Lattice animals: A fast enumeration algorithm and new perimeter polynomials

Article *in* Journal of Statistical Physics · March 1990

DOI: 10.1007/BF01026565

CITATIONS
43

READS
252

1 author:

Stephan Mertens
Otto-von-Guericke-Universität Magdeburg
89 PUBLICATIONS **1,967** CITATIONS

SEE PROFILE

Journal of Statistical Physics, Vol. 58, Nos. 5/6, 1990

Lattice Animals: A Fast Enumeration Algorithm and New Perimeter Polynomials

S. Mertens[1]

Received August 21, 1989

A fast computer algorithm for enumerating isolated connected clusters on a regular lattice and its Fortran implementation are presented. New perimeter polynomials are calculated for the square, the triangular, the simple cubic, and the square lattice with next nearest neighbors.

KEY WORDS: Cluster enumeration; Fortran program; perimeter polynomials.

1. INTRODUCTION

The problem of determining the number of finite clusters on a regular lattice ("lattice animals" or "polyominoes") with given size s and perimeter t arises mainly in the context of series expansions in percolation problems.[1] If we denote this number (per lattice site) by g_{st}, the mean number per lattice site of clusters of size s in the classical site percolation problem is given through

$$\begin{aligned} \langle n_s \rangle &= \sum_t g_{st} p^s q^t \\ &=: p^s D_s(q) \end{aligned} \tag{1}$$

where $p = 1 - q$ denotes the probability that a single lattice site has the attribute that makes it a cluster site. D_s is usually called *perimeter polynomial.* $D_s(1) =: g_s$ gives the total number per lattice site of clusters of size s. Consider, for example, the clusters of size 3 on the triangular lattice: There

[1] Institut für Theoretische Physik, D-3400 Göttingen, Federal Republic of Germany.

0022-4715/90/0300-1095$06.00/0

are two of them with perimeter nine, and nine with perimeter ten, i.e., $D_3(q) = 2q^9 + 9q^{10}$.

The calculation of the g_{st} for large s quickly gets complicated enough to be better put on a computer. An algorithm for this cluster enumeration was presented by Martin[2] and Redner,[3] and perimeter polynomials for various lattices have been calculated with it.[5–11] The enumeration time, however, grows like the total number of clusters, i.e., exponential with s. This limits the maximum calculable size s very soon.

In this communication an algorithm is presented which calculates the perimeter polynomials faster than the algorithm of Martin. Although this algorithm, too, suffers from the exponential growth of CPU time with cluster size s, the gain in speed is sufficient to obtain new results even with little computer time. These new perimeter polynomials for the square, the triangular, and the simple cubic lattice and the square lattice with next nearest neighbors (abbreviated as nnSquare) are listed in Appendix A.

There are many things that can be calculated with the perimeter poly-

Table I. Total Number g_s of Clusters Grouped by Sites[a]

Sites	nnSquare	Triangular	Cubic
1	1	1	1
2	4	3	3
3	20	11	15
4	110	44	86
5	638	186	534
6	3832	814	3481
7	23592	3652	23502
8	147941	16689	162913
9	940982	77359	1152870
10	6053180	362671	8294738
11	39299408	1716033	60494549
12	**257105146**	8182213	446205905
13	**1692931066**	39267086	3322769321
14	**11208974860**	189492795	**24946773111**
15		918837374	**188625900446**
16		4474080844	
17		21866153748	
18		**107217298977**	
19		**527266673134**	

[a] The numbers for the simple square lattice up to $s = 24$ can be found in ref. 4. Bold numbers are new to us. Note that our last three digits of g_{13} on the cubic lattice disagree with Lam's.[13] Our value agrees, however, with that of ref. 8.

Table II. Coefficients[a] for the Expansion of $S(p) = \sum b_r p^r$

r	Square	nnSquare	Triangular	Cubic
1	4	8	6	6
2	12	32	18	30
3	24	108	48	114
4	52	348	126	438
5	108	1068	300	1542
6	224	3180	750	5754
7	412	9216	1686	19574
8	844	26452	4074	71958
9	1528	**73708**	8868	233574
10	3152	**206872**	20892	870666
11	5036	**563200**	44634	2696274
12	11984	**1555460**	103392	*10375770
13	15040	**4124568**	216348	**30198116**
14	46512		499908	**122634404**
15	34788		1017780	
16	197612		*2383596	
17	4036		**4648470**	
18	929368		**11271102**	
19	*−702592			
20	**4847552**			
21	**−7033956**			
22	**27903296**			
23	**−54403996**			

[a] Bold coefficients are new to us. b_{23} for the square lattice has been obtained using g_{24} of ref. 4. Note that b_{19}, b_{21}, and b_{23} of the square lattice are negative. The values marked with asterisks confirm those in ref. 9.

nomials, but since this paper is mainly concerned with the algorithm, only two quantities have been obtained from the new data: The total number of clusters of size s, g_s (Table I), and the coefficients of the series expansion of the mean size of clusters at low densities, $S(p) = p^{-1} \sum s^2 \langle n_s \rangle$ (Table II). The knowledge of the perimeter polynomials up to a size $s_{\max}$ allows the calculation of g_s up to $s = s_{\max} + 1$ and the series expansion of S up to order $s_{\max}$.[2]

2. THE ALGORITHM

In 1981 Redelmeier[4] calculated the total number of s-clusters on the simple quadratic lattice up to a size of 24. Although similar in the basic

[2] See, however, ref. 5 for how to calculate $g_{s_{\max}+2}$ and the series expansion up to order $s_{\max} + 1$ using graph-theoretic methods.

concept, his algorithm differs from that of Martin in some important details which make the enumeration faster. In what follows the algorithm of Martin is denoted algorithm M, the one of Redelmeier algorithm R.

Both algorithms generate all clusters up to a size s_{max} in a recursive manner, i.e., given any s-cluster, the algorithm builds all $(s+1)$-clusters by adding one new cluster site. If all possible $(s+1)$-clusters with this particular s-cluster "parent" are generated, a new s-cluster has to be found which again serves as a parent for a new generation $(s+1)$-clusters. This leads to depth-first traversal of a "family tree" of all clusters (to cite Redelmeier). Each child cluster in the tree consists of its parent plus one new cluster site. The choice of this new site underlies two restrictions which ensure that each cluster appears exactly once in the tree. The first one is due to the translational symmetry of the underlying lattice: Certain lattice sites have to be nonaccessible ("blocked") for any cluster site (see Fig. 1a), preventing the new cluster site from being placed there. The second restriction is the requirement that the new cluster site has to be chosen so that no older brother or ancestor's older brother in the tree contains it (Fig. 1b). As pointed out by Redelmeier, these two restrictions are sufficient to let every node in the tree be different from every other.

Algorithm M realizes the second restriction through a prohibition mechanism: Each time a site is removed from a current s-cluster, the corresponding lattice site becomes "s-prohibited," i.e., is marked nonaccessible for all of the cluster's children and younger brothers. After every possibility to create an s-cluster in this subtree has been explored, all s-prohibited sites are "freed," i.e., are made accessible again and control is passed to the $(s-1)$-cluster ancestor node in the family tree. This mechanism clearly fulfills the second restriction, but it requires extra bookkeeping about the number of s-prohibited sites and their locations and

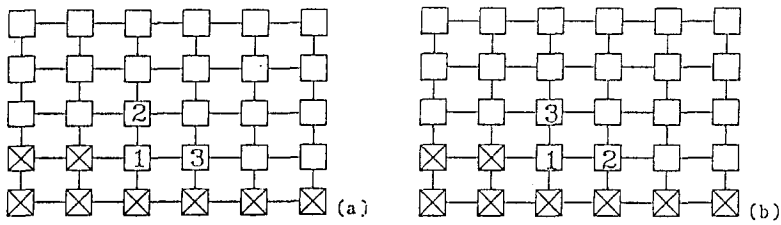

Fig. 1. The ×-sites of this square lattice are labeled "blocked" and the first site in the hierarchy of cluster-sites is the site labeled 1, the "origin." This forces the left most site of the bottom row of any cluster to be at the origin. The second restriction prevents an algorithm from generating the 3-cluster in (b) which differs from that in (a) only in its sequence of generated sites, indicated by the numbers 1, 2 and 3.

frequent modifications of lattice sites from "free" to "prohibited" and vice versa, which is very time consuming.

While algorithm M uses the information of where not to place subsequent cluster sites, algorithm R passes the information of where to place new cluster sites to each recursive invocation of the procedure. This "untried set" contains all points that are adjacent to the parent cluster and have not been used by the ancestors or the ancestors' older brothers. Algorithm R was originally designed for the enumeration of clusters on the square lattice, but the "untried set" ansatz is a very powerful mechanism to traverse the "family tree" on general lattices. It reduces the necessary bookkeeping by a decent amount, making the traversal of the tree much faster than the prohibition ansatz of algorithm M. Avoiding the explicit construction of clusters (see below), Algorithm R can be made even faster. The fast tree-traversing-capability of algorithm R is one reason why I borrow the "untried set" idea from Redelmeier. The other reason is that this technique facilitates the calculation of the perimeter, as is shown below.

The following routine is given such an untried set, the size s of the cluster to be generated, and the perimeter t of the parent $(s-1)$-cluster. The steps 1,..., 4 are then repeated until the untried set is exhausted. Each *iteration* generates a child of the parent and each *recursion* all of the offspring of that child.

1. Remove an element from the untried set.
2. Determine "free" and "blocked" neighbors of this point; $nn :=$ number of these new neighbor sites.
3. Count new cluster: Increase $g_{s,t+nn-1}$ by one.
4. If $s < s_{\max}$:
   (a) Add "free" new neighbors to the untried set and label corresponding lattice sites "reachable."
   (b) Call this routine recursively with the current untried set, $t := t + nn - 1$ and $s := s + 1$.
   (c) Remove new neighbors from the untried set and relabel corresponding lattice sites "free."

At any one time, each lattice site (excluding the "blocked" ones) is either "reachable" (i.e., an element of the current untried set or the untried set of an older brother) or "free." The algorithm is started with all lattice sites being "free" (besides the "blocked" sites and the "origin" in Fig. 1a, which is "reachable"), the untried set containing only the "origin," $t = 0$ and $s = 1$. The fact that in step 1 an element is taken from the untried set without labeling the lattice site "free" corresponds in a way to the "prohibition"

mechanism in algorithm M: Keeping this site "reachable" prevents it from being added to an untried set somewhere in the recursive offspring of that node.

As discussed by Redelmeier, the untried set can be coded as a linked list, which makes the necessary operations on it (adding and removing elements and passing it as a parameter) very effective and therefore very fast. The use of the untried set, moreover, has the advantage that the cluster-perimeter can be traced with almost no extra effort, since the new neighbors of an added cluster site (step 2) have to be determined anyway to update the untried set in step 4a. It should be noted that the above algorithm avoids the explicit construction of a cluster, i.e., labeling the lattice site just taken from the untried set in step 1 as "occupied" as in Redelmeier's original algorithm. This is not necessary, because the relevant information is wether a lattice site is accessible in the current node and not the reason for its nonaccessibility, i.e., whether it is an element of the current cluster or a former element of one of its older brothers.

A quantitative comparison of this algorithm with algorithm M in terms of CPU time will be given in the next section.

3. A FORTRAN IMPLEMENTATION

For the sake of simplicity a little complication has not been mentioned in the preceding section: The counting of "blocked" neighbors. To determine the correct perimeter, the algorithm has to ensure that no "blocked" site is counted more than once as a perimeter site. This problem is most easily dealt with in the square lattice, since here the only site in the "blocked" area that can be a neighbor of more than one cluster site is the left neighbor of the "origin." In this case, we start the algorithm with the 1-cluster instead of the 0-cluster and label the left neighbor of the origin "counted" since it has already contributed to the perimeter of the 1-cluster. For other lattices (such as the triangular lattice, for example), each "blocked" site which is adjacent to the "free" area can be a neighbor of two (or more) cluster sites. One is therefore forced to trace the "blocked" sites counted in step 2. This can be achieved by labeling them "counted" in step 4a and relabeling them "blocked" in step 4c.

A Fortran program for the calculation of perimeter polynomials in the square lattice is listed in Appendix B. The two-dimensional lattice is stored in the linear list **latt[1...nlatt]. nnn** denotes the number of nearest neighbors and **direct[1...nnn]** contains the adjacency vectors which give the nearest neighbors of any site, **nsmax** denotes the maximum cluster size to be generated and **maxt** the maximum perimeter that can occur. The perimeter of the current s-cluster is stored in **avail(s)**, the coefficients of the

perimeter polynomials in the linear list **g**. The variables **succ**, **first**, and **newfi** are used for coding the untried set as a linked list; see ref. 4 for a detailed description of this method. The rest of the variables are mainly dummy variables to avoid time-consuming operations on array variables or offset variables used to code g_{st} as a linear list.

The initialization of the lattice, the 1-cluster, and its untried set takes place between lines 12 and 34. In lines 35...39 an element is removed from the untried set, in lines 40...48 the neighborhood of this site is explored and the free neighbors are added to the untried set. The new cluster is counted in line 50. If the current cluster size s is smaller than $s_{max}-1$, the procedure is called recursively (lines 52...57). Otherwise, only the number of neighbor sites of each of the possible s_{max}-clusters is calculated (lines 59...70). The recently added neighbors are removed from the untried set in lines 72...76. If there are still elements in the current untried set, the procedure is iterated (line 77), otherwise it returns to the parent node (lines 78...81) or finally shows the results and stops.

The presented Fortran program can be made faster at the price of more lines of code. For example, in the loop between lines 41 and 48 the loop variable m always runs from 1 to 4 (the number of nearest neighbors). A replacement of this loop with four replicas of the loop's body saves a lot of index calculations, since **direction(m)** can be replaced by **direction(1)... direction(4)** and the index calculation is done at compile rather than at run time. Another means to improve the performance is provided by the fact that only one site of the four neighbors of a given cluster site can eventually be in the "blocked" area. Therefore the detailed test in line 42 can be replaced by a simpler (and faster) one in three of four cases. For the quadratic lattice, this modifications improve the performance by about 24% (see Table III); for other lattice types the gain in speed is even higher, especially for lattices with high coordination numbers.

In Table III, the performance of the presented Fortran program (and its "tuned" version as described above) is compared to Redner's implementation of algorithm M. It can be seen that Redner's program is about 46% slower than the fast version of the presented Fortran program despite the fact that Redner's program only calculates the total number of s-clusters. Actually, this should make it faster than any calculation of the full perimeter polynomials. Demme and Diemer[12] presented a modification of algorithm M which enumerates clusters about λ times faster than the original implementation of Redner, where $\lambda \approx$ coordination number of the lattice. They used the fact that it suffices to explore the "family tree" down to the nodes with $s = s_{max}-1$ if one introduces an additional bookkeeping about the accessible perimeter sites of a cluster. This corresponds to the fact that the total number of clusters of size s can be calculated from the

Table III. Total CPU Time (in sec) on an Apollo DN 4500 Workstation for the Enumeration of Lattice Animals on the Square Lattice[a]

s_{max}	Algorithm M (program of ref. 3)	Presented algorithm	
		Program as listed	Fast version
12	5.4	4.9	3.8
13	20.3	18.4	14.0
14	79.1	68.6	54.5
15	291.7	262.7	199.2

[a] An IBM 3090 is about three times faster, but gives about the same speed ratios. Notice that algorithm M only enumerates the total numbers of s-clusters, while the algorithm presented here also gives their perimeter.

perimeter polynomial with $s-1$. The performance of the program of Demme and Diemer for s_{max} therefore has to be compared with the performance of the presented program for $s_{max}-1$. It turns out that the Demme and Diemer version of Algorithm M is about 68% slower (for the square lattice) than the fast version of our program and our algorithm could be made even faster for the simpler task of calculating the total number of clusters.

APPENDIX A. NEW PERIMETER POLYNOMIALS

In this Appendix, perimeter polynomials for various lattice types (square, triangular, cubic, square with next nearest neighbors) are presented which seem new (Tables IVA–IVD). The used CPU time as background processes on an Apollo DN 3500 or 4500 workstation ranges from about 30 h for the square lattice with next nearest neighbors ($s_{max}=13$) up to about 1 month for the square lattice ($s_{max}=22$). The g_{st} for smaller values of s can be found in ref. 5 (square), ref. 11 (nnsquare), refs. 5 and 10 (triangular), and ref. 6 (cubic).

Duarte[14] calculated (among other things) all g_{st} for $t \leqslant 16$ on the square lattice. His values are confirmed.

Table IVA. New Perimeter Polynomials for Square Lattice

t	$s=18$	$s=19$	$s=20$	$s=21$	$s=22$
14	4	0	0	0	0
15	396	124	28	4	0
16	8146	3982	1730	651	206
17	77042	49820	29263	15664	7632
18	498510	386626	277540	184792	114170
19	2375948	2185492	1842286	1442972	1058218
20	8892252	9568542	9371179	8467282	7137662
21	26424552	33581728	38089751	39444812	37849142
22	63570106	95790204	126608106	150367840	163610644
23	124322284	224749652	348128020	476549024	589572902
24	198771190	435951906	799099308	1268641853	1792118418
25	260020876	701526660	1537756259	2855017064	4634111086
26	278241194	937190080	2487219956	5453409264	10245773246
27	242759710	1038528312	3382206302	8855133452	19431734658
28	171725416	952066016	3863223002	12226523903	31651808512
29	97636026	718391056	3696468187	14336402380	44276317808
30	44239618	443166310	2950845400	14242603046	53124561334
31	15780916	221576912	1953171286	11942529948	54546106532
32	4382132	88988542	1064426530	8409232029	47767546994
33	928608	28363508	473427096	4940061004	35515036016
34	147426	7090716	170289758	2403638058	22294159292
35	16792	1362772	48953476	960196876	11737001634
36	1332	197494	11108759	312084862	5144218940
37	64	20636	1950048	81552556	1861029946
38	2	1516	259508	16916512	550671010
39	0	68	25096	2729340	131694496
40	0	2	1712	335497	25121626
41	0	0	72	30100	3745232
42	0	0	2	1920	427292
43	0	0	0	76	35808
44	0	0	0	2	2140
45	0	0	0	0	80
46	0	0	0	0	2

Table IV B. New Perimeter Polynomials for Square Lattice with Next Nearest Neighbors

t	$s=11$	$s=12$	$s=13$
18	8	2	0
19	16	0	0
20	298	151	68
21	972	524	192
22	3768	2486	1554
23	12076	9580	6796
24	33442	30739	23701
25	81668	84477	77928
26	185898	223164	230904
27	374564	518884	596864
28	703094	1104476	1461950
29	1179252	2180318	3246404
30	1867098	3976698	6661962
31	2653520	6601780	12785416
32	3500572	10362729	22810042
33	4253768	14867879	37555092
34	4741066	19827396	58372070
35	4790232	24605800	84302484
36	4481228	28261864	113533120
37	3747168	29863552	142979396
38	2842960	29266646	168351436
39	1878600	26269734	184110852
40	1095812	21495975	187495542
41	543908	15838460	176766448
42	230056	10522966	154250074
43	75480	6118796	123080096
44	18998	3135682	89906050
45	3396	1365574	58994028
46	452	494560	34776132
47	36	138764	17998340
48	2	30132	8167838
49	0	4756	3119288
50	0	560	976976
51	0	40	239116
52	0	2	45753
53	0	0	6428
54	0	0	680
55	0	0	44
56	0	0	2

Table IVC. New Perimeter Polynomials for Triangular Lattice

t	$s=16$	$s=17$	$s=18$
17	6	0	0
18	290	87	14
19	3147	1458	613
20	21924	13074	6864
21	117632	81606	52419
22	514503	411546	305656
23	1920135	1743216	1446105
24	6259778	6350256	5974463
25	17891511	20551044	21301341
26	45442314	58642875	68191791
27	102050537	150320514	194840911
28	203225319	343641300	501155544
29	355865085	702686730	1164695703
30	545610411	1280881581	2432848553
31	723205227	2066045316	4578711889
32	813752322	2934136986	7712998128
33	757005387	3618901248	11567355829
34	550789344	3804395967	15337518381
35	277984614	3310771548	17724834867
36	72421358	2250460905	17506888220
37	0	1059161730	14319930851
38	0	256954761	9135628605
39	0	0	4028224110
40	0	0	914388120

Table IV D. New Perimeter Polynomials for Simple Cubic Lattice

t	$s = 12$	$s = 13$	$s = 14$
26	9	0	0
27	0	0	0
28	432	48	0
29	4668	132	0
30	25440	3673	412
31	138904	25568	1908
32	620231	146086	24378
33	2097936	729428	159144
34	5926745	2907755	839738
35	13865948	9634686	3801489
36	27402345	26792718	14333094
37	45460473	64142668	45462840
38	63712706	131866119	125799096
39	75644082	233643764	303030330
40	75589074	356875730	641449368
41	62963158	470803212	1189701924
42	42141124	536248708	1937592309
43	21088314	523333552	2772958188
44	7408509	432412758	3483707502
45	1785240	294422852	3833369250
46	294660	156471102	3665727768
47	33264	61279704	3009350772
48	2520	17123236	2075360802
49	120	3383940	1153919334
50	3	472431	492225102
51	0	46220	155424804
52	0	3096	35770806
53	0	132	5974992
54	0	3	721578
55	0	0	62304
56	0	0	3732
57	0	0	144
58	0	0	3

APPENDIX B. A FORTRAN PROGRAM FOR THE SQUARE LATTICE

```
      PARAMETER(nmax=12,nnn=4,
     +maxt=2*nmax+2,nsize=nmax*maxt,
     +nlatt=2*nmax*(nmax+2)+1)
      INTEGER free,block,reach,
     +count,g(nsize),latt(nlatt),
     +succ(nlatt),direct(nnn),
     +avail(nmax),first(nmax),
     +newfi(nmax),s,t,test,
     +goff,offset
      PARAMETER(count=-2,reach=-1,
     +free=0,block=1)
      DATA latt/nlatt*free/,
     +succ/nlatt*0/,g/nsize*0/
      direct(1) = -1
      direct(2) = 2*nmax
      direct(3) = 1
      direct(4) = -2*nmax
      norigi = 3*nmax
      DO 10,i=1,norigi-1
10       latt(i)=block
      avail(1) = 1
      first(1) = norigi
      latt(norigi) = reach
      avail(2) = nnn
      latt(norigi-1) = count
      succ(norigi+2*nmax) = 0
      latt(norigi+2*nmax) = reach
      succ(norigi+1) = norigi+2*nmax
      latt(norigi+1) = reach
      first(2) = norigi+1
      g(nnn) = 1
      s = 2
      maxoff = (nmax-1)*maxt
      goff = maxt
1000  IF (first(s).EQ.0) GO TO 1110
1010  now = first(s)
      locfir = succ(now)
      newfi(s) = locfir
      newper = avail(s) - 1
      m=1
1020  test = now + direct(m)
      IF (latt(test)) 1050,1030,1040
1030  succ(test) = locfir
      locfir = test
      latt(test) = reach
1040  newper = newper+1
1050  m = m+1
      IF (m.LE.nnn) GO TO 1020
      offset = goff + newper
      g(offset) = g(offset)+1
      IF (s.LT.nmax-1) THEN
        first(s) = locfir
        s = s+1
        goff = goff + maxt
        avail(s) = newper
        first(s) = locfir
        GO TO 1000
      ELSE
        maxnt = newper - 1
        maxnfi = locfir
1060    IF (maxnfi.EQ.0) GO TO 1080
        now = maxnfi
        maxnfi = succ(now)
        maxntn = maxnt
        DO 1070,m=1,nnn
        test = now + direct(m)
1070    IF(latt(test).GE.free)maxntn=maxntn+1
        g(maxoff+maxntn) = g(maxoff+maxntn)+1
        GO TO 1060
1080    CONTINUE
      ENDIF
1090  IF (locfir.EQ.newfi(s)) GO TO 1100
      latt(locfir) = free
      locfir = succ(locfir)
      GO TO 1090
1100  first(s) = locfir
      IF (locfir.NE.0) GOTO 1010
1110  s = s-1
      goff = goff - maxt
      locfir = first(s)
      IF (s.GT.1) GO TO 1090
      DO 2000,t=1,maxt
2000    PRINT 2010,(g((s-1)*maxt+t),s=1,nmax)
2010  FORMAT(100I10)
      END
```

ACKNOWLEDGMENTS

I thank D. Stauffer and J. A. M. S. Duarte for constructive suggestions on the manuscript and U. Schulz for his help with the computer system. All calculation were performed on the Apollo DN 3500/4500 workstations of the Institut für Theoretische Physik der Universität Göttingen.

REFERENCES

1. D. Stauffer, *Introduction to Percolation Theory* (Taylor & Francis, London, 1985), and references therein.
2. J. L. Martin, in *Phase Transitions and Critical Phenomena*, Vol. 3, C. Domb and M. S. Green, eds. (Academic Press, New York, 1974), pp. 97–112.
3. S. Redner, *J. Stat. Phys.* **29**:309 (1982).
4. D. H. Redelmeier, *Discr. Math.* **36**:191 (1981).
5. M. F. Sykes and M. Glen, *J. Phys. A: Math. Gen.* **9**:87 (1976).
6. M. F. Sykes, D. S. Gaunt, and M. Glen, *J. Phys. A: Math. Gen.* **10**:1705 (1976).
7. D. S. Gaunt, M. F. Sykes, and H. Ruskin, *J. Phys. A: Math. Gen.* **9**:1899 (1976).
8. M. F. Sykes and M. K. Wilkinson, *J. Phys. A: Math. Gen.* **19**:3407 (1986).
9. M. F. Sykes and M. K. Wilkinson, *J. Phys. A: Math. Gen.* **19**:3415 (1986).
10. A. Margolina, Z. V. Djordjevic, D. Stauffer, and H. E. Stanley, *Phys. Rev. B* **28**:1652 (1983).
11. H. R. Peters, D. Stauffer, H. P. Hölters, and K. Loewenich, *Z. Physik B* **34**:399 (1979).
12. E. S. Demme and K. Diemer, *J. Undergrad. Res. Phys.* **3**:25 (1984).
13. P. M. Lam, *Phys. Rev. A* **34**:2339 (1986).
14. J. A. M. S. Duarte, *Portgal. Phys.* **12**:99 (1981).