A Percolation Algorithm Based on Cellular Automata

Mohammad Khanjary [a,*], Masoud Sabaei [b] and Mohammad Reza Meybodi [b]
[a] Computer Engineering Department, Science and Research Branch, Islamic Azad University, Tehran, Iran
[b] Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran

***Abstract*—In many applications of percolation theory, checking the establishment of the spanning clump/cluster of overlapping particles that spans all over the field is an essential task. Given a percolation theory field modeled by two-dimensional lattice (matrix), in this paper, we present an algorithm which determines if there is a spanning clump in lattice or not. The spanning clump is the largest cluster in the field which that spans the entire network vertically, horizontally or both. Due to wide range of properties and applications of cellular automata such as simplicity and distributedness, we use them in our algorithm. The proposed algorithm is simple but yet useful and also could be run in a parallel / multicore machines. Also, the approach of the algorithm could be extended to higher dimensions.**

I. INTRODUCTION

The concept of continuum percolation, originally due to Gilbert [1], is to find the critical density of a Poisson point process at which an unbounded connected component almost surely appears so that the network can provide long-distance multihop communication. Since then, Gilbert's model has become the basis for studying continuum percolation in different technical fields such as wireless networks e.g. [2] and [3]. Recently, percolation theory has been considered by researchers to be used to examine coverage and connectivity in sensor networks too [4]-[12].

In general, percolation theory could be classified to two models called discrete percolation [13] and continuum percolation [14]. In discrete percolation (also called the lattice model), the sites of the lattice are close or open due to probability p and may have different tessellation such as square, triangle, honeycomb and etc. While in continuum percolation, the positions of the sites are randomly distributed and thus, there is no need to have a different analysis for each of these regular lattices. While in discrete percolation theory, we are interested in finding the critical probability denoted by p_c in which percolation occurs, in continuum percolation we are interested in finding the critical density denoted by λ_c at which an infinite or large clump of overlapping objects first appears that spans the entire network. The density λ_c is the critical value for the density λ such that there exists no such clump of overlapping objects almost surely when $\lambda < \lambda_c$ (the system is said to be in the subcritical phase), but it exists almost surely when $\lambda > \lambda_c$ (the system is said to be supercritical) and we say that percolation occurs.

In most of research in continuum percolation, the critical density of different shapes will be calculated by huge Monte Carlo method [15]. Therefore using an algorithm to test the occurrence of percolation in the network is an essential and important task of such research.

In this paper, we present an algorithm to check the establishment of spanning clump (percolation) in a two-dimensional lattice by using cellular automata. The proposed algorithm works locally and could be executed distributed/parallel which makes it suitable for huge data experiments. Also, the approach of the algorithm could be extended to higher dimensions.

The remainder of this paper is organized as follows: section II reviews related works, section III presents a brief description on cellular automata and its features, section IV presents the approach of our algorithm and finally section V concludes the paper.

II. RELATED WORKS

The proposed algorithms for percolation usually are related to finding out the percolation probabilities as a framework for Monte Carlo simulations. In other words, these algorithms try to estimate the critical probabilities in which percolation starts for different materials. As an instance, Hoshen et al. [16] proposed an extended multiple labeling technique for site-bond percolation problem with for square and triangular lattices. The site-bond approach is useful when a percolation process cannot be exclusively described in the context of pure site or pure bond percolation.

In [17], authors obtained precise estimates for the fractal dimensions of the sample spanning cluster, the backbone, and the minimal path in order to identify the universality classes of four different Invasion percolation (IP) processes (site and bond IP, with and without trapping) by using efficient algorithms for simulating invasion percolation. In two dimensions IP is characterized by two universality classes, one each for IP without trapping, and site and bond IP with trapping. In a three-dimensional site IP with and without trapping is in the universality class of random percolation, while bond IP with trapping is in a distinct universality class, which may be the same as that of optimal paths in strongly disordered media. Also, in [18], Masson et al. presented a computationally fast

* khanjary@srbiau.ac.ir

Invasion Percolation (IP) algorithm. IP is a numerical approach for generating realistic fluid distributions for quasi-static immiscible fluid invasion in porous media. The algorithm proposed uses a binary-tree data structure to identify the site connected to the invasion cluster that is the next to be invaded and gravity is included. Also, trapping is not explicitly treated in the numerical examples but can be added.

In [19], authors presented a recursive algorithm for sampling properties of physical clusters such as size distribution and percolation. The approach can be applied to any system with periodic boundary conditions, given a spatial definition of a cluster. The recursive cluster identification algorithm is somewhat slower than the iterative methods at low volume fraction but is at least as fast at high densities. The percolation analysis, however, is considerably faster using recursion, for all systems studied. In other research [20], authors presented an efficient algorithm for finding the current-carrying backbone in the planar site percolation model. It finds the backbone in speed to be almost four times as high as depth-first-search algorithms. Similar algorithm has been introduced in comparison to commonly-used Tarjan's depth-first-search algorithm in [21].

In [22], authors proposed a stochastic cellular automata model for wild-land fire spread dynamics under flat terrain and no-wind conditions. They modeled the dynamics of fire spread as a stochastic event with an effective fire spread probability S which is a function of three probabilities: the proportion of vegetation cells across the lattice, the probability of a burning cell become burnt, and the probability of the fire spread from a burning cell to a neighbor vegetation cell.

III. CELLULAR AUTOMATA

A cellular automaton (CA) is a rule-based computing machine, which was first proposed by von Newmann in early 1950s and systematic studies were pioneered by Wolfram in 1980s. Since a cellular automaton consists of space and time, it is essentially equivalent to a dynamical system that is discrete in both space and time. The evolution of such a discrete system is governed by certain updating rules rather than differential equations. Although the updating rules can take many different forms, most common cellular automata use relatively simple rules.

Formally, cellular automata are classified to three categories [23].

1. Finite-State Cellular Automata

In general, we can define a finite-state cellular automaton with a transition rule $G = [g_{ij,\dots,l}], (i,j,\dots,l = 1,2,\dots,N)$ from one state $\Phi^t = [\phi^t_{ij,\dots,l}]$ at time level n to a new state $\Phi^{t+1} = [\phi^{t+1}_{ij,\dots,l}]$ at a new time step $n+1$. The value of subscript $(i,j,\dots,l)$ denotes the dimension, d, of the cellular automaton. Therefore, a CA in the d-dimensional space has N^d cells. For the 2D case, this can be written as

$$G = \Phi^t \to \Phi^{t+1}, g_{ij}: \phi^t_{ij} \to \phi^{t+1}_{ij}, \qquad (i,j = 1,2,\dots,N).$$

In the case of sum-rule with $4r+1$ neighbors, this becomes

$$\phi^{t+1}_{ij} = G\left(\sum_{\alpha=-r}^{r}\sum_{\beta=-r}^{r} a_{\alpha\beta}\phi^t_{i+\alpha,j+\beta}\right), \quad (i,j = 1,2,\dots,N),$$

where $a_{\alpha\beta}$ $(\alpha,\beta = \pm1,\pm2,\dots,\pm r)$ are the coefficients. The cellular automata with fixed rules defined this way are deterministic cellular automata. In contrast, there exists another type, namely, the stochastic cellular automata that arise naturally from the stochastic models for natural systems.

2. Stochastic Cellular Automata

When using cellular automata to simulate the phenomena with stochastic components or noise such as percolation and stochastic process, the more effective way is to introduce some probability associated with certain rules. Usually, there is a set of rules and each rule is applied with a probability.

Another way is that the state of a cell is updated according to a rule only if certain conditions are met or certain values are reached for some random variables. For example, the rule for 2D a cellular automaton $g(\phi^t_{ij}) = \phi^{t+1}_{ij}$ is applied at a cell only if a random variable $v \le \Gamma(\phi^t_{ij})$ where the function $\Gamma \in [0,1]$. At each time step, a random number v is generated for each cell (i,j), and the new state will be updated only if the generated random number is greater than Γ, otherwise, it remains unchanged. Cellular automata constructed this way are called stochastic or probabilistic cellular automata.

3. Reversible Cellular Automata

A cellular automaton with an updating rule $g(\phi^t_{ij}) = \phi^{t+1}_{ij}$ is generally irreversible in the sense that it is impossible to know the states of a region such as all zeros were the same at a previous time step or not. However, certain class of rules will enable the automata to be reversible. For example, a simple finite difference (FD) scheme for a dynamical system

$$u(t+1) = g[u(t)] - u(t-1)$$

or

$$u(t-1) = g[u(t)] - u(t+1),$$

is reversible since for any function $g(u)$, one can compute $u(t+1)$ from $u(t)$ and $u(t-1)$, and invert $u(t-1)$ from $u(t)$ and $u(t+1)$. The automaton rule for 2D reversible automata can be similarly constructed as

$$u^{t+1}_{i,j} = g(u^t_{i,j}) - u^{t-1}_{i,j},$$

together with appropriate boundary conditions such as fixed-state boundary conditions.

IV. THE ALGORITHM: PADSN

1. The Model

The model is based on the spatially explicit representation and the landscape is depicted as a square and two-dimensional lattice (a typical square lattice has been shown in Fig. 1). Each cell is defined by:

i. Its discrete position (i, j) in the lattice, where $i = 1, \dots, l$ is the row and $j = 1, \dots, l$ is the column.

ii. The finite set of internal states variables that describes the possible behavior of the cells in a given time step t which are $S_{(i,j)}^{t} \in [B, O, P]$ where **B** means the cell is *blocked* and nothing can percolate through this cell. These cells are shown by *black* shades in Fig. 1. **O** means this cell is *open* (not *blocked*) and could be percolated, but it has not been percolated so far. These cells are shown by *white* shades in Fig. 1. **P** means this cell was an *open* cell and is *percolated* now. These cells are shown by *blue* shades in Fig. 1.

iii. The set of finite Moore neighborhood cells $N(i, j)$, where is the Moore neighborhood as shown in Fig. 2 and represents the neighborhood relations in the model and comprises the eight cells surrounding (i^*, j^*) of a central cell (i, j) according with the definition following definition:

$$N(i,j) = \{(i^*, j^*): |i - i^*| \leq 1, \quad |j - j^*| \leq 1\}$$

iv. The transition function that calculates the future cell state as a function of the present state of the cell and presents neighborhood cell states $f: S_{(i,j)}^{t} \times S_{N(i,j)}^{t} \rightarrow S_{(i,j)}^{t+1}$ where the time t is also represented by discrete values or time steps. Thus, the time evolution of the model is driven by the interaction between the cell states and the cell neighborhood states. Starting from a given configuration of cells initial states, the cellular automaton self-replicates the sequent cell states. The cellular automata model is stochastic because the state transition function is performed according to probabilities values.

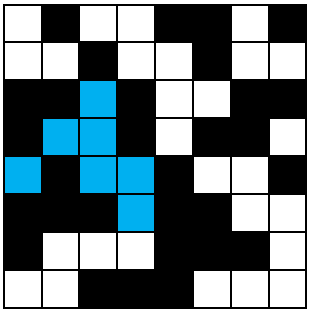


Fig. 1. A typical 8×8 lattice. Black shades represents the blocked cells, white shades represents the open cells and blue shades represents the percolated cells.

2. State Transition

As it mentioned in previous section, we use the Moore neighborhood model in our algorithm. As it is shown in Fig. 2, each cell has eight neighbors. If current state of a cell in *blocked*, nothing could change its state and it will keep its state forever (see Fig. 3(a)).

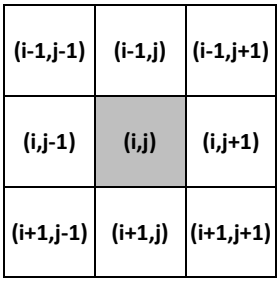
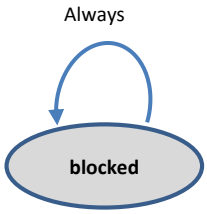
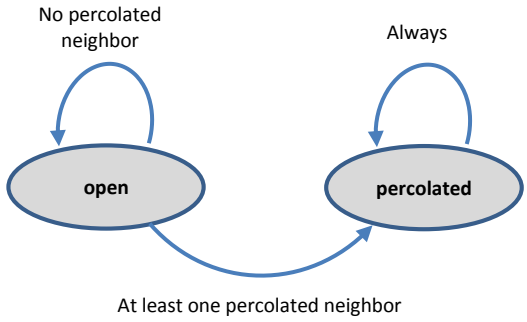


Fig. 2. The Moore neighborhood model which is used in the algorithm.

Also, if the current state of the cell is *percolated*, nothing can change its state (see Fig. 3(b)). But if the current state of the cell is *open* and there is at least one neighbor with *percolated* state, then its state will be changed to *percolated* (see Fig. 3(b)).



a) State transition for blocked status.



b) State transition for open and percolated status.

Fig. 3. State transition of cellular automata in our algorithm.

3. The algorithm

The simplest data structure for representing a lattice is matrix. But as it is shown in Fig. 3, the only state transition will be happened from *open* state to *percolated* state and nothing will be done for *blocked* and *percolated* states. Therefore, it does not need to check all cells and only checking the conditions of cell with *open* state is enough to progress the algorithm. Therefore, a preprocess phase will be done on the matrix to insert all cells with *open* state into a linked list. Then, state of each cell in this linked list will be updated by using mentioned rules in previous section. At the beginning some cells with *open* state in the edge of lattice will be turned to *percolated*. If state of a cell changed to *percolated*, it will be removed from linked list. Fig. 4 shows the pseudo code of the algorithm.

```
% Pseudo code of Percolation Algorithm based on Cellular Automata (PACA)

function PACA(LinkedList, Lattice)
    for all cell(i,j) in LinkedList
        if cell(i,j) has at least one neighbor with percolated state in Lattice
           change the state of cell(i,j) to percolatd
           if cell(i,j) is a edge cell in Lattice
               return "there is a spanning clump in the lattice!"
           end if
        end if
    end for
    return "there is not any spanning clump in the lattice!"
end function
```

Fig. 4. . The proposed algorithm.

The algorithm will be terminated in one of the two following situations:

- When the current cell is an edge cell in the lattice and state transition from *open* state to *percolated* state has been done in this iteration, it means a percolation clump from other side of lattice to this side exists. In other words, percolation occurred. In this situation, the algorithm will be terminated and checking other cells is not needed.

- When all cells in linked list passed and no state transition could be done on these cells. It means there is not any other open cell in the lattice which its state could be changed anymore. This tells that there is not any spanning clump and algorithm will be terminated.

Fig. 5 shows all phases of the algorithm on a sample 8×8 lattice.

4. Complexity of the algorithm

In this section, we discuss on complexity of the proposed algorithm. Given a $n \times n$ lattice with m open cells, obviously, the best case will happen when there is a chain of m cells which spans the lattice as it is shown in Fig. 6. Therefore, the complexity for best case of the algorithm will be $\Omega(m)$. Also, the worst case of the algorithm is happened when the lattice is like Fig. 7. As it seen, number of phases to complete the algorithm will be as

$$\left(\frac{n}{2} \times (n-1)\right) + 1 = O(n^2)$$

For each phase at most m cells in linked list must be processed. Therefore, in worst case the complexity of the algorithm will be $O(mn^2)$.

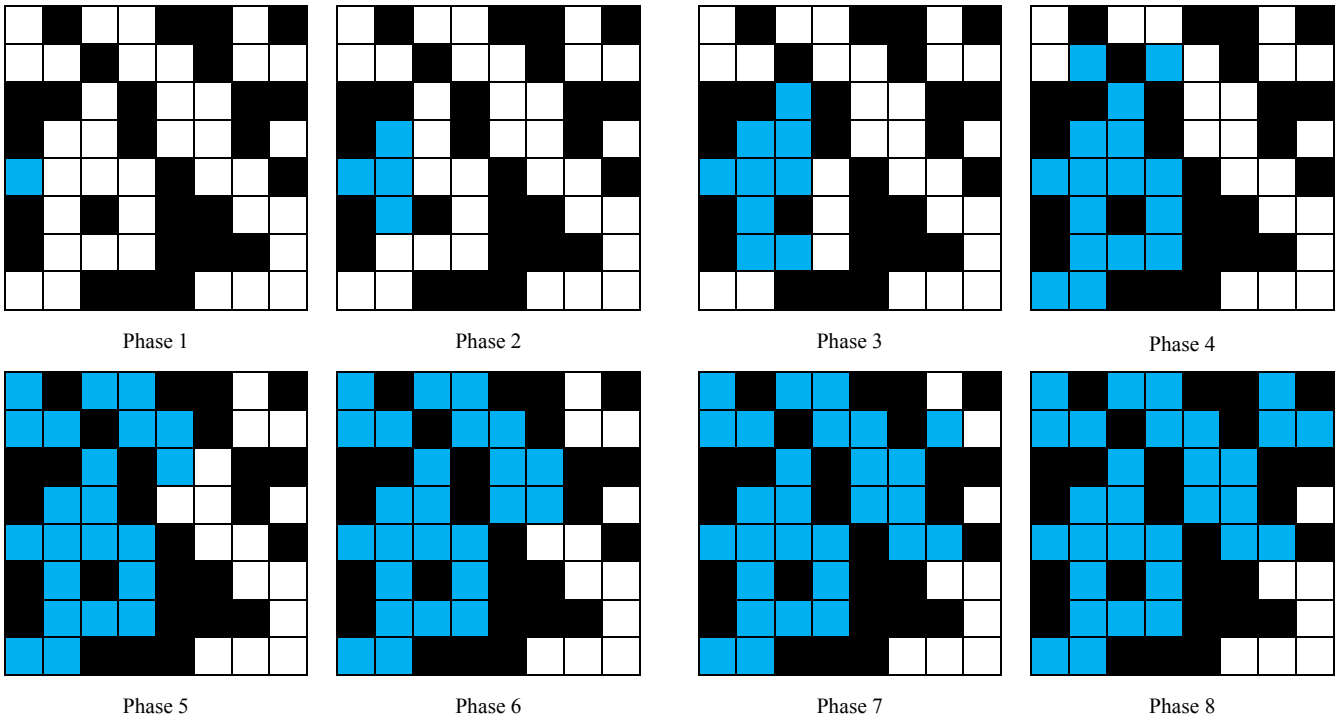


Fig. 5. Phases of the algorithm on a sample 8×8 lattice. As seen, at the latest phase (8th) of our sample, state of a cell in the other edge of lattice changes to *percolated* and due to first condition, algorithm will be terminated.

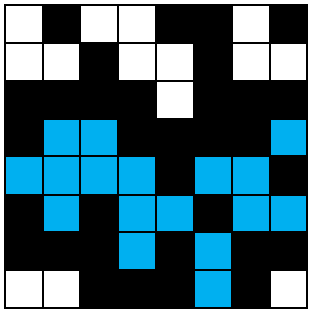


Fig. 6. Best case for the algorithm.

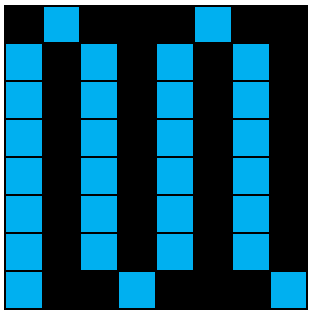


Fig. 7. Worst case for the algorithm.

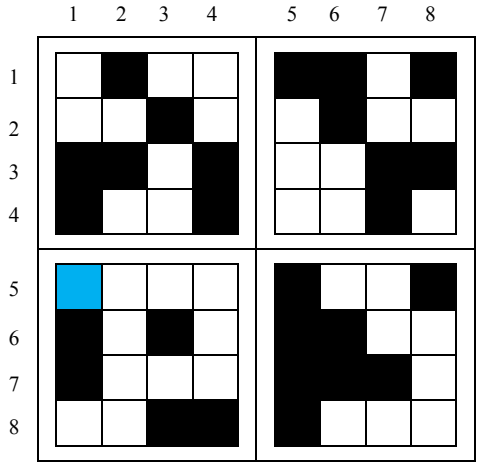


Fig. 8. Decompose of lattice to execute algorithm parallel.

TABLE I
INDEXES OF DECOMPOSED SLICES

$r = k^2$	Row indexes	Col indexes
1	1 to 5	1 to 5
2	4 to 8	1 to 5
3	1 to 5	4 to 8
4	4 to 8	4 to 8

5. Distributedness

As it is mentioned in previous sections, the proposed algorithm could be executed distributed or parallel. Assuming that we have a single original lattice which all threads can access it. The lattice could be decomposed to some equal slices and then, an instance of algorithm code be run for each slice separately.

Consider Fig. 8. As it is shown, a $n \times n$ lattice could be broken to $r = k^2$ slices with $m \times m$ cells (where $m = \frac{n}{k}$). In our example (Fig. 8), we decomposed a 8×8 lattice to 4 slices with 4×4 cells. To consider the border effects of each slice in neighboring slices, in each iteration, the algorithm read $(m + 2) \times (m + 2)$ cells from the lattice (if applicable). In other word, the surrounding row and column of a slice which put into other neighboring slices affect the results and must be considered. The indexes of each slices in Fig. 8 which must be read in each phase have been shown in table I.

V. Conclusion and Future Works

In this paper, we introduced an algorithm based on cellular automata to check the establishment of spanning clump (percolation) in a two-dimensional The proposed algorithm works locally and could be executed distributed/parallel which makes it suitable for huge data experiments. The order of proposed algorithm in worst case is $O(mn^2)$. Moreover, the approach of the algorithm could be extended to higher dimensions.

References

[1] E.N. Gilbert, "Random Plane Networks," J. SIAM, vol. 9, no. 4, pp. 533-543, 1961.
[2] I. Glauche, W. Krause, R. Sollacher and M. Greiner, "Continuum Percolation of Wireless Ad Hoc Communication Networks," Physica A, vol. 325, pp. 577-600, 2003.
[3] A. Jiang and J. Bruck, "Monotone Percolation and the Topology Control of Wireless Networks," in Proc. IEEE INFOCOM, 2005, pp. 327-338.
[4] H.M. Ammari and S.K. Das, "Integrated coverage and connectivity in wireless sensor networks: A two-dimensional percolation problem," IEEE Transactions on Computers, vol. 57, no. 10, pp. 1423-1434, 2008.
[5] H.M. Ammari and S.K. Das, "Critical density for coverage and connectivity in three-dimensional wireless sensor networks using continuum percolation," IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 6, pp. 872-885, 2009.
[6] F. Xing and W. Wang, "On the critical phase transition time of wireless multi-hop networks with random failures," in Proc. ACM MOBICOM, 2008, pp. 175-186.
[7] L. Liu, X. Zhang and H.M. Ma, "Optimal density estimation for exposure-path prevention in wireless sensor networks using percolation theory," in Proc. IEEE INFOCOM, 2012, pp. 2601-2605.
[8] L. Liu, X. Zhang and H. Ma, "Percolation theory-based exposure-path prevention for wireless sensor networks coverage in internet of things," IEEE Sensors Journal, vol. 13, no. 10, pp. 3625-3636, 2013.
[9] L. Liang, Z. Xi and M. Huadong, "Exposure-path prevention in directional sensor networks using sector model based percolation," in Proc. IEEE ICC, 2009, pp. 1-5.
[10] G. Yang andD. Qiao, "Critical conditions for connected-k-coverage in sensor networks," IEEE Communications Letters, vol. 12, no. 9, pp. 651-653, 2008.
[11] P. Balister, Z. Zheng, S. Kumar and P. Sinha, "Trap coverage: Allowing coverage holes of bounded diameter in wireless sensor networks," in Proc. IEEE INFOCOM, 2009, pp. 136-144.
[12] M. Khanjary, M. Sabaei and M.R. Meybodi, "Critical density for coverage and connectivity in two-dimensional aligned-orientation directional sensor networks using continuum percolation," IEEE Sensors Journal, vol. 14, no. 8, pp. 2856-2863, 2014.
[13] G. Grimmett, "Percolation," Springer Verlag, 1989.
[14] R. Meester and R. Roy, "Continuum Percolation," Cambridge University Press, 1996.
[15] S. Mertens and C. Moore, "Continuum Percolation Thresholds in Two Dimensions," Phys. Rev. E 86, 061109, 2012.

[16] J. Hoshen, P. Klymko and R. Kopelman, "Percolation and cluster distribution. III. Algorithms for the site-bond problem," Journal of Statistical Physics, vol. 21, no. 5, pp. 583-600, 1979.
[17] A.P. Sheppard, M.A. Knackstedt, W.V. Pinczewski and M. Sahimi, "Invasion percolation: new algorithms and universality classes," J. Phys. A: Math. Gen., vol. 32, pp. L521–L529, 1999.
[18] Y. Masson and S.R. Pride, "A Fast Algorithm for Invasion Percolation," Transp Porous Med., vol. 102, pp. 301–312, 2014.
[19] T. Edvinsson, P. J. Rasmark and C. Elvingson, "Cluster Identification and Percolation Analysis Using a Recursive Algorithm," Molecular Simulation, vol. 23, no. 3, pp. 169-190, 1999.
[20] W.G. Yin and R. Tao, "Algorithm for finding two-dimensional site percolation backbones," Physica B: Condensed Matter, vol. 279, no. 1–3, pp. 84-86, 2000.
[21] W.G. Yin and R. Tao, "Rapid Algorithm For Identifying Backbones In The Two-Dimensional Percolation Model," International Journal of Modern Physics C, vol. 14, no. 10, pp. 1427-1437, 2003.
[22] R.M. Almeida and E.E.N. Macau, "Stochastic Cellular Automata Model For Wildland Fire Spread Dynamics," in Proc. DINCON, Serra Negra, 2010, pp. 249-253.
[23] X.S. Yang and Y. Young, "Cellular Automata, PDEs and Pattern Formation," Handbook of Bioinspired Algorithms and Applications, Chapman & Hall/CRC Press, 2005, pp. 271-282.