International Journal of Modern Physics C
Vol. 25, No. 11 (2014) 1450064 (11 pages)

DOI: 10.1142/S0129183114500648

New algorithm to test percolation conditions within the Newman–Ziff algorithm

I. V. Tronin

Molecular Physics Department
National Research Nuclear University MEPhI
Kashirskoe sh. 31, Moscow, 115409, Russian Federation
IVTronin@mephi.ru

Received 14 March 2014
Accepted 28 March 2014
Published 5 May 2014

A new algorithm to test percolation conditions for the solution of percolation problems on a lattice and continuum percolation for spaces of an arbitrary dimension has been proposed within the Newman–Ziff algorithm. The algorithm is based on the use of bitwise operators and does not reduce the efficiency of the operation of the Newman–Ziff algorithm as a whole. This algorithm makes it possible to verify the existence of both clusters touching boundaries at an arbitrary point and single-loop clusters continuously connecting the opposite boundaries in a percolating system with periodic boundary conditions. The existence of a cluster touching the boundaries of the system at an arbitrary point for each direction, the formation of a one-loop cluster, and the formation of a cluster with an arbitrary number of loops on a torus can be identified in one calculation by combining the proposed algorithm with the known approaches for the identification of the existence of a percolation cluster. The operation time of the proposed algorithm is linear in the number of objects in the system.

Keywords: Percolation theory; numerical algorithms; continuum percolation; lattice percolation.

PACS Nos.: 02.70.-c, 05.10.Ln, 05.70.Jk.

1. Introduction: Formulation of the Problem

The problem of the determination of the percolation threshold in the problems of percolation theory is one of the central problems both for theory[1,2] and for numerical simulation.[3,4] Algorithms for numerical simulation of percolation processes have been improved in recent time; with the fast development of computer engineering, the percolation threshold in percolation problems on various lattices[5–7] and continuum percolation problems[8–12] can be determined on a usual personal computer. At the same time, many thousands of typical calculations are required to ensure the necessary accuracy of the determination of the percolation threshold; as a result, there is the problem of the efficiency of the algorithm for the calculation of the percolation threshold in a single calculation.

Percolation problems are numerically solved for finite-size systems. The determination of the dependence of the percolation threshold on the size of the system makes it possible to find the asymptotic value of the percolation threshold for an infinite system. The main aim of the algorithm for the determination of the percolation threshold at a fixed size of the system is to identify the existence of a cluster connecting the opposite boundaries of the system. At present, there are two main approaches to the solution of this problem. The first approach was proposed in Ref. 4 and is called the Hoshen–Kopelman algorithm. A state of the system with a given density of objects was generated in this algorithm. For this state, the algorithm joins objects into clusters and, then, identifies the presence or absence of a percolation cluster, i.e. a cluster connecting the opposite boundaries of the system. If such a cluster exists, this realization with a given density of objects "percolates." Finally, the Hoshen–Kopelman algorithm allows determining the dependence of the probability of the formation of the "percolation" cluster on the concentration of objects in the system with a fixed size. From the physical point of view, the Hoshen–Kopelman algorithm implements percolation on a canonical ensemble.[3,13] Its operation time is proportional to the square of the number of objects in the system $O(N^2)$.

In 2001, Newman and Ziff[3] proposed another approach to the determination of the percolation threshold. In this approach, objects are added to the system one-by-one and it is determined at each step whether an object belongs to clusters existing in the system. The addition of an object leading to the appearance of a cluster connecting the opposite boundaries of the system results in percolation of the system and, thereby, determines the number of objects at which percolation occurs. Since the density of an object is not fixed in a single calculation, this algorithm implements percolation in a grand canonical ensemble.[3,13] The main advantage of the Newman–Ziff algorithm is that its operation time is proportional to the number of objects in the system $O(N)$. At present, the Newman–Ziff algorithm is the most efficient algorithm of the determination of the percolation threshold.

The high efficiency of the Newman–Ziff algorithm is based primarily on the use of a highly efficient method for finding and uniting clusters ("weighted union/find with path compression"), which is as follows. Each object in the system is assigned with an integer called a label. If the label is positive, the corresponding object belongs to a cluster and the value of the label is an indicator of the root object of the cluster. If the label is negative, the corresponding object is treated as the root object of the cluster and the absolute value of the label is considered as the size of the cluster. A new added object is labeled by -1. When two clusters are joined, the label of the root of the smaller cluster is changed by referring to the root of the larger cluster. This scheme with minor modifications makes it possible to create an efficient algorithm for finding and uniting clusters.

The second important procedure affecting the efficiency of the algorithm as a whole is the procedure of testing percolation conditions after the addition of a new object to the system. For a finite system, the percolation condition is formulated in terms of boundary conditions, which can be imposed by different methods. For

clarity, we consider the problem of sites on a two-dimensional square lattice. Boundary conditions for other lattices, for the case of continuum percolation, and for a higher dimension of the space are formulated similarly. We suppose that the system percolates if the system includes a cluster connecting the left and right sides of the square lattice (the so-called "open" boundaries). Within the Newman–Ziff algorithm, after the addition of a new site, it is necessary to verify the appearance of a cluster satisfying this condition. A simple and efficient testing method is to begin the calculation with the formation of vertical clusters with the length equal to the side of the square on the left and right sides of the square. If the clusters on the left and right sides after the addition of a new site have the same root, it is accepted that the percolation condition is satisfied. The test of such a condition is reduced to the determination of the roots of any two sites on the left and right sides and does not reduce the speed of the algorithm.

However, this formulation of the boundary conditions has a significant disadvantage: within this approach, percolation only in the horizontal (or vertical) direction can be tested, whereas a number of problems require the test of percolation in all directions independently.[14] In these problems, percolation conditions along the axes are tested by passing each object on the first and second boundaries and by verifying whether the objects belong to one cluster. As a result, the operation time of the algorithm increases by several times.[14]

Another type of boundary conditions is so-called periodic boundary conditions. In this case, a site on the left side is identified with a site on the right side at the same level with the former site. In a problem with such boundary conditions, each site is assigned not only with the label, but also with a path to the root of the cluster in both axes. The test of the percolation condition in one of the axes is reduced in this case to the condition that the sum of the paths from two joined clusters to their common root should be longer than the side of the square.[15] In this formulation, the percolation threshold can be determined separately in each of the axes.

Periodic boundary conditions topologically correspond to the percolation problem and search for a closed path on a torus. In this case, the percolation cluster can generally have an arbitrary number of loops (winding number) on the torus. Such clusters are not necessarily of applied interest. On the contrary, in applied problems, it is necessary to verify the existence of a cluster continuously connecting identified points of a system without many intersections of the boundary. In other words, it is necessary to verify the existence of a one-loop cluster continuously connecting the opposite boundaries of the system. This is impossible within the method proposed in Refs. 3 and 15.

In this work, we propose an efficient algorithm that makes it possible to test the percolation condition in each of the coordinate axes independently. The algorithm is based on logical operators and can be used for problems of lattice and continuum percolation. It allows the combination of boundary conditions in each of the directions depending on a particular problem. The proposed algorithm applied to a problem with open boundaries makes it possible to test the percolation conditions in

each of the directions. This algorithm applied to a problem with periodic boundary conditions allows identifying the appearance of clusters with winding number equals to one and continuously connecting the opposite boundaries in the system.

2. Description of the Algorithm

The main idea of the algorithm is that the test of percolation conditions after the addition of a new object to the system should not require a large number of operations. In particular, in the case of periodic boundary conditions, the test of the percolation condition reduces to the summation of paths from two objects to their common root. In the case of open boundary conditions, the test of percolation reduces to the search for the roots of two objects. If it is necessary to perform the search for the entire set of objects (as in the case of the test of all objects located on the opposite boundaries), the operation time increases significantly and the main time at the addition of the object is spent on the test of percolation conditions. Thus, it is necessary to ensure the test of percolation conditions with the smallest number of operations.

Finally, the problem of the identification of a percolation cluster in the system is reduced to the verification of the existence of a cluster touching the opposite boundaries in the system. In this case, the "ends" of the cluster in the case of open boundary conditions can be located at arbitrary points of the opposite boundaries, whereas for periodic boundary conditions, they should be located at identified points. In order to test touching conditions, those cells along each of the boundaries that touch the cluster (in the case of percolation on a discrete lattice, a cell is a site or a bond, whereas the entire calculation region for continuum percolation is preliminarily divided into cells whose size is equal to the size of objects[8]) are written into special arrays created in the program. These arrays will be called contact arrays. The number of such arrays for the two-dimensional percolation problem is four, which is the number of the boundaries of the system. To save memory and to easily test the touch conditions, the ordinal numbers of cells are written into arrays bit-by-bit; i.e. each **bit** corresponds to **one** cell at the boundary. The elements of an array are several variables, e.g. eight-byte integers (integer*8). Since each bit corresponds to exactly one cell at the boundary, the number of variables used is equal to the length of the boundary divided by the number of bits in each variable. When integer*8 variables are used, $l/64$ variables for each boundary are necessary (if the length of the system is not a multiple of 64, $l/64$ should be rounded up). Thus, contact arrays have the structure $x(n, m)$, where n is the outer index of the array and $m \leq l/64$ is the ordinal number of an integer*8 variable.

A new integer characteristic, position in contact arrays, is introduced for the root site of the cluster. In fact, it is a reference to the element of contact arrays that corresponds to a given cluster. An important remark should be made. If all clusters appearing in the system were written into contact arrays, huge main memory would be required. This primarily concerns the very beginning of the computational

procedure when clusters appearing in the system consist of one object and do not touch boundaries. For this reason, to save memory, only clusters that contain an object touching the boundary are written into contact arrays. The position of such a cluster in the contact array is nonzero. The position of the cluster that does not touch boundaries is zero in the contact arrays.

Changes in contact arrays can be caused by two reasons: first, the appearance of an object touching the boundary and, second, joining of two clusters, one of which (or both) touches boundaries. In the former case, a new element corresponding to the appearing object is created in the contact arrays. In this element, the bit corresponding to a cell in which the object appears is set to be 1 and other bits are set to be 0. In the latter case, it is necessary to perform two operations: to remove the element corresponding to the added cluster from the contact arrays and to rewrite the contact arrays for a new resulting cluster. In order to increase the operation speed, an element is not removed from the contact arrays. Instead of this removal, a stack of the elements of contact arrays that are marked as removed is formed. Further, when new clusters appear, the elements of the stack are first tested. If the stack is empty, the contact arrays are simply expanded. If the stack contains removed elements, the last element from the stack is taken and contact arrays corresponding to this element are rewritten for a new cluster. Such a procedure makes it possible to strongly reduce the required memory.

Rewriting of contact arrays for the new cluster that appears when two other clusters are joined is performed with the use of a bitwise logical OR operator:

$$x_{\text{new}}(n_1, m) = x_1(n_1, m).\text{OR}.x_2(n_2, m), \quad m \leq l/64,$$

where n_1 is the position of the larger cluster in contact arrays and n_2 is the position of the added cluster.

For periodic boundary conditions, percolation conditions in each axis are tested with contact arrays by means of the application of the bitwise logical AND operator. If two clusters are joined at a certain calculation step, contact arrays of the upper and lower boundaries, as well as contact arrays of the right and left boundaries, are pairwise joined by means of the bitwise AND operator. If the joining result is not 0, percolation in the corresponding axis occurs. For open boundary conditions, the test of percolation conditions is reduced to the verification of the presence of nonzero bits in contact arrays of the upper and lower boundaries, as well as of the left and right boundaries. Thus, to test the percolation conditions, it is necessary to perform several logical operators (the number of operators is equal to the index m of contact arrays).

The appearance of an object touching the boundary, as well as the creation of contact arrays for this object, should be considered separately. This process differs for open and periodic boundary conditions. For open boundary conditions, the object is considered as touching the boundary if the boundaries of the object become beyond the calculation region (in the case of percolation on the lattice, the object is considered as touching the boundary, if the corresponding coordinate is equal to the

coordinate of the boundary). For periodic boundary conditions, an object is treated as touching the boundary only if it intersects with an object in the corresponding cell at the opposite boundary (in the case of percolation on the lattice, the corresponding cell at the opposite boundary should be occupied). In this case, the contact arrays for both clusters are rewritten and the corresponding bits are set to be 1 and the clusters themselves are not joined.

The sequence of actions implementing the proposed algorithm is briefly as follows:

(1) To create the contact arrays *top*, *left*, *bottom* and *right* and the stack of the removed elements of contact arrays.
(2) To generate a new object in the system and to treat it as a unit-size cluster.
(3) If an object touches the boundary, to create a new element for this object in contact arrays according to the above rules. To take the ordinal number of the new element from the stack of removed elements if the stack is not empty.
(4) For continuum percolation, to test the conditions of intersection of the new object with neighbors. For percolation on the lattice, to test the state of neighboring elements of the lattice (empty/occupied).
(5) If an object intersects with another object, to join clusters. In this case, the root site of the small cluster becomes an indicator to the root site of the large cluster, and their contact arrays are joined by means of the logical OR operator. The contact arrays for the small cluster is placed in the stack of removed elements. To perform this operation for all neighbors of the newly created object.
(6) For each new cluster, to test percolation conditions: *top*.AND.*bottom* and *left*. AND.*right*. A nonzero result means percolation in the corresponding direction.

It is noteworthy that the proposed algorithm makes it possible to test percolation conditions independently for each pair of boundaries and each type of boundary conditions. The formulation of the boundary conditions is reduced to the rules of the formation of contact arrays and test of the percolation conditions with the use of these arrays. This means that, within this approach, boundary conditions can be combined; in particular, open boundary conditions can be used for one direction and periodic boundary conditions can be used for another direction or both conditions can be tested for one or each of the directions. Furthermore, the proposed algorithm can be combined with the algorithm of search for clusters with an arbitrary winding number on a torus proposed in Ref. 15. Indeed, to find a cluster with an arbitrary winding number, it is necessary to calculate the path from each element of the cluster to the root site; contact arrays are not required in this case. Thus, the existence of clusters touching boundaries at an arbitrary point, clusters with one loop, and clusters with an arbitrary number of loops on the torus can be verified in one calculation.

To simultaneously verify the existence of clusters touching boundaries at an arbitrary point and clusters with one loop in the system, the algorithm should be slightly modified. In this situation, in addition to contact arrays, which will be used to test percolation conditions with periodic boundary conditions, it is necessary to

introduce additional four bits specifying the presence or absence of the contact of a cluster with four boundaries of the system. These bits are set to be 1 if the cluster touches the corresponding boundary. When clusters are joined, bits, as well as contact arrays, are joined by means of the logical OR operator. The contact arrays themselves are filled according to the rules formulated above for periodic boundary conditions. The existence of a cluster with one loop is verified according to the above scheme for contact arrays, whereas the verification of the existence of a cluster touching boundaries at an arbitrary point requires the convolution of the bits that were introduced above and refer to the opposite boundaries by means of the bitwise AND operator.

We also note that the introduction of contact arrays is necessary in the case of periodic boundary conditions. If only open boundary conditions are used at all boundaries, only four bits corresponding to the contact of a cluster with four boundaries can be introduced for each cluster instead of contact arrays. This can somewhat reduce the amount of main memory required for the calculation. The speed of the algorithm remains almost unchanged.

3. Results

To illustrate the operation of the algorithm, we consider the solution of the site problem on an 8×8 square lattice. Let two clusters of blue and red sites exist in the system at a certain step (Fig. 1). A site marked by a black circle is added to the system at the next step. Before the addition of the site, contact arrays for the red and blue clusters in the case of open boundary conditions contain the following values (in the binary notation): $top = 00100000$, $bottom = 0$, $left = 0$, and $right = 0$ for the blue cluster and $top = 0$, $bottom = 00000100$, $left = 0$ and $right = 0$ for the red cluster. After the addition of the black site, these clusters are joined into one cluster with the contact array containing the values $top = 00100000$, $bottom = 00000100$,

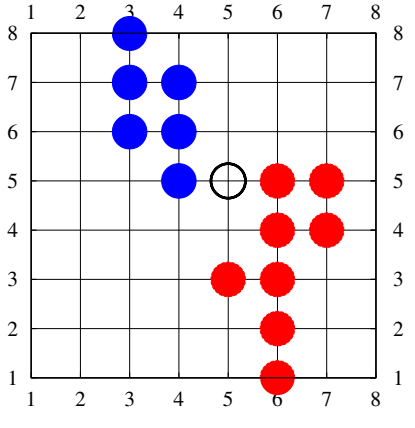

Fig. 1. (Color online) Problem of sites on an 8×8 square lattice. Example 1.

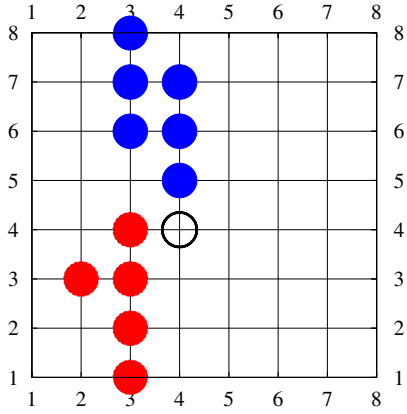

Fig. 2. (Color online) Problem of sites on an 8×8 square lattice. Example 1.

$left = 0$ and $right = 0$. The test of the percolation condition in the vertical axis is reduced to the verification of the presence of nonzero bits simultaneously in the arrays *top* and *bottom*. Since both arrays are nonzero, the verification gives a positive result. For periodic boundary conditions, both arrays *top* and *bottom* are zero and the percolation condition is not satisfied.

For the system shown in Fig. 2, before the addition of the black site, contact arrays for any type of boundary conditions have the form $top = 00100000$, $bottom = 0$, $left = 0$ and $right = 0$ for the blue cluster and $top = 0$, $bottom = 00100000$, $left = 0$ and $right = 0$ for the red cluster. The addition of the site results in the appearance of a cluster with contact arrays $top = 00100000$, $bottom = 00100000$, $left = 0$ and $right = 0$. In this situation, percolation conditions in the vertical axis are satisfied both for periodic boundary conditions and for open boundary conditions.

The proposed algorithm was implemented in the fortran90 language for the problem of two-dimensional continuum percolation of the fully penetrable discs of unit radius with periodic boundary conditions. In the operation process, memory for the array of elements, contact arrays and the stack of removed elements was used dynamically.

In Fig. 3, we show results from calculations of wrapping probability functions for percolation along a specified axis and along either axis. Wrapping probability is the probability that the one-loop cluster exists which wraps around a specified axis in the former case and along either axis in the latter case. Functions are calculated using the methods proposed in Refs. 3 and 8. According to these methods we stop each trial when wrapping clusters appear along both axes. Then we convolve obtained data with the Poisson distribution.

Estimated critical filling fractions are shown in Fig. 4. Critical filling fraction converges to the percolation threshold value $\eta_c = 1.1280(8)$ for the continuum percolation of disks.[8]

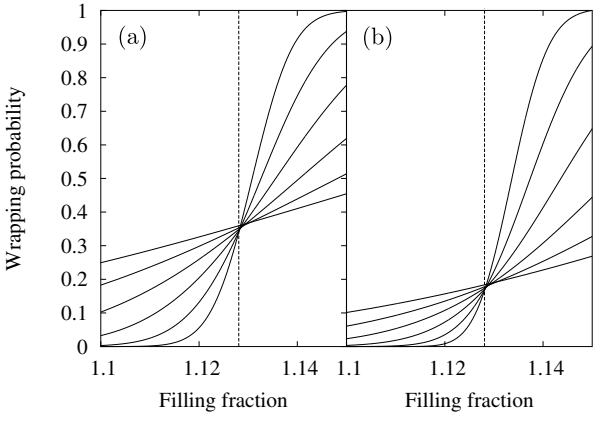

Fig. 3. Plots of the cluster wrapping probability functions in the region of the percolation for percolation (a) along a specied axis, (b) along either axis.

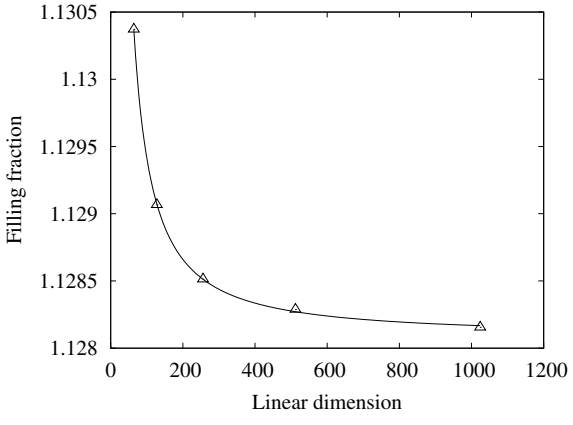

Fig. 4. Estimated critical filling fractions for continuum percolation of disks.

The efficiency of the algorithm was determined by comparing the time of one calculation with the calculation performed by the method proposed in Ref. 15 and the method of the determination of the percolation threshold, which implies the passage of all boundary cells and test of intersection conditions. Each calculation was performed until the achievement of a certain fixed number of objects in the system and at the same initialization of a pseudorandom number generator.

Figure 5 shows the dependences of the calculation time on the area of the system (square of the linear size) for three algorithms: the Machta algorithm proposed in Ref. 15, the algorithm proposed in this work, and the standard algorithm for testing periodic boundary conditions, which implies the successive test of elements in all boundary cells. It is seen that the proposed algorithm is

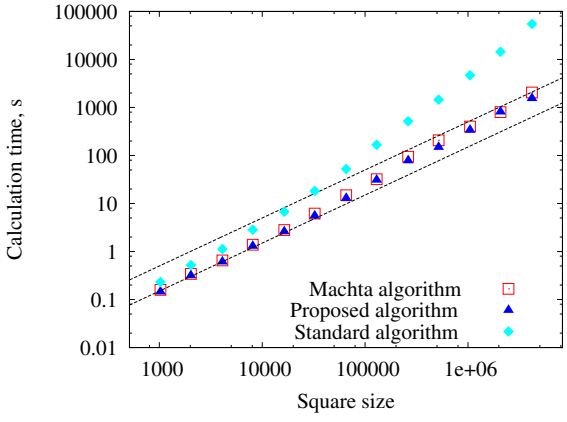

Fig. 5. (Color online) Comparison of the capacities of the algorithms.

comparable in efficiency to the Machta algorithm, i.e. requires the calculation time $O(N)$. Deviation from linearity is due to the effect of the use of long-access memory with an increase in the necessary amount of memory, which was mentioned in Ref. 8.

Moreover, for all sizes of the system, the operation time of the standard algorithm is much longer than the operation time of the proposed algorithm. With an increase in the size of the system, the operation time of the standard algorithm increases much more rapidly than the operation time of the proposed algorithm. This is because of the necessity of numerous operations for testing percolation conditions in the axes. Percolation conditions are tested if clusters are joined. This means that percolation conditions for the system near the percolation threshold should be tested at almost each calculation step of the addition of a new object to the system; as a result, the calculation time increases significantly.

4. Conclusions

A new algorithm to test percolation conditions for problems of continuum and lattice percolation has been proposed with the simulation of the percolation process within the Newman–Ziff algorithm. This algorithm makes it possible to verify the existence of both clusters touching boundaries at an arbitrary point (open boundary conditions) and one-loop clusters satisfying periodic boundary conditions. The test of percolation conditions requires a few bitwise operators and insignificantly affects the calculation time with the Newman–Ziff algorithm, which is linear in the number of objects in the system. The proposed algorithm can be combined with the Machta algorithm; as a result, the existence of clusters satisfying open and periodic boundary conditions, one-loop clusters, and clusters with an arbitrary winding number in the system can be verified in one calculation.

Acknowledgments

Author would like to thank V.D. Borman, V.N. Tronin and A.M. Grekhov for helpful comments. This work was partly supported by the Russian Foundation for Basic Research (grant 14-08-00805 a).

References

1. H. T. Pinson, *J. Stat. Phys.* **75**, 1167 (1994).
2. J. L. Cardy, *J. Phys. A*, *Math. Gen.* **25**, L201 (1992).
3. M. E. J. Newman and R. M. Ziff, *Phys. Rev. E* **64**, 016706 (2001).
4. J. Hoshen and R. Kopelman, *Phys. Rev. B* **14**, 3438 (1976).
5. G. Pruessner and N. R. Moloney, *J. Stat. Phys.* **115**, 839 (2004).
6. P. Kleban and R. M. Ziff, *Phys. Rev. B* **57**, R8075 (1998).
7. P. M. C. de Oliveira, R. A. Nóbrega and D. Stauffer, *Braz. J. Phys.* **33**, 616 (2003).
8. S. Mertens and C. Moore, *Phys. Rev. E* **86**, 061109 (2012).
9. J. Li and S.-L. Zhang, *Phys. Rev. E* **80**, 040104 (2009).
10. S. B. Lee and S. Torquato, *Phys. Rev. A* **41**, 5338 (1990).
11. J. Li and M. Östling, *Phys. Rev. E* **88**, 012101 (2013).
12. V. Sasidevan, *Phys. Rev. E* **88**, 022140 (2013).
13. H. Hu, H. W. J. Blte and Y. Deng, *J. Phys. A*, *Math. Theor.* **45**, 494006 (2012).
14. H. Yang, *Phys. Rev. E* **85**, 042106 (2012).
15. J. Machta, Y. S. Choi, A. Lucke, T. Schweizer and L. M. Chayes, *Phys. Rev. E* **54**, 1332 (1996).