. Lett.* **56**:545 (1986).
8. H. Saleur and B. Duplatier, *Phys. Rev. Lett.* **58**:2325 (1987); A. Bunde and J. F. Gouyet, *J. Phys. A* **18**:L285 (1985).
9. B. Sapoval, M. Rosso, and J. F. Gouyet, *J. Phys. Lett.* (Paris) **46**:L149 (1985).
10. P. Grassberger, *J. Phys. A* **25**:5475 (1992).
11. R. M. Ziff, *Phys. Rev. Lett.* **69**:2670 (1992).
12. G. M. Adel'son-Vel'skiĭ and E. M. Landis, *Dokl. Akad. Nauk SSSR* **146**:263 (1962) [*Sov. Math.* 3:1259 (1962)]; D. E. Knuth, *Sorting and Searching* (Addison-Wesley, Reading, Massachusetts, 1973).
13. R. Sedgewick, *Algorithms in C* (Addison-Wesley, Reading, Massachusetts, 1990).
14. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. (Cambridge University Press, Cambridge, 1992), pp. 300–304.
15. M. Rosso, *J. Phys. A* **22**:L131 (1989).
16. R. M. Ziff and B. Sapoval, *J. Phys. A* **19**:L1169 (1986).
17. M. Rosso, J. F. Gouyet, and B. Sapoval, *Phys. Rev. B* **32**:6053 (1986).
18. D. Wilkinson and J. F. Willemsen, *J. Phys. A* **16**:3365 (1983); D. Wilkinson and M. Barsony, *J. Phys. A* **17**:L129 (1984).

*Communicated by D. Stauffer*