Journal of Statistical Physics, Vol. 29, No. 2, 1982

A FORTRAN Program for Cluster Enumeration

S. Redner[1]

Received May 11, 1982

A simple FORTRAN program based primarily on the algorithm of Martin is presented for enumerating isolated connected clusters of up to a given specified number of particles on the square lattice. A brief explanation of the workings of the program is also provided to facilitate its use by those interested in this problem. To order 15, the program enumerates clusters at an average rate of approximately 63,000/sec when implemented on an IBM 370/168 with the extended optimizing compiler.

KEY WORDS: Cluster enumeration; Martin algorithm; FORTRAN program.

Over the past decade, there has been extensive work in applying series methods to cluster statistical problems such as percolation and lattice animals.[1] Many of the series calculations that have been performed are based on cluster enumeration, a problem of classic difficulty that was first posed in the context of graph theory.[2] The goal of the enumeration is to count the number of suitably normalized connected clusters of sites or bonds on a regular lattice. Several groups have developed computer programs to accomplish this enumeration and have applied their techniques to a wide variety of problems.[3–8] However, as yet, a listing of such a program has not been published, although a fairly detailed discussion of the algorithm underlying one particular enumeration strategy has appeared in an article by Martin[9] in the Domb and Green series.

In this paper, we provide a FORTRAN program for cluster enumeration which is based primarily on Martin's algorithm. (We assume that the

[1] Center for Polymer Studies[2] and Department of Physics, Boston University, Boston, Massachusetts 02215.

[2] Supported in part by grants from the ARO, NSF, and ONR.

0022-4715/82/1000-0309$03.00/0

reader has some familiarity with that publication in what follows.) The program is short and simple, being only 36 lines long, and it is presented in an Appendix to serve as a useful resource to those interested in learning about, or using enumeration methods.

For simplicity, the program given is for the enumeration of site clusters, defined as a group of nearest-neighbor occupied sites, on the square lattice. This program is sufficiently simple and flexible so that it can be easily generalized to treat bond or site enumeration on any regular lattice, as well as certain more specialized classes of enumeration problems. The key to the Martin algorithm is associating a unique labeling to each n-site cluster which is generated in a recursive way. This ensures that each cluster is counted only once, due to the uniqueness of the labeling, and that all possible clusters are generated, because of the recursive labeling. This labeling scheme will be evident in our program, as we shall explain below.

To begin, we define some of the quantities and arrays used in the program. For convenience, the entire lattice is stored in the one-dimensional array, termed *iocc*. The dimension of this array determines the maximum cluster size that can fit on the lattice without wraparound problems (see Fig. 1). With the dimensions specified for all the arrays in line 1 of the program, clusters of up to 20 sites can be accommodated. The location of each lattice site that is still available for becoming part of the cluster is given the value 0 in *iocc*, while the location of a nonavailable site is given the value 1. There are several ways that nonavailability of a site can occur, and we will discuss this point in more detail below.

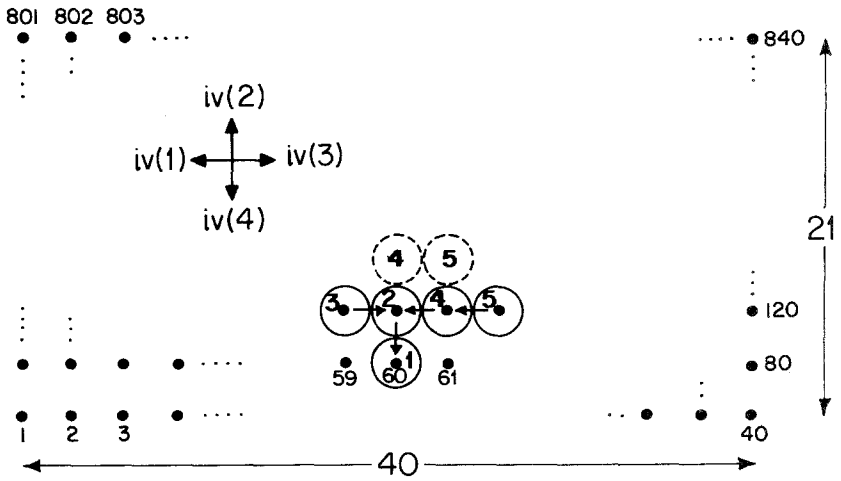

Fig. 1. Schematic picture of the lattice, and a typical cluster at some intermediate stage of the enumeration is shown. Full circles are the cluster sites, while dotted circles are prohibited sites whose order is written. The arrows indicate the root to which each cluster site is attached.

The geometry of the lattice is defined through the adjacency vectors, *iv*, which give the nearest neighbors of any site. By changing these adjacency vectors, and by increasing the dimensions of certain arrays as needed, the program can be modified to enumerate site or bond clusters on any lattice. For the values for *iv* given in line 2, the lattice is a 40×21 rectangular grid. In parallel with the cluster storage mode described above, the locations of the cluster sites are also stored consecutively, in the array *ip*. This will provide the sequential labeling scheme required by the Martin algorithm. The *n*th cluster site joins to the rest of the cluster by being attached to a particular "root" site already in the cluster whose order is between 1 and $n - 1$. The value of $jpt(n)$ gives the order of this root site to which the *n*th site is attached.

As clusters are successively built, the array *numb* is updated to record the completion of each successful cluster. However, once a cluster of a maximum predetermined size (*nmax*) is constructed, it then becomes necessary to remove cluster sites intermittently, in the reverse order that the sites were added, so that the enumeration proceeds to completion. This "backtracking" procedure is crucial to the success of the program, and is explained below.

Once a *j*th-order site is removed, the location of that site becomes "*j* prohibited" (following Martin's nomenclature). This condition means that such a location becomes unavailable for the addition of another cluster site until a later stage is reached in the enumeration where the *j*-prohibited sites are "freed." Such a restriction is required so that the same cluster is not generated from a different labeling of the sites. These prohibited sites are stored in *jp*; the second argument of the array gives the order at which the given site is prohibited, while the first argument gives the number of prohibited sites of that order. This number, in turn, is stored in *np*. Thus the prohibited sites of each order are stored sequentially. However, in testing whether a particular site is prohibited, it is faster to store only the locations of the prohibited sites. This is done in the array *iocc*, where the location of a prohibited site is given the value 1, while a nonprohibited site is given the value 0. This use of *iocc* is in addition to its use in storing occupied cluster sites as well.

The program begins with a single occupied site in the middle of the second row of the lattice, site #60 as specified in the data statement (line 2). Beginning with this site, we add to it $iv(i)$, with i incrementing sequentially between 1 and 4 to locate a nearest-neighbor site. More generally, the program finds the nearest neighbors of the current root, $ip(j)$, in an $(n - 1)$-site cluster (line 5). The new location, *inow*, will become the next site in the cluster if three conditions are satisfied. These conditions are tested by the statement in line 6, where the program tests whether *inow* is available for occupancy by a new cluster site. This includes the possibilities

that *inow* is already occupied by (i) an earlier cluster site, (ii) by an already prohibited site, or (iii) the location of the current site is "earlier" than the origin (site #60). The current site must be "later" than the origin so that the same cluster is not generated more than once with only the overall order of the cluster labeling reversed. The last test is most conveniently performed by initializing $iocc(i)$ to 1 for all i between 1 and 60, thereby making these "earlier" sites unavailable for becoming cluster sites.

If *inow* successfully passes these checks, then a valid additional cluster site has been found. Control is transferred to line 9 where the value *inow* is stored in $ip(n)$, $numb(n)$ is incremented, and $iocc(inow)$ is set equal to 1. The last statement means that *inow* becomes part of the cluster, and that its location is no longer available for occupation by later cluster sites. This is the appropriate stage (after line 12) where one may measure the cluster perimeter, radius, or other configurational properties of interest. For these measurements, it may be desirable to differentiate between cluster sites and prohibited sites in the array *iocc*. This may be accounted for by adding the two statements indicated in the program. With these additions, the locations of cluster sites are given the value 1 in *iocc*, while the locations of prohibited sites are given the value 2.

If n is less than *nmax*, the order j of the root site to which *inow* is attached is also stored (line 14), n is incremented, and the program then attempts to add another site to the cluster. To do this, the program goes to line 19 and checks whether the four possible nearest neighbors of the current root site have been utilized previously. If this is the case, then it is no longer possible to attach an additional site to the current root. Therefore, a new root is chosen, by incrementing the site label for the root by 1 (line 20), and the program will attempt to add sites to the updated root.

If n has reached *nmax*, then it is necessary to add the location of the last cluster site to the list of prohibited sites of order n (lines 17 and 18). Since a prohibited site is created where the cluster site was removed, this particular location is still not available for later occupancy by other cluster sites. We can account for this fact most simply by doing nothing. In this way, the locations of occupied cluster sites and prohibited sites may be stored simultaneously in *iocc*. At this stage, the program has reached line 19 where it proceeds as described above.

On the other hand, it may occur that *inow* is not a valid additional cluster site according to the test in line 6. If this happens, control is transferred to line 20 where a new root is chosen so that cluster building may begin once again on this new root. However, if all cluster sites have been used as roots, it is then necessary to implement the "backtracking" procedure which performs the intermittent prohibition of certain lattice sites necessary to ensure the unique cluster labeling. All prohibited sites of

order n are freed for possible later occupancy by additional cluster sites in lines 23–27. The cluster site of order $n-1$ should also be removed and a prohibited site of order $n-1$ should be created. As explained above, this latter task is accomplished by doing nothing in the array *iocc*. However, it is necessary to store the prohibited site of order $n-1$ in *jp* (lines 29 and 30). Finally, the root of the last site in the cluster is identified (line 31) and cluster building begins once again on this root.

When the backtracking procedure reaches the first site of the cluster, then all possibilities for root and cluster sites have been exhausted (line 22). At this point, the program is finished and the results are printed.

The program can be run as displayed using a standard FORTRAN compiler on the IBM 370/168 machine at Boston University with the VPS operating system. With this compiler all array and variable space is initialized to zero automatically. However, the program runs considerably faster using a FORTRAN extended optimizing compiler, in which case it is necessary for the user to initialize the arrays properly at the outset of the program. This may be accomplished by zeroing all but the first 60 elements of *iocc*, and all but the first elements of *ip* and *numb*. With the extended compiler, the program requires approximately 11.1, 40.4, 155.3, and 587.0 sec of cpu time to enumerate all clusters of up to size 12, 13, 14, and 15, respectively. This represents a cluster counting rate of approximately 63,000 per second. This counting rate may not be the best possible, but it is sufficient to obtain series of meaningful length for a wide variety of problems.

ACKNOWLEDGMENTS

This paper would not have been written were it not for the friendly persuasion of D. Stauffer. I am grateful to him for many constructive suggestions on both the computer program and the manuscript itself. I also thank H. E. Stanley for a critical reading of the manuscript and pertinent suggestions, and J. A. M. S. Duarte and J. Rogiers for helpful correspondence.

APPENDIX: COMPUTER PROGRAM FOR CLUSTER ENUMERATION

The two lines indicated as insertions are needed if it is desired to distinguish between cluster sites and prohibited sites (see text). The output of this program is the number of connected n-site clusters on the square lattice with $1 \leqslant n \leqslant nmax$. (These are given to order 19 in Table 1 of Ref. 4.) For completeness, these terms are: 1, 2, 6, 19, 63, 216, 760, 2725, 9910,

36446, 135268, 505861, 1903890, 7204874, 27394666, 104592937, 400795844, 1540820542, and 5940738676.

```
1       dimension iv(4),ip(20),jpt(20),np(20),jp(20,20),iocc(840),
        numb(20)
2       data nmax,numb(1),j,n,ip(1),iv,iocc/11,1,1,2,60,-1,40,1,
        -40,60*1/
3    10 k = 1
4     1 do 2 i = k,4
5       inow = ip(j) + iv(i)
6       if (iocc(inow).eq.0) go to 3
7     2 continue
8       go to 4
9     3 ip(n) = inow
10      k = i + 1
11      numb(n) = numb(n) + 1
12      iocc(inow) = 1
13      if (n.eq.nmax) go to 6
14      jpt(n) = j
15      n = n + 1
16      go to 5
17    6 np(n) = np(n) + 1
18      jp(np(n),n) = inow                    < iocc(inow) = 2
19    5 if (k.lt.5) go to 1
20    4 j = j + 1
21      if (j.lt.n) go to 10
22      if (n.eq.2) go to 9
23      nx = np(n)
24      if (nx.eq.0) go to 7
25      np(n) = 0
26      do 8 m = 1,nx
27    8 iocc(jp(m,n)) = 0
28    7 n = n - 1
29      np(n) = np(n) + 1                     < iocc(ip(n)) = 2
30      jp(np(n),n) = ip(n)
31      j = jpt(n)
32      go to 10
33    9 write (6,100) (numb(m),m = 1,nmax)
34  100 format (6i12)
35      stop
36      end
```

REFERENCES

1. For extensive references on these fields, see e.g., D. Stauffer, *Phys. Rep.* **54**:1 (1979); J. W. Essam, *Rep. Prog. Phys.* **43**:833 (1980).
2. See, e.g., F. Harary, in *Graph Theory and Theoretical Physics*, F. Harary, ed., Academic Press, London (1967), pp. 1–41; D. A. Klarner, *Can. J. Math.* **19**:851 (1967).
3. C. Domb, *Philos. Mag. Suppl.* **9**:301 (1960).
4. M. F. Sykes and M. Glen, *J. Phys. A: Gen. Phys.* **9**:87 (1976).
5. M. F. Sykes, D. S. Gaunt, and M. Glen, *J. Phys. A: Gen. Phys.* **9**:1705 (1976).
6. D. S. Gaunt and H. Ruskin, *J. Phys. A: Gen. Phys.* **11**:1369 (1978).
7. H. P. Peters, D. Stauffer, H. P. Hölters, and K. Loewenich, *Z. Phys. B* **34**:399 (1979).
8. J. A. M. S. Duarte and H. Ruskin, *Physica* **111A**:423 (1982).
9. J. L. Martin, in *Phase Transitions and Critical Phenomena*, Vol. 3, C. Domb and M. S. Green, eds., Academic Press, New York (1972), p. 97.