J. Phys. A: Math. Gen. 19 (1986) L1169-1172. Printed in Great Britain

LETTER TO THE EDITOR

The efficient determination of the percolation threshold by a frontier-generating walk in a gradient

Robert M Ziff† and B Sapoval‡

† Department of Chemical Engineering, The University of Michigan, Ann Arbor, Michigan 48109, USA

‡ Laboratoire de Physique de la Matière Condensée, Ecole Polytechnique, 91128 Palaiseau, France

Received 11 September 1986

Abstract. The frontier in gradient percolation is generated directly by a type of self-avoiding random walk. The existence of the gradient permits one to generate an infinite walk on a computer of finite memory. From this walk, the percolation threshold p_c for a two-dimensional lattice can be determined with apparently maximum efficiency for a naive Monte Carlo calculation ($\pm N^{-1/2}$). For a square lattice, the value $p_c = 0.592\,745 \pm 0.000\,002$ is found from a simulation of $N = 2.6 \times 10^{11}$ total steps (occupied and blocked perimeter sites). The power of the method is verified on the Kagomé site percolation case.

Recently, two new approaches for studying percolation clusters in two dimensions and for efficiently finding the critical percolation probability p_c have been independently developed. In one approach, Ziff *et al* [1] introduced a random walk which generates the perimeter of the percolation cluster directly, and this walk has been used to find p_c by finding the point where internal and external perimeters are generated with equal probability [2]. In the other approach, Sapoval *et al* [3] introduced the idea of studying percolation in a system with a gradient in the occupation probability, which leads to an extended and controlled frontier between the percolating and non-percolating regions and which also allows p_c to be found (see below). These two approaches, both based upon perimeters of percolation clusters, have led to very precise values of p_c.

Here we point out that these approaches can be combined to generate the frontier directly and to find p_c even more efficiently. Simply, if the perimeter-generating walk of [1] is carried out in a gradient of p, then the frontier of [3] will be exactly generated, without generating any of the occupied sites, in either region, that are not part of the frontier. (For the algorithm of the walk, see [1].) In essence, the walk is a very efficient method to generate the frontier alone. The value of p_c may be found from the frontier [4] by two methods. In the first, p_c is found as the average value of the probability sampled by both the occupied and blocked sites of the walk:

$$p_c = p(\bar{y}) \tag{1}$$

where $\bar{y}$ is the average y value of all the sites in the walk and $p(y)$ is the occupation probability at y (the gradient is assumed to be in the $-y$ direction). The second method is to find p_c is by the formula

$$p_c = N_{occ}/N_{total} \tag{2}$$

where N_{occ} is the number of occupied sites and N_{total} is the number of occupied plus blocked sites generated in the walk.

The net effect of the gradient on the walk is to make it move, on the average, in a direction perpendicular to the direction of the gradient. The strength of the gradient controls the width or the correlation length of the walk [3]. When the gradient is very high, the walk is squeezed to a straight line, and (1) or (2) gives $p_c = 0.5$. As the gradient is relaxed, the walk expands and the apparent p_c increases as more occupied than blocked sites are generated. The average width σ of the walk is $\approx 0.5|\nabla p|^{-0.57}$ [3].

We have carried out extensive simulations of this walk on a two-dimensional square lattice for the purpose of determining p_c. A lattice of 2048×2048 sites was used, with a gradient of $1/40\,000$ in the $-y$ direction and the value $p = 0.593$ along the line $y = 1024$, so p varied linearly between about 0.567 and 0.619. According to the analysis in [4], the finite-gradient correction to p_c in this case is $\approx 5 \times 10^{-7}$, which is smaller than the statistical error that can be reached in our computations and so can be ignored. Also with this gradient $\sigma \approx 215$, so the edges of this lattice in the y direction should very rarely be hit, and the information in the x direction can be forgotten after one pass of the box.

To start the walk without any closures (a completed perimeter), which would be common in such a low gradient, the $x = 0$ line was filled with occupied sites from $y = 0$ to 1023 and blocked sites from $y = 1024$ to 2047, and the walk was started at $x = 1$, $y = 1023$. This boundary kept the walk at $x > 0$. After a while, the walk got 'started' and continued in the positive x direction, as shown in figure 1. The statistics of the first pass of the lattice were discarded to eliminate any bias from the startup process.

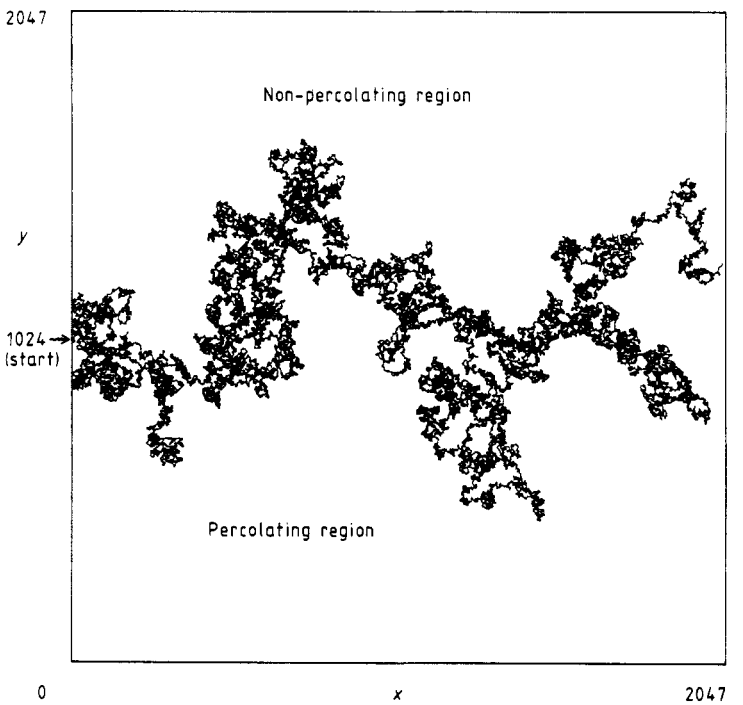

Figure 1. The first pass of the walk on a 2048×2048 lattice. The gradient is in the $-y$ direction. It can be seen that the walk hit the boundary $x = 0$ several times at the start. This example shows rather large deviations in the position of the walk; more typical ones tend to stay closer to the centre ($y \approx 1024$). This walk was stopped when $x = 2047$ was first reached, and a total of 322 782 occupied plus blocked sites were generated. This figure may be compared with the perimeter of a single large percolation cluster shown in [2].

Every time the walk hit a new higher value in the x direction, that column in the lattice was reset to be blank (unvisited) sites for all y. When the walk hit the end $x = 2047$, periodic boundary conditions were used to wrap it back to $x = 0$. Thus the walk could be continued indefinitely using a finite-memory computer.

A total of 2.6×10^{11} occupied plus blocked sites were generated by carrying out simultaneous runs on many Apollo workstation computers (with different random number seeds), using about 5500 total hours of computer time. (These are small minicomputers, at least two orders of magnitude slower than a high-speed mainframe or supercomputer.) The boundaries in the y direction were hit only about 30 times, at which point the simulations were stopped. Violation in the x direction (in which the walk makes it back to discarded sites) was never found. Using either (1) or (2), we find

$$p_c = 0.592\,745(2) \tag{3}$$

where the number in parentheses represents the error in the last digit at the 68% confidence level (one standard deviation). The results of the other determinations of p_c are shown in table 1. It can be seen that our value is consistent with previous results and is at least an order of magnitude more precise.

Table 1. Determinations of p_c for site percolation on a square lattice.

Value	Method	Reference	Year
0.48	Fifth-order series	[6]	1960
0.55	Ninth-order series	[7]	1961
0.581(15)	MC on 2000 sites	[8]	1961
0.580(18)	MC on 78^2	[9]	1963
0.59(1)	Tenth-order series	[10]	1964
0.593(2)	Nineteenth-order series	[11]	1976
0.595	MC on 1000^2	[12]	1976
0.591(1)	Series analysis	[13]	1976
0.592 7(3)	MC on 4000^2	[14]	1978
0.593 1(6)	MC on 500^2	[15]	1980
0.592 7(2)	Transfer matrix	[16]	1982
0.592 3(7)	Series analysis	[17]	1982
0.592 77(5)	MC on $50\,000^2$	[18]	1984
0.592 7(1)	MC on $160\,000^2$	[19]	1985
0.592 74(10)	Transfer matrix	[20]	1985
0.592 80(1)	Gradient frontier	[4]	1985
0.592 75(3)	Perimeter walks	[2]	1986
0.592 73(6)	Transfer matrix	[21]	1986
0.592 745(2)	Frontier walks	This work	1986

The error quoted in (3) represents both the observed deviation from different runs, and the minimum deviation implied by the statistical uncertainty $N^{-1/2}$. Because the observed error is the statistical minimum, and because no points other than those used in the calculation were generated, we believe that this is the most efficient method possible for a naive Monte Carlo measurement of p_c. We note that in previous determinations of p_c, the given error is in general much larger than the statistical limit $N^{-1/2}$, where N is the total number of sites visited (or random numbers generated). We note also the simplicity of the program to generate these walks: the main part is

about 20 lines long, and there is no complicated bookkeeping or labelling, no cluster searching, no binning of distributions, and no delicate extrapolations. Furthermore, essentially only one run needs to be carried out; one does not have to try different values of p as in most other methods.

As a check on the method and the random number generator (a Tausworthe-type shift generator), some runs were also carried out on a lattice with known p_c. The triangular lattice, which is often used for this purpose because p_c is exactly $\frac{1}{2}$, is not ideal because of the perfect symmetry between the occupied and blocked sites. Instead we used the Kagomé lattice (the matching lattice of bond percolation on the honeycomb lattice), for which p_c(site) is known exactly as $1-2\sin(\pi/18)=0.652\,7036$ [5]. Generating 1.2×10^{10} steps on the 2048×2048 lattice with a gradient of $1/25\,000$, we found $N_{occ}/N_{total}=0.652\,704(9)$. The mean turned out to be correct well within one standard deviation.

In conclusion, we find that applying the generating walk to the frontier in a gradient is a very efficient and well controlled way of generating those frontiers, and probably the best Monte Carlo way to find p_c for a two-dimensional lattice. We have found the most precise value of p_c for a square lattice given to data. The striking agreement between p_c found by this method and the other varied methods is good evidence that the general understanding of the percolation process in finite and infinite systems that underlies these studies is correct.

References

[1] Ziff R M, Cummings P T and Stell G 1984 *J. Phys. A: Math. Gen.* **17** 3009
[2] Ziff R M 1986 *Phys. Rev. Lett* **56** 545
[3] Sapoval B, Rosso M and Gouyet J F 1985 *J. Physique Lett.* **46** L149
[4] Rosso M, Gouyet J F and Sapoval B 1986 *Phys. Rev.* B **32** 6053
Gouyet J F, Rosso M, Sapoval B, Cassereau S and Couture B 1987 to be published
[5] Sykes M F and Essam J W 1963 *Phys. Rev. Lett.* **10** 3
[6] Elliott R J, Heap B R, Morgan D J and Rushbrooke G S 1960 *Phys. Rev. Lett.* **5** 366
[7] Domb C and Sykes M F 1961 *Phys. Rev.* **122** 77
[8] Frisch H L, Sonnenblick E, Vyssotsky V A and Hammersley J M 1961 *Phys. Rev.* **124** 1021
[9] Dean P 1963 *Proc. Camb. Phil. Soc.* **59** 397
[10] Sykes M F and Essam J W 1964 *Phys. Rev.* **133** A310
[11] Sykes M F, Gaunt D S and Glen M 1976 *J. Phys. A: Math. Gen.* **9** 97
[12] Roussenq J, Clerc J, Giraud G, Guyon E and Ottavi H 1976 *J. Physique* **37** L99
[13] Stauffer D 1976 *Z. Phys.* B **25** 391
[14] Hoshen J, Kopelman R and Monberg E M 1978 *J. Stat. Phys.* **19** 219
[15] Reynolds P, Stanley H E and Klein W 1980 *Phys. Rev.* B **21** 1223
[16] Derrida B and de Seze L 1982 *J. Physique* **43** 475
[17] Djordjevic Z V, Stanley H E and Margolina A 1982 *J. Phys. A: Math. Gen.* **15** L405
[18] Gebele T 1984 *J. Phys. A: Math. Gen.* **17** L51
[19] Rapaport D C 1985 *J. Phys. A: Math. Gen.* **18** L175
[20] Derrida B and Stauffer D 1985 *J. Physique* **46** 1623
[21] Kertész J 1986 *J. Phys. A: Math. Gen.* **19** 599