BRAC University Journal, Vol. II, No. 2, 2005, pp. 103-106

A FAST TREE BASED CLUSTER LABELING ALGORITHM FOR MONTE CARLO SIMULATIONS

Marzuk M Kamal
Department of Physics
Bangladesh University of Engineering & Technology
Dhaka– 1000, Bangladesh

and

M. Nasimul Haque
Department of Mathematics & Natural Science
BRAC University, 66 Mohakhali C/A
Dhaka – 1212, Bangladesh

ABSTRACT

A highly efficient tree based algorithm for studying site or bond percolation on any lattice system is described. Our approach is to identify the connectivity of the lattice sites in a single phase and to reduce the redundant computational load in each lattice update. Efficiency increases due to the creation of a multi-branched tree of the pointers of the cluster numbers at the time of investigation of cluster organization. At the later updates, the computational efficiency increases further as the algorithm would have to work only on the randomly chosen lattice sites or bonds instead of traversing the entire lattice.

Key words: Percolation, Cluster labeling algorithm, Monte Carlo method

I. INTRODUCTION

The concept of percolation is very simple, each site on a specified lattice is independently either "occupied," with probability p, or not with probability $1 - p$. The occupied sites form contiguous clusters which have some interesting properties. If the value of p is too small then few sites are occupied and in most of the cases the occupied sites are far away from each other forming many clusters. As we increase the value of p, these disconnected clusters grow bigger and sometimes two separate clusters touch each other merging to one cluster. We call a system "percolates", if a single cluster is big enough to touch both of the opposite ends of the lattice system. It is observed that cluster becomes spanned above some value of p and we call it percolation threshold, p_c. This is a second order or continuous phase transition between "*not spanned*" and "*spanned*" states. The value of p_c depends on the topology of the lattice. [1]

One can also consider bond percolation in which the bonds of the lattice are occupied with probability p, and this system shows behavior qualitatively similar to though different in some details from site percolation.

Percolation phase transition is quite easily understandable. But, still there are many things about it that are still not known. For example, despite decades of effort, no exact solution of the site percolation problem yet exists on the simplest two-dimensional lattice and on any higher dimensional lattice. As there is no exact/theoretical solution of higher dimensional percolation problems, numerical simulation plays a very important role for studying. Because of these and many other gaps in our current understanding of percolation, numerical simulations have found wide use in the field.

Site and bond percolation have found a huge variety of uses in many fields. Percolation models appeared originally in studies of actual percolation in materials percolation of fluids through rock for example but have since been used in studies of many other systems, including granular materials, composite materials, polymers, concrete, aero gel

and other porous media, and many others. Percolation also finds uses outside of physics, where it has been employed to model resistor networks, forest fires and other ecological disturbances, epidemics, robustness of the Internet and other networks, biological evolution, and social influence, amongst other things.[2-7]

II. THE ALGORITHM

Before discussing our algorithm we would like to talk about the standard algorithms for percolation. In standard methods, the whole new state of the lattice must be recreated for every different lattice size n and all the clusters have to be identified. Various authors have pointed out [9-12], however, if we want to generate states for each value of n then we can save ourselves some effort by noticing that a correct sample state with n+1 occupied sites or bonds is given by adding one extra randomly chosen site or bond to a correct sample state with n sites or bonds. In other words, we can create an entire set of correct percolation states by adding sites or bonds one by one to the lattice, starting with an empty lattice. If we randomly change some lattice sites than the standard methods suggests that we have to traverse through the entire lattice to identify the newly formed clusters. For a small sized lattice this is not a big problem. Algorithms like Hoshen-Kopelman [8] can easily handle this problem. But applying the same algorithm for very big lattice would consume painfully long time especially when the algorithm is applied in Monte Carlo simulations.

In our algorithm we have constructed a matrix of pointers of the cluster values the time of initialization of the lattice. This pointer are connected by a multi branched tree which contains all the information of the cluster system. At the first phase we identify the clusters and construct the tree to maintain the connectivity of the improperly labeled clusters. We also maintain the cluster sizes in another array which is related to the tree. The root of the tree contains all the information of the cluster. The main essence of the algorithm could be found after randomly changing the lattice sites. As we have already constructed a tree of the lattice sites for keeping track of the clusters, now we only need to readjust the tree node for the lattice site to be updated.

For convenience we would like to describe our algorithm for bond percolation. This algorithm could also be applied to site percolation with little modification. To start with we have to construct a lattice with some occupied sites. We consider it more convenient if we just construct the lattice with some random probability. This may be done as follows:

1. Pick a random number within [0, 1]. If this random number is greater than another random number within [0, 1], then proceed to step 2 else jump to step 3.
2. Pick an integer random number and test whether it is odd or even. If odd then occupy the bond else leave unoccupied.
3. Repeat until all the bonds are traversed.

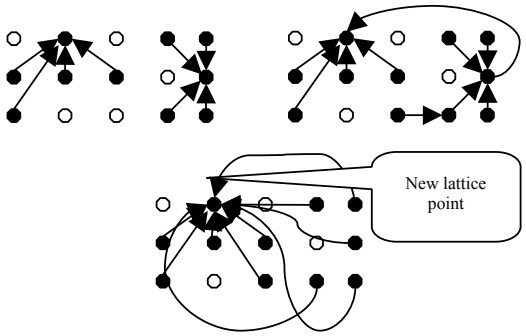

Figure 1: Initially presence of two different clusters is observed. All the sites of each cluster points to their respective root. If we occupy another bond which may belong to any of the clusters, the new occupied bond/site actually merges two different clusters into one. As clusters are merged the sites must look for a common root as which is shown in the third figure.

This lattice creation process consumes CPU time of O(N).

Now we have the lattice to work on. In order to identify the clusters we first assume that each occupied site is an independent cluster and we try to connect these clusters on the basis of nearest neighbor position. And a data tree is constructed which contains the size and connectivity information of each cluster. The flow of our algorithm is as follows:

1. Each occupied site is treated as independent cluster. If no nearest neighbors are occupied give this site a new cluster number. Insert this information in the data tree.
2. If there are any nearest neighbors check for there roots. If the roots are same then just increase the size of the lattice by one.
3. If roots are different then the root of the smaller cluster points to the root of the

larger cluster. If clusters are of same size just set the root of one cluster point to the root of the other cluster.

4. Repeat until the entire lattice is traversed.
5. Compress the path of the data tree of each cluster just to reduce working overload in future calculation.

The algorithm is simple enough to get confused about its efficiency. In fact, the algorithm is very fast when implemented as the cluster update occurs in a single phase. In the next section we have shown some results obtained from applying the algorithm.

III. RESULT

As we have mentioned earlier that few sites or bonds are occupied for low p and most of the sites are occupied for high value of p_c. For small p few sites need to be relabeled after identification of clusters where as for high p number of cluster will reduce and will become only one cluster for p=1. Therefore, number of relabeling will be small in the above mentioned situation. But near p_c number of clusters is large. Therefore we need to perform many relabeling after cluster identification, amalgamation and separation. As we can see in Figure 2 that cluster relabeling number becomes high for some value of p_c. For 2D square lattice the percolation threshold p_c = ½. And our algorithm when applied to 32x32 lattice averaged over 10000 lattice updates shows a pick very close to p_c = ½. Compared to standard algorithms our algorithm requires much less relabeling.

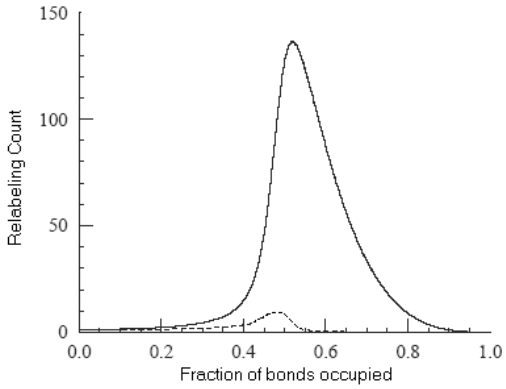

Figure 2: Number of relabeling required for different values of p in a 32x32 lattice.

The speed of the algorithm is compared to other standard cluster labeling algorithms. We executed both algorithms in the same computer to have a relative estimation of the speed our algorithm. We see (Fig 3) the computational time increases linearly with the system size and slope of the curve corresponding the tree based algorithm is smaller that the Hoshen-Kopelman algorithm.

VI. CONCLUSION

The aim of the paper was to explain the main idea of our tree based cluster labeling algorithm. We have seen that the algorithm works quite better that the standard algorithm. Here, we have compared the algorithm with Hoshen-Kopelman algorithm in the same computer. From the results obtained from different papers we are convinced that our tree base algorithm, when applied with Monte Carlo Simulations, works at efficiency level comparable to the fastest available algorithm.

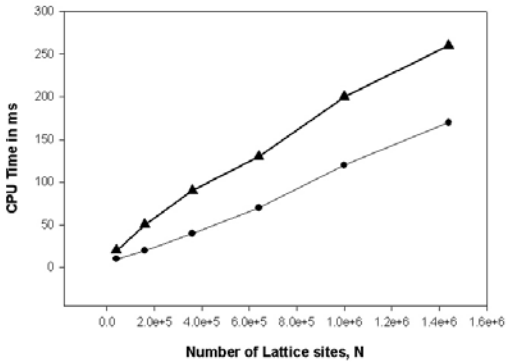

Figure 3: Time taken for a single update of L = 100, 200, 400, 600, 800, 1000 & 1200 systems. Here $N = L$xL. It is observed from the above graph, the computational time taken by our tree based algorithm increases almost linearly with the lattice size. The triangular points show the corresponding time taken by Hoshen-Kopelman algorithm.

REFERENCES

1. D Stauffer and A Aharony, "*Introduction to Percolation Theory*", Second Edition, Taylor & Francis, London (1992)
2. S. de Bondt, L. Froyen, and A. Deruyttere, "Electrical conductivity of composites: A percolation approach," *J. Mater. Sci.* 27, 1983-1988 (1992).
3. J. Machta, "Phase transitions in fractal porous media," *Phys. Rev. Lett.* 66, 169-172 (1991).

4. K. Moon and S.M. Girvin, "Critical behavior of superfluid ^{4}He in aerogel," *Phys. Rev. Lett.* 75, 1328-1331 (1995).

5. C.L. Henley, "Statics of a self-organized percolation model," *Phys. Rev. Lett.* 71, 2741-4728 (1993).

6. C.Moore and M.E.J. newman, "Exact solution of site and bond percolation on small-world networks," *Phys. Rev. E* 62, 7059-7064 (2000).

7. S. Solomon, G. Weisbuch, L. de Arcangelis, N. Jan, and D. Stauffer, "Social percolation models," *Physica A* 277, 239-247 (2000).

8. J. Hoshen and R. Kopelman, "Percolation and cluster distribution: Cluster multiple labeling technique and critical concentration algorithm," *Phys. Rev. B* 14, 3438-3445 (1976).

9. C.-K. Hu, "Histrogram Monte Carlo renormalization group method for percolation problems," *Phys. Rev. B* 46, 6592-6595 (1992).

10. H. Gould and J. Tobochnik, *An introduction to computer simulation methods*, 2nd edition, p. 444, Addison-Wesley, Reading, MA (1996).

11. C. Moukarzel, "A fast algorithm for backbones," *Int. J. Mod. Phys. C* 9, 887-895 (1998).

12. J.E. de Freitas, L.S. Lucena, and S. Roux, "Percolation as a dynamical phenomenon," *Physica A* 266, 81-85 (1999).