The Comparison of DFS and BFS Methods on 2D Ising Model

Dmitrii Yu. Kapitan[1,2], Alexey E. Rybin[1,2], Egor V. Vasiliev[1,2], Alexander V. Perzhu[1,2], Petr D. Andriuschenko[1,2]

[1] Far Eastern Federal University, Vladivostok, Russia
[2] Institute of Applied Mathematics, Far Eastern Branch, Russian Academy of Science, Vladivostok, Russia

Abstract

We consider Deep-First Search and Breadth-First Search graph traversal algorithms for finding clusters boundaries on 2D Ising model. We implement and compare these algorithms at different cluster density on lattice. It is shown that Breadth-First Search method has a huge advantage at clusters search on the lattice.

1 Introduction

The theory of percolation is used in physics, chemistry, and other fields to describe the emergence of connected structures (clusters) in random environments consisting of individual elements [1]. In order to study various effects within the framework of the percolation theory, the tools are needed that make it possible to quickly and accurately determine clusters by given characteristics. Due to the very large number of splits into clusters, speeding up the process of finding such even by a small value will give a significant increase in the calculation speed of the system as a whole. Therefore, effective tools are needed to search for clustering, the average number of nodes in a cluster, the distribution of clusters by size, the appearance of an infinite cluster and the proportion of its constituent nodes, as well as the accumulation of statistics across clusters. In this paper, the authors compare the effectiveness of two different approaches to finding clusters on the example of the two-dimensional Ising model.

2 Depth-First Search

A depth search version was explored in the 19th century by the French mathematician Charles Pierre Tremo as a strategy for solving labyrinths. This is called the Tremo algorithm. The Tremo algorithm is an effective method for finding a way out of a maze that requires drawing lines on the floor to mark the path, and is guaranteed to work for all mazes that have well-defined passages. In fact, this algorithm, which was discovered in the 19th century, was used about a hundred years later as the foundation for a modern depth-first search algorithm [2-4].

The depth search algorithm is very simple. First consider the starting node. Randomly select one of the neighbors of this node and go there. If this node has neighbors, arbitrarily choose one of them and go there if we have not visited this node yet. And we simply repeat this process until one of two conditions occurs. If we reach the end node, we complete the algorithm and report success. If we reach the site only with those neighbors which we have already visited, or there are no neighbors at all, we go back one step and visit one of the neighbors that we did not visit last time.

This algorithm is called depth search, because we always give preference to finding the deepest node we know about. If there are connections, they are broken arbitrarily, but as soon as we break our first connection (choosing which of the neighbors of the initial node to explore first), we will not try to look for other neighbors of the starting node until the first neighbor (and all of its neighbors) have been fully studied.

In the sequential case, the algorithm has algorithmic complexity $O(|V| + |E|)$, where $|V|$ - number of vertices in the graph, $|E|$ - number of edges in the graph.

In: Sergey I. Smagin, Alexander A. Zatsarinnyy (eds.): V International Conference Information Technologies and High-Performance Computing (ITHPC-2019), Khabarovsk, Russia, 16-19 Sep, 2019, published at http://ceur-ws.org

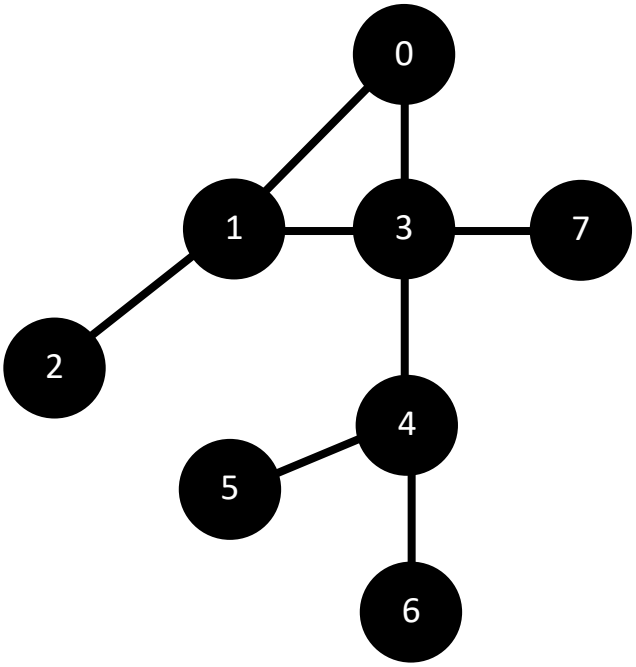

Figure 1: Depth-First Search Algorithm

3 Breadth-First Search

A breadth search was formally proposed by E. F. Moore [5] in the context of a search for a path through the labyrinth in 1959. Lee [6] independently discovered the same algorithm in the context of PCB conductor layout in 1961.

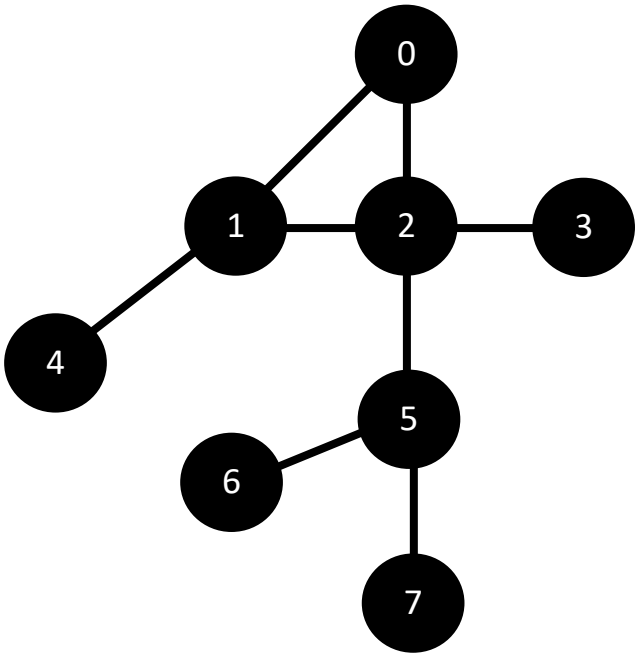

Figure 2: Breadth-First Search Algorithm

A breadth-first search allows you to calculate the shortest distances (in terms of the number of edges) from a selected vertex of a directed graph to all other vertices, and/or build a root directional tree which distances coincide with the distances in the original graph. In addition, the breadth search allows us to solve the task of verifying the reachability (if there are paths between the vertex source and the rest of the vertices of the graph) [7, 8, 9].

The algorithm is based on traversing the vertices of the graph "by layers". At each step, there are many “advanced” vertices, for which adjacent ones are checked whether they belong to those that have not yet been visited. Still not

visited vertices are added to the new set of "advanced" vertices processed in the next step. Initially, only the source node enters the set of “advanced” vertices, from which the traversal begins.

This algorithm has similar complexity to DFS - O(|V| + |E|).

The breadth-first search algorithm is similar to the DFS algorithm, with the only difference that nodes are placed not in the stack, but in the queue. In this case, the principle of FIFO will be applied - Fist In First Out. The algorithm can be used to find the shortest path between two nodes.

The effect of the BFS on the example of a cluster search in the lattice model is shown on Figure 3.

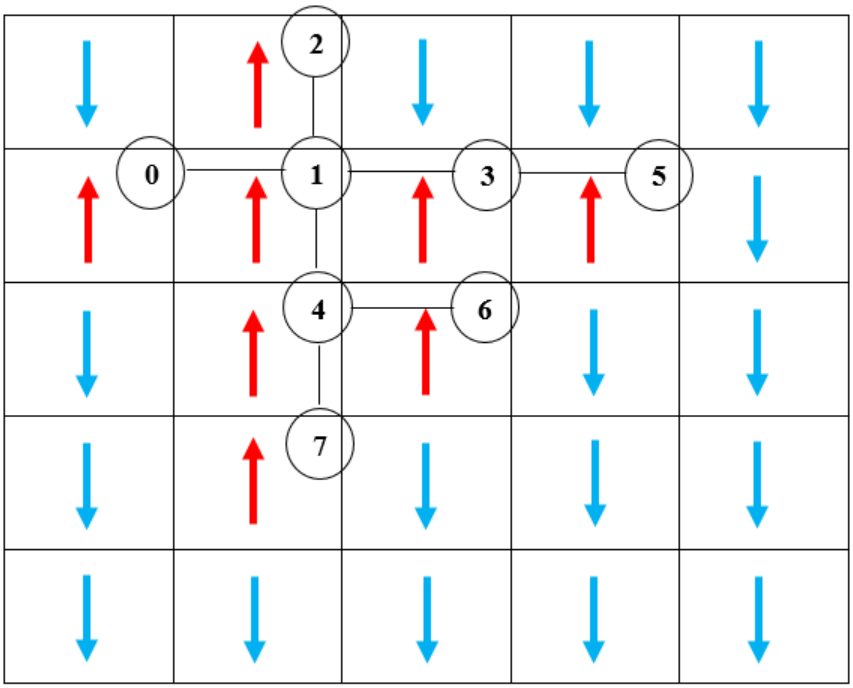

Figure 3: Implementation of Breadth-First Search algorithm on square lattice

4 Comparison

The fundamental difference between trees and graphs (which makes a difference in the implementation of DFS / BFS) is that there are no cycles in the trees. In graph theory, there is a cycle in any graph where you can leave a node and go through the graph back to that node. Non-directed graphs always contain loops, because you can simply move between any two neighbors. There is one exception to this rule: a graph without edges [10-14]. But this sort of graphs won’t be discussed in this article.

Most BFS algorithms perform an abbreviated DFS before traversing each adjacency list, placing this result in a queue, which is then bypassed. For this reason, when used in trees, it is twice as slow as DFS. However, the advantage is that if you are looking for close neighbors, BFS is faster than DFS [15].

An important factor when choosing an algorithm for implementation on a high-performance hardware is the simplicity and efficiency of parallelization, to get all the benefits provided by the cluster. The implementation of the breadth-first search algorithm allows you to simply parallelize the program, with a greater effect than a parallel algorithm for depth-first search.

The difference between depth-first search and breadth-first search is that (in the case of an undirected graph) the result of the depth-first search algorithm is a certain route, following which you can go around all the graph vertices accessible from the initial vertex [16]. In this way, it is fundamentally different from the breadth-first search, where many vertices are simultaneously processed, and only one vertex is processed in the depth-first search at each moment of the algorithm execution.

On the other hand, the DFS does not find the shortest paths, but it is applicable in situations where the graph is not completely known, but is being investigated by some automated device [17].

To compare these two algorithms, the program was created on the C++. The algorithms were tested on a square lattice of the two-dimensional Ising model, where temperature is a randomizing factor, i.e., at low temperatures, the system is completely ordered. To calculate the size of the maximum cluster, the number of spins was set as N = 100x100, 300x300, 500x500, 700x700, 1000x1000, 2000x2000. The spins were clustered by orientation. Calculations were carried out for one stream in a supercomputer cluster. Several experiments were conducted. In the first experiment, a low-temperature situation was modeled, with all spins of the system entering the cluster. The second

experiment was conducted near the phase transition temperature T_c = 2.269, where the maximum cluster of co-directed spins begins to appear, i.e., it is large, but not all spins are included in it. In the third experiment, the lattice is completely randomized and consists of a set of small clusters of codirectional spins.

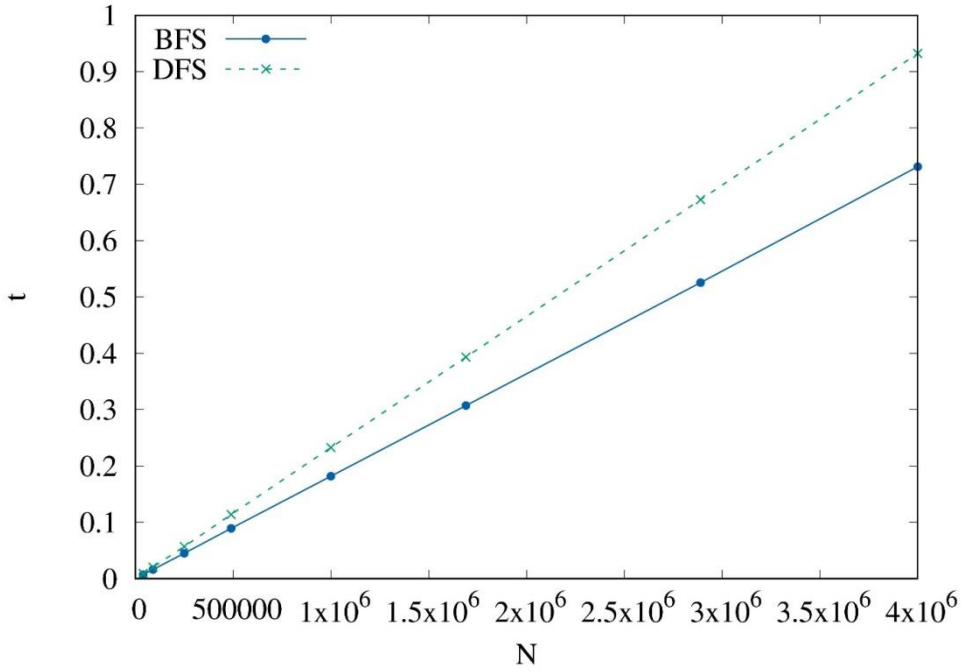

Figure 4: Comparison of calculation speed between BFS and DFS algorithms when each spin enters to the cluster

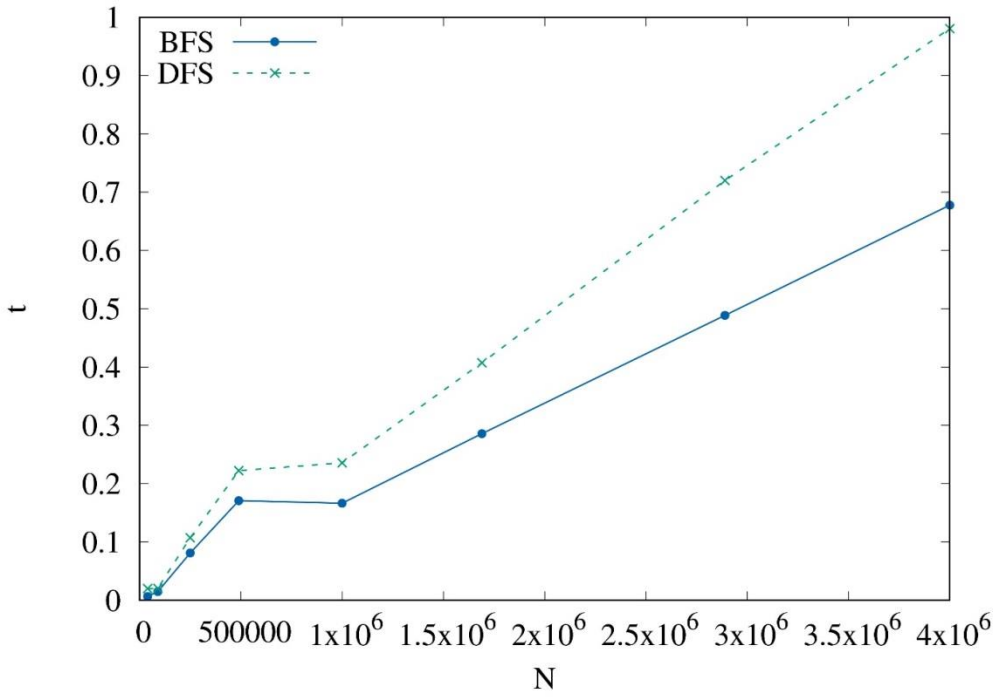

Figure 5: Comparison of calculation speed between BFS and DFS algorithms at phase transition area

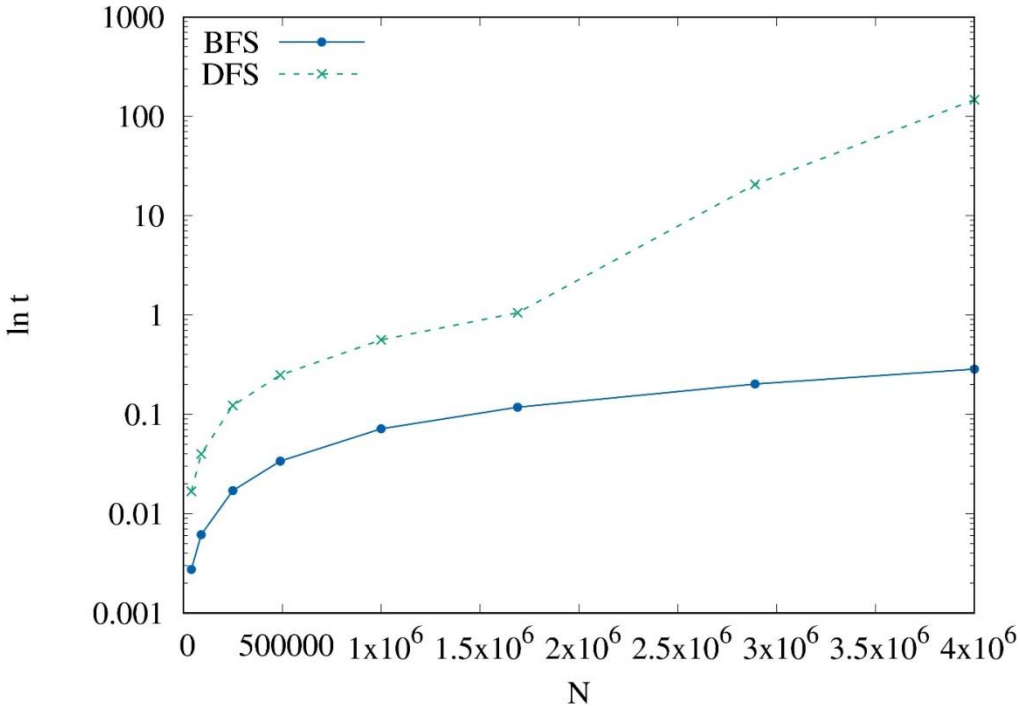

Figure 6: Comparison of calculation speed between BFS algorithm and DFS algorithm when spins oriented chaotically

5 Conclusion

In this paper, algorithms were considered for finding clustering, the average number of nodes in a cluster, the size distribution of clusters, and the appearance of an infinite cluster. We considered breadth-first search and depth-first search traversing algorithms. A program was also written to compare two algorithms within the framework of the conditions we are interested in. On the basis of the obtained results, it can be concluded that when choosing an algorithm for searching clusters, one should use a wide search algorithm due to its advantage in execution speed on a graph and a simpler parallelization implementation, which is of great importance for supercomputer calculations. And it will help in further studies of ferromagnetic and antiferromagnetic spin models.

Acknowledgements

The research was carried out within the state assignment of FASO of Russia No. 075-00400-19-01.

Computational resources provided by FEFU supercomputer cluster (cluster.dvfu.ru) and Shared Facility Center "Data Center of FEB RAS" (Khabarovsk, Russia).

References

1. Tarasevich, Yu.Yu.: Percolation: Theory, Applications, Algorithms // Moscow, URSS Publisher. 64 p. (2002)

2. Tiwari, P.: An efficient parallel algorithm for shifting the root of a depth first spanning tree. // Journal of Algorithms — Volume 7, Issue 1 – Pp. 105-119. (1986)

3. Awerbuch, B.: A new distributed Depth-First-Search algorithm // Information Processing Letters. — Volume 20, Issue 3 — Pages 147-150 (1985)

4. Mohan, B. Sharma; Sitharama S. Iyengar; Narasimha K. Mandyam, An efficient distributed depth-first-search algorithm // Information Processing Letters 32. — Pages 183-186 (1989)

5. Moore, E. F.: The shortest path through a maze // Proceedings of an International Symposium on the Theory of Switching (Cambridge, Massachusetts, 2–5 April 1957) — Harvard University Press. — Vol. 2. — P. 285–292. — 345 p. (Annals of the Computation Laboratory of Harvard University; Vol. 30) (1959)

6. Lee, C. Y.: An algorithm for path connection and its applications. IRE Transactions on Electronic Computers, EC-10(3), pp. 346–365 (1961)

7. Makki, S.A.M.: Efficient distributed breadth-first search algorithm // Computer Communications (1996)

8. Yunzhou, Zhu; To-Yat, Cheung: A new distributed breadth-first-search algorithm // Elsevier B.V. (1987)

9. Awerbuch, B., Gallager, R.: Distributed BFS algorithms // Foundations of Computer Science // 26th Annual Symposium on Foundations of Computer Science (1985)

10. Shimon, E.: Graph Algorithms. // W. H. Freeman & Co. New York, NY, USA (1979)

11. Levitin, A.: Introduction to the Design and Analysis of Algorithms // Addison Wesley, 592 pages. (2011)

12. Kocay, W., Kreher, D.L.: Graphs, Algorithms, and Optimization // CRC Press. (2004)

13. Richard, J. Trudeau: Introduction to Graph Theory // Dover Publications Inc., NY (1993)

14. Khee Meng Koh, F. M., Dong, Dong Fengming, Tay, Eng Guan: Introduction to Graph Theory: Solutions Manual// World Scientific. (2006)

15. Cui, Zehan, Chen, Licheng, Chen, Mingyu, Bao, Yungang, Huang, Yongbing, Lv, Huiwei: Evaluation and Optimization of Breadth-First Search on NUMA Cluster. // IEEE International Conference on Cluster Computing — CLUSTER 2012 — Pages 438-448. (2012)

16. Youzou Miyadera, Koushi Anzai, Hiroshi Unno, Takeo Yaku: Depth-first layout algorithm for trees // Springer Science & Business Media, (2013.)

17. Cidon, I.: Yet another distributed depth-first-search algorithm // Inf. Process. Lett. 26, 6 (January) — Pages 301-305. (1988)