Available online at www.sciencedirect.com
SCIENCE DIRECT®
Physica A 321 (2003) 665–678
PHYSICA A
www.elsevier.com/locate/physa

Extension of Hoshen–Kopelman algorithm to non-lattice environments

Ahmed Al-Futaisi[a,b], Tadeusz W. Patzek[a,*]

[a] *Department of Civil & Environmental Engineering, University of California, Berkeley, CA 94720, USA*
[b] *Civil Engineering Department, Sultan Qaboos University, Alkhoudh 123, Oman*

Received 14 May 2002

Abstract

We extend the Hoshen–Kopelman (HK) algorithm for cluster labeling to non-lattice environments, where sites are placed at random at non-lattice points. This extension is useful for continuum systems and disordered networks. Our extension of the HK algorithm relies on several data structures that describe network connectivity regardless of its dimensionality. Just as for the classic HK algorithm on lattices, our extension is completed in a single pass through the sites of the network and cluster relabeling operates on a vector whose size is much smaller than the size of the network. Our extension of the HK algorithm works for any environment (lattice or non-lattice) of any dimensionality, type (sites, bonds or both), and with arbitrary connectivity between the sites. The proposed extension is illustrated through a simple network consisting of 16 sites and 24 bonds, and applied to a complex network extracted from a 3D micro-focused X-ray CT image of Bentheimer sandstone consisting of 3677 sites and 8952 bonds.

PACS: 02.70.−c; 05.40.+j

Keywords: Cluster labeling; Hoshen–Kopelman algorithm; Non-lattice; Disordered networks; Continuum systems

1. Introduction

The introduction of the Hoshen–Kopelman (HK) algorithm [1] in 1976 was an important breakthrough in the analysis of cluster size statistics in percolation

* Corresponding author. Tel.: +1-510-643-5834.
E-mail address: patzek@patzek.berkeley.edu (T.W. Patzek).

0378-4371/03/$ - see front matter
PII: S0378-4371(02)01586-8

theory. Only after the introduction of this algorithm, did Monte-Carlo simulations of very large lattices become possible [2–4]. The algorithm's single and sequential pass through the lattice linearizes the time and memory space requirement as a function of the lattice size [5]. Although the algorithm was initially applied in statistical physics, nowadays it is applied in many diverse fields [6–10]. The algorithm is not restricted to pure site or pure node lattices; it has been extended to lattices that consist of sites and bonds [11–13]. In the enhanced Hoshen–Kopelman (EHK) algorithm [14], the HK algorithm has been extended to determine information not only on the cluster size but also on the structure of the clusters, such as the radius of gyration, internal perimeter, or spatial moments. The EHK algorithm has been applied to large images [15], and used to calculate cluster properties and entropy in percolation models [16]. Different approaches have been proposed to parallelize the HK algorithm [17–20].

The majority of the HK algorithm adaptations and implementations were for lattice environments (discrete systems). Only a few studies were devoted to discussing the HK algorithm implementation in non-lattice environments (continuous systems). In such systems, the positions of the sites are arbitrary, and not restricted to the discrete points of a regular lattice. J. Hoshen comments:[1] "*... The HK algorithm can also be used for a non-lattice environment, where sites are placed at random at non-lattice points, but it is more complicated...*". Gawlinski and Stanley [21] were the first to adapt the HK algorithm to handle continuum percolation by overlaying an imaginary covering mesh onto a square area. Their adaptation was implemented in other continuum percolation models on discs and spheres [22,23]. Non-lattice environments exist not only in the percolation theory of disordered discs and spheres, but also in the networks of many interacting units that are observed in complex systems. Researchers are only now beginning to unravel the structure and dynamics of such complex networks [24].

This paper describes an extension of the HK algorithm to complex networks. Such networks can be lattices or non-lattices in two, three or higher-dimensions; they can consist of sites, bonds or both; and each site can have a different number of connecting bonds. The data structures we introduce here describe an arbitrary network. Clusters of sites, bonds, or sites-and-bonds in such a network can be labeled with our implementation of the HK algorithm.

The paper is organized as follows: First, we present the data structures that suffice to describe any network. Then, the HK algorithm implementation that uses these data structures is proposed. After that, the results of labeling a simple network consisting of sites and bonds are reported. Then, we discuss the applicability of our HK algorithm implementation to networks consisting of sites or bonds only. Finally, we apply

[1] Personal communication by e-mail, July 25, 2001.

our procedure to a complex network extracted from the 3D image of a real porous rock.

2. Data structures

Applying the HK algorithm to a non-lattice environment becomes easier if we arrange the network description in a certain format. This format should be the same for a 2D lattice, 3D lattice, and a non-lattice environment, where sites are placed at random at non-lattice points. Moreover, network traversal does not need to be sequential from one fixed direction to another, but ought be carried out from any point in the network to any other point in the network. With these two considerations in mind, we present the data format we believe is useful when using the HK algorithm in an arbitrary environment. This format was the main reason for our successful implementation of the HK algorithm to cluster labeling of complex disordered networks that consist of nodes and links placed at irregular points in three dimensions.

The best way to describe the proposed data structures is through an illustrative example. For simplicity, we consider a small disordered network of 16 sites (nodes) and 24 bonds (links), Fig. 1. In order to avoid sequential implementation of the HK algorithm, the nodes and links are labeled at random. The black elements (nodes or links) are occupied, whereas the gray elements are empty.

The network information is split into two sets of data. The first one is related to the nodes and the second to the links.

The node information is stored in three arrays: `Node`, `NodeS`, and `NodeNext`. The array `Node` contains the (arbitrary) labels of each node in the network. Although we arrange the nodes in an increasing order, the algorithm works with any node ordering. The occupancy of each node is stored in the array `NodeS`. If the node is

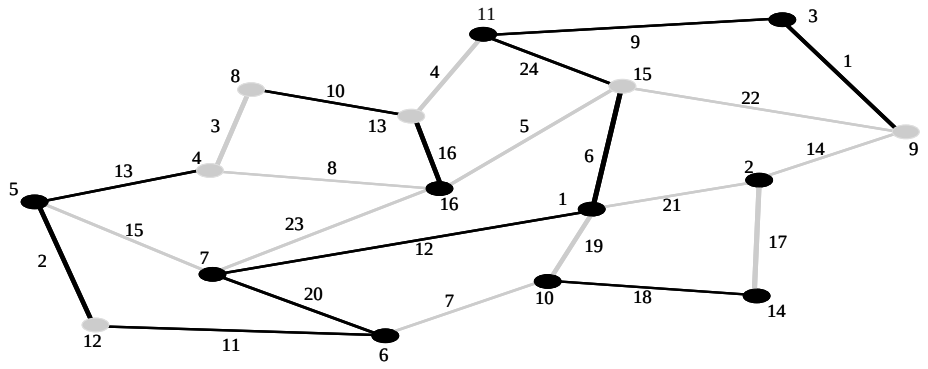

Fig. 1. Simple network of 16 nodes and 24 links (occupied elements are black, whereas gray elements are empty).

occupied, its NodeS is one, otherwise it is zero. Again, the numbers one and zero are selected arbitrarily. The connectivity of the nodes in the network is described through the NodeNext array. In this array, we define the neighboring nodes that are directly connected to each node (for example, node 7 is connected to nodes {5,16,1,6}). The size of this array is the total number of nodes in the network by the maximum connectivity or coordination numbers (4 in this network). Therefore for the nodes that have coordination numbers less than the maximum, we define their appropriate neighboring nodes and assign zero to the remaining array elements.

$$\text{Node} = \begin{vmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{vmatrix} \quad \text{NodeS} = \begin{vmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{vmatrix} \quad \text{NodeNext} = \begin{vmatrix} 7 & 15 & 2 & 10 \\ 1 & 9 & 14 & 0 \\ 11 & 9 & 0 & 0 \\ 8 & 16 & 5 & 0 \\ 4 & 7 & 12 & 0 \\ 12 & 7 & 10 & 0 \\ 5 & 16 & 1 & 6 \\ 13 & 4 & 0 & 0 \\ 15 & 3 & 2 & 0 \\ 6 & 1 & 14 & 0 \\ 13 & 3 & 15 & 0 \\ 5 & 6 & 0 & 0 \\ 8 & 11 & 16 & 0 \\ 2 & 10 & 0 & 0 \\ 16 & 11 & 9 & 1 \\ 4 & 13 & 15 & 7 \end{vmatrix}$$

The second set of data is related to the links of the network. Similarly to the nodes, the link information is stored in three arrays: Link, LinkS, LinksOfNode:

In Link, the labels of the links are stored. The array LinkS describes the occupancy of each link. If the link is occupied, we assign one, otherwise, we assign zero. In LinksOfNode, we define the links that are attached to each node. For consistency, we report the attached links of a node in LinksOfNode in the sequence already used to report the neighboring nodes in NodeNext (for example, node 7 is attached to links {15,23,12,20}). Therefore, the size of the array LinksOfNode is identical to the size of the array NodeNext. With these data structures in hand, we are ready to describe our implementation of the HK

algorithm.

$$\text{Link} = \begin{vmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \end{vmatrix} \quad \text{LinkS} = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{vmatrix} \quad \text{LinksOfNode} = \begin{vmatrix} 12 & 6 & 21 & 19 \\ 21 & 14 & 17 & 0 \\ 9 & 1 & 0 & 0 \\ 3 & 8 & 13 & 0 \\ 13 & 15 & 2 & 0 \\ 11 & 20 & 7 & 0 \\ 15 & 23 & 12 & 20 \\ 10 & 3 & 0 & 0 \\ 22 & 1 & 4 & 0 \\ 7 & 19 & 18 & 0 \\ 4 & 9 & 24 & 0 \\ 2 & 11 & 0 & 0 \\ 10 & 4 & 16 & 0 \\ 17 & 18 & 0 & 0 \\ 5 & 24 & 22 & 6 \\ 8 & 16 & 5 & 23 \end{vmatrix}$$

3. The HK algorithm implementation

Here we implement the HK algorithm with the data structures described above. Even though an arbitrary network consists of nodes and links, we will traverse the network only by the nodes. Each node in the network is scanned based on its order in the

array Node. In what follows, our implementation of the HK algorithm is described in seven steps. In each step, we record the result of the step when applied to the network shown in Fig. 1. The description of the algorithm is also translated into the MATLAB programming language in Appendix A. We have chosen MATLAB for its wide use and clarity of the code. The HK algorithm implementation for regular lattices can be found in other computer languages [25–27]. Our implementation of the HK algorithm is:

1. Read the network data structures: Node, NodeS, NodeNext, Link, LinkS, LinksOfNode.
2. The arrays NodeL and LinkL are created with same sizes as Node, and Link, respectively, and entries equal to all zeros. NodeL stores node labels, whereas LinkL stores link labels.
   $\mathtt{NodeL} = \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}$
   $\mathtt{LinkL} = \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}$
3. Create an empty array NodeLP. This array holds information about the cluster labels, and is the essence of the HK algorithm. After scanning the entire network, we work only with NodeLP. We do not create a similar array for the links because we traverse the network only by the nodes.
   $\mathtt{NodeLP} = []$
4. Scan the network nodes following their order in the array Node. At each node, i, as we traverse the network, elements in NodeL, and NodeLP are changed according to the criteria below:
   (a) If the node is unoccupied ($\mathtt{NodeS}(i) = 0$), go on to the next node. In the network in Fig. 1, the nodes {4,8,9,12,13,15} are all of this type.
   (b) If the node is occupied ($\mathtt{NodeS}(i) = 1$) but does not have any links together with their adjacent nodes that are occupied, such as {2,5,16}, then start a new cluster. Record this new cluster in NodeL, and NodeLP.
   (c) If the node is occupied ($\mathtt{NodeS}(i) = 1$), and in addition it has at least one link and its corresponding adjacent node that both are occupied, e.g., {1,3,6,7,10, 11,14}, then we define two possibilities:
      (i) If none of the neighboring nodes is labeled, then start a new cluster and update NodeL, and NodeLP, accordingly.
      (ii) If there exists a labeled neighbor, then set NodeL of the node and NodeLP of the neighboring occupied nodes equal to the minimum of NodeLP(NodeLP) of the neighboring occupied nodes.
      $\mathtt{NodeLP} = [1,2,3,4,1,6,7]$
      $\mathtt{NodeL} = \{1,2,3,0,4,5,1,0,0,6,3,0,0,6,0,7\}$
5. After the scan in Step 4 is completed, operate on the array NodeLP only:
   (a) Correct labels in NodeLP recursively so that connected nodes (i.e., occupied nodes connected to occupied links and nodes) have the same cluster number.
   $\mathtt{NodeLP} = [1,2,3,4,1,6,7]$
   (b) Renumber labels in NodeLP to be sequential.
   $\mathtt{NodeLP} = [1,2,3,4,1,5,6]$

6. Apply the corrected labels in NodeLP to the arrays NodeL and LinkL.
   NodeL = {1, 2, 3, 0, 4, 1, 1, 0, 0, 5, 3, 0, 0, 5, 0, 6}
   LinkL = {3, 4, 0, 0, 0, 1, 0, 0, 3, 0, 1, 1, 4, 0, 0, 6, 0, 5, 0, 1, 0, 0, 0, 3}
7. Finally, label the clusters that consist of a single link.
   LinkL = {3, 4, 0, 0, 0, 1, 0, 0, 3, 7, 1, 1, 4, 0, 0, 6, 0, 5, 0, 1, 0, 0, 0, 3}

The application of the HK algorithm to the network in Fig. 1 results in seven clusters

Cluster	Nodes	Links
1	1, 6, 7	6, 11, 12, 20
2	2	
3	3, 11	1, 9, 24
4	5	2, 13
5	10, 14	18
6	16	16
7		10

It is clear that our extension of the original HK algorithm is generic. The proposed input data structures, NodeNext and LinksOfNode, make the algorithm work similarly in lattices and non-lattices in two, three or more dimensions, and in site, bond, or site-and-bond environments. Our extension examines the nodes in a network in a single sweep. The relabeling steps are sequential and recursive, and operate on a small vector, NodeLP (much smaller than the size of the entire network). Therefore, when applied to lattices, the proposed extension is similar to the original HK algorithm. All inner operations in the cluster labeling algorithm, label sorting, and renumbering can be vectorized.

4. Networks consisting of nodes or links only

In networks consisting of nodes only, we can assume that all links are occupied (i.e., their LinkS entries are equal to one) and then run the algorithm described in Section 3. For the output, we read the array NodeL only. If we do not want to construct the LinksOfNode array and use the NodeNext only, minor changes in the algorithm are necessary. In Step 1, read the data Node, NodeS, and NodeNext. In Steps 2 and 3, initialize the variables NodeL, NodeLP. Then for traversing the network in Step 4, apply condition 4a if the node is unoccupied; or condition 4b if the node is occupied but has no labeled neighboring nodes in NodeNext; and finally condition 4c is applied if the node is occupied and has at least one labeled neighboring node in NodeNext. There are no necessary changes in Step 5 because it deals with the array NodeLP. In Step 6, apply the correct labels to the array NodeL and ignore LinkL. Step 7

is now not necessary. If we imagine the network in Fig. 1 as consisting of nodes only, there are two clusters reported in `NodeL` as {1,1,2,0,1,1,1,0,0,1,2,0,0,1,0,1}. The same treatment can be applied to networks consisting of links only. We can either run the algorithm described in Section 3 by assuming that all nodes are occupied (`NodeS` = 1), or apply the above changes to the algorithm by considering links only. If the network in Fig. 1 consists of links only, there are four clusters reported in `LinkL` as {1,1,0,0,0,1,0,0,1,3,1,1,1,0,0,3,0,4,0,1,0,0,0,1}.

5. Application: a pore network extracted from 3D images of a porous medium

The second example uses a network, Fig. 2, that represents the void space of a real rock. The network was extracted from a 3D micro-focused X-ray CT image and represents a $(2.5\ \text{mm})^3$ sample of Bentheimer sandstone.[2] It consists of 3677 nodes and 8952 links. Due to the network size, we plot only the links. The network connectivity varies from zero to sixteen links (pore throats) connected to a node (a pore body).

In the network, the complex and variable cross-sections of pore bodies and pore throats are approximated with cylindrical channels of arbitrary triangular, rectangular and circular cross-sections. Table 1 summarizes the important features of the network. More details can be found in Refs. [28,29].

The Statoil network was used to simulate two-phase, immiscible flow of a non-wetting fluid (oil) and a wetting fluid (water). In porous media, the fluid flow paths are determined by the medium, but the medium itself is in some sense random [30]. In two-phase flow, the invading fluid must be connected to the inlet to continue invading and the defending fluid must be connected to the outlet to be displaced. This dynamic

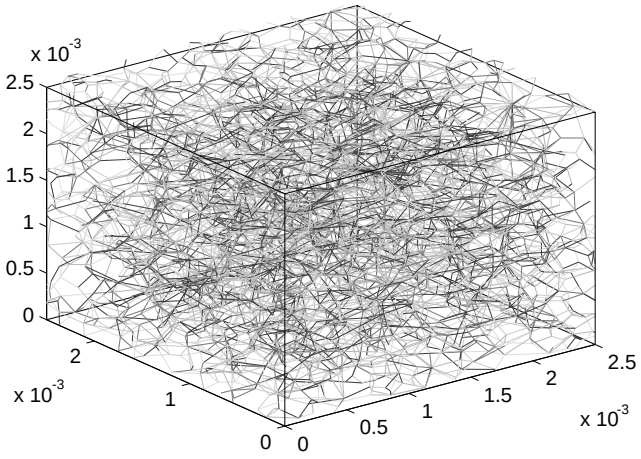

Fig. 2. 3D network representation of Bentheimer sandstone.

[2] The network was obtained from Dr. Paal-Erik Øren of Statoil.

Table 1
Summary of statistics of the network and its pore bodies and throats

Network dimensions (mm): 2.55 × 2.55 × 2.55
Porosity: 0.2342
Microporosity: 0.0141

Pore bodies (3677 Nodes)
Cross-sections: Triangles: 3496 Squares: 146 Cylinders: 35
Total node volume (mm^3): 1.6856
Total microporosity volume (mm^3): 0.2343

	Min	Mean	Max	S.D.	C.V.
Volume (mm^3)	5.991E − 6	4.5842E − 4	0.0011	1.828E − 4	0.3989
Area (mm^2)	3.118E − 6	8.8017E − 3	0.0330	6.4847E − 3	0.7368
Length (mm)	1.992E − 3	6.3436E − 2	0.1816	2.8901E − 2	0.4556
Inscribed circle radius (mm)	9.962E − 4	3.1718E − 2	0.0908	1.4450E − 2	0.4556
Shape factor	1.163E − 2	3.4399E − 2	0.0796	9.4198E − 3	0.2738
Microporosity volume (mm^3)	0	6.3717E − 5	0.0004	5.2514E − 5	0.8242
Number of corners	1	3	4	0.27816	0.0927

Pore links (8925 Links)
Cross-sections: Triangles: 6977 Squares: 1159 Cylinders: 789
Total link volume (mm^3) : 1.9638

	Min	Mean	Max	S.D.	C.V.
Volume (mm^3)	5.3917E − 6	2.200E − 4	0.0011	1.5564E − 4	0.7073
Area (mm^2)	2.5254E − 6	2.3922E − 3	0.0216	2.9018E − 3	1.2130
Length (mm)	9.9619E − 4	7.3909E − 2	0.1593	2.2369E − 2	0.3027
Inscribed circle radius (mm)	2.0170E − 4	1.4950E − 2	0.0619	9.5657E − 3	0.6399
Shape factor	5.4243E − 4	4.0112E − 2	0.0796	1.7643E − 2	0.4398
Number of corners	1	3	4	0.69377	0.2313

Connectivity
Number of inlet links: 183
Number of outlet links: 208

	Min	Mean	Max	S.D.	C.V.
Coordination number	1	4	16	1.8867	0.3673

Shape factor = (cross-section Perimeter)^2/area. S.D. = Standard deviation. C.V. = Coefficient of variation.

percolation process is called invasion percolation [31]. If clusters of the defending fluid can become disconnected from the outlet, we call this process percolation with trapping [32]. We have simulated two different processes: *primary drainage* in which oil invades water, and *secondary imbibition* in which water invades the oil. Primary drainage is a pure bond invasion-percolation problem, whereas secondary imbibition is a problem in mixed (bond-and-site) invasion-percolation and ordinary-percolation with trapping. By the end of secondary imbibition, water disconnects some of the oil-filled links and nodes from the outlet, and traps numerous clusters of oil. It is useful to study the

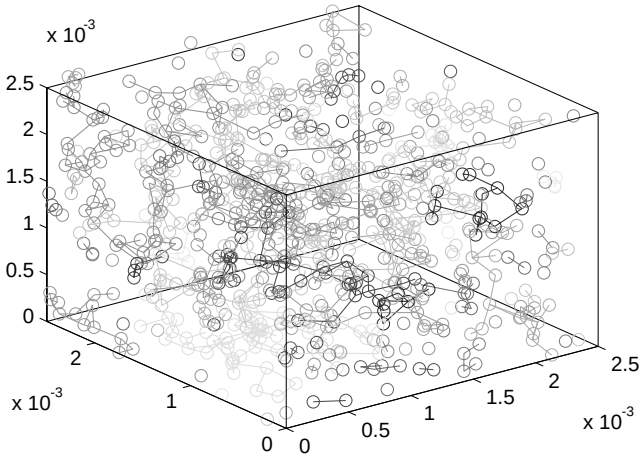

Fig. 3. Clusters of trapped oil.

spatial distribution and sizes of these trapped oil clusters. We have used our extended HK algorithm for disordered networks to identify the oil clusters trapped in secondary imbibition. In Fig. 3, we show 230 clusters of oil trapped for a single range of advancing contact angles. More details of the simulation can be found elsewhere [33].

6. Conclusions

The Hoshen–Kopelman (HK) algorithm has been extended to non-lattice environments, where network elements (sites or bonds) are positioned at random points in space. The extension has been facilitated by the use of two data structures, `NodeNext` and `LinksOfNode`, that describe the connectivity of an arbitrary network. Our extension of the HK algorithm is not restricted to a non-lattice environment, and can be applied to lattices in two, three or higher dimensions. It can also be applied to networks of sites, bonds, or sites-and-bonds. The new algorithm has been used to study the sizes of oil clusters trapped in a disordered pore network extracted from a 3D micro-focused X-ray CT image of Bentheimer sandstone. The proposed extension of the HK algorithm should be useful in the studies of percolation in continuum systems and in the detection of fluid clusters in more realistic networks extracted from complex rocks.

Appendix A. MATLAB Function HKNonLattice.m

Here, we provide the complete version of the MATLAB function that performs our implementation of the HK algorithm. Using the data structures proposed in this paper as an input, the function calculates cluster statics in lattices and complex disordered networks.

```
function [NumberOfClusters, NodeL, LinkL]=HKNonLattice(NodeS, LinkS,...
                                                NodeNext, LinksOfNode, OFlag)
%=========================================================
%
%  Adaptation of Hoshen--Kopelman cluster labeling algorithm for disordered
%  networks (nodes and links are placed at random in space)
%
%  Input arguments:
%
%    NodeS              =Occupancy state of nodes
%    LinkS              = Occupancy state of links
%    NodeNext           = Neighboring nodes connected to each node
%    LinksOfNode        = Links attached to each node
%    OFlag              = Flag for occupied nodes and links (1 in this paper)
%
%  Output arguments:
%
%    NumberOfClusters   = Number of occupied clusters
%    NodeL              = Cluster labels of nodes
%    LinkL              = Cluster labels of links
%
%  By: Ahmed AL-Futaisi and Tadeusz Patzek
%
%=========================================

%
%  STEP 1: READ THE DATA AND INITIALIZE THE OUTPUT
%
NumberOfNodes    = length(NodeS);      % Number of nodes in current network
NumberOfLinks    = length(LinkS);      % Number of links in current network
NumberOfClusters = 0;

%
%  STEP 2: INITIALIZE THE HK ALGORITHM VARIABLES NodeL and LinkL
%
NodeL=zeros(NumberOfNodes,1);      % Array to store cluster labels of nodes
LinkL=zeros(NumberOfLinks,1);      % Array to store cluster labels of links


%
%  STEP 3: CREATE EMPTY ARRAY NodeLP AND START CLUSTER COUNTER
%
NodeLP=[];         % Array used for relabeling steps
Cluster=0;         % Cluster counter

%
%  STEP 4: SCAN THE NETWORK NODES
%
for i=1:NumberOfNodes
    %
    % Check if the node (Case 4c):
    %   1. has OFlag occupancy
    %   2. has both NodeNext and LinksOfNode that have OFlag occupancy
    N=find((NodeS(i)==OFlag).*(NodeS(nonzeros(NodeNext(i,:)))==OFlag).*...
          (LinkS(nonzeros(LinksOfNode(i,:)))==OFlag));
    %
```

```
    if (~isempty(N))
       %
       % Define the occupancy status of NodeNext
       %
       Nodes=NodeNext(i,N);
       NodeNextL=NodeL(Nodes);
       %
       % Case 4c i: No labeled neighbour
       %
       if any(NodeNextL)==0   % Start a new cluster
             Cluster=Cluster+1;
             NodeL(i)=Cluster;
             NodeLP(end+1)=Cluster;
       %
       % Case 4c ii: There exists a labeled neighbor
       %
       else % Put in the minimum labeling
             N=(NodeLP(nonzeros(NodeNextL)));
             NodeLPmin=min(NodeLP(N));
             NodeL(i)=NodeLPmin;
             NodeLP(N)=NodeLPmin;
       end
    %
    % This node is type 4b:
    elseif NodeS(i)==OFlag
       Cluster=Cluster+1; % Start a new cluster
       NodeL(i)=Cluster;
       NodeLP(end+1)=Cluster;
    end
    %
    % Skip nodes that are type 4a
end
%
%   STEP 5A: CORRECT LABELS IN NodeLP RECURSIVELY
%
for i=1:length(NodeLP)
       N=i;
       while (NodeLP(N)<N)
          N=NodeLP(N);
       end
       NodeLP(i)=N;
end
%
%   STEP 5B: RENUMBER LABELS IN NodeLP TO RUN SEQUENTIALLY
%
NodeLP1=sort(NodeLP);
RelabL=NodeLP1(2:end).*(NodeLP1(2:end) > NodeLP1(1:end-1));
RelabL=[NodeLP1(1), nonzeros(RelabL)'];
for i=1:length(RelabL)
       NodeLP(find(NodeLP==RelabL(i)))=i;
end
%
%   STEP 6: APPLY THE CORRECT LABELS TO THE ARRAYS NodeL AND LinkL
%
```

```
for i=1:length(NodeLP)
     N=nonzeros(LinksOfNode(find(NodeL==i),:));
     LinkL(nonzeros(N.*(LinkS(N)==OFlag)))=NodeLP(i);
     NodeL(find(NodeL==i))=NodeLP(i);
end
%
%   STEP 7: FINALLY, LABEL THE CLUSTERS THAT CONSIST OF SINGLE LINKS
%
SingleLink=find(LinkL==0 & LinkS==OFlag);
if ~isempty(SingleLink)
     NewCluster=max(max(NodeL),max(LinkL))+1
     NewCluster=[NewCluster:NewCluster+(length(SingleLink)-1)]
     LinkL(SingleLink)=NewCluster;
end
%
%   RECORD NUMBER OF CLUSTERS
%
NumberOfClusters=max(max(NodeL),max(LinkL));
%
return
```

References

[1] J. Hoshen, R. Kopelman, Percolation and cluster distribution I. Cluster multiple labeling technique and critical concentration algorithm, Phys. Rev. B 14 (8) (1976) 3438–3445.

[2] A. Margolina, H. Nakanishi, D. Stauffer, H.E. Stanley, Monte Carlo and series study of corrections to scaling in two-dimensional percolation, J. Phys. A 17 (1984) 1683–1701.

[3] D.C. Rapaport, Cluster number scaling in two-dimensional percolation, J. Phys. A 19 (1986) 291–304.

[4] D.C. Rapaport, Cluster size distribution at criticality, J. Stat. Phys. 66 (1986) 679–682.

[5] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[6] F. Eddi, J. Mariani, G. Waysand, Transient synaptic redundancy in the developing cerebellum and isostatic random stacking of hard spheres, Biol. Cybern. 74 (2) (1996) 139–146.

[7] S. Hamimov, M.A. Michalev, A. Savchenko, O.I. Yordanov, Classification of radar signatures by autoregressive model-fitting and cluster-analysis, IEEE Trans. Geosci. Remote Sens. 27 (5) (1989) 606–610.

[8] J.H. Kinney, D.L. Haupt, M.C. Nichols, T.M. Breunig, G.W. Marshall, S.J. Marshall, The X-Ray tomographic microscope: 3-dimensional perspectives of evolving microstructures, Nucl. Instrum. Methods A 347 (1–3) (1994) 480–486.

[9] R.G. Moreira, M.A. Barrufetb, Spatial distribution of oil after deep-fat frying of tortilla chips from a stochastic model, J. Food Eng. 27 (3) (1996) 279–290.

[10] L. Zhang, N.A. Seaton, Simulation of catalyst fouling at the particle and reactor levels, Chem. Eng. Sci. 51 (12) (1996) 3257–3272.

[11] P.R.A. Campos, L.F.C. Pessoa, F.G. Brady Moreira, Cluster-size statistics of site-bond-correlated percolation models, Phys. Rev. B 56 (1) (1997) 40–42.

[12] J. Hoshen, Percolation analog for a two component liquid vapor system, Chem. Phys. Lett. 75 (1980) 347–349.

[13] J. Hoshen, P. Klymko, R. Kopelman, Percolation and cluster distribution III. Algorithm for the site-bond problem, J. Stat. Phys. 21 (5) (1979) 583–600.

[14] J. Hoshen, M.W. Berry, K.S. Minser, Percolation and cluster structure parameters: the enhanced Hoshen–Kopelman algorithm, Phys. Rev. B 56 (2) (1997) 1455–1460.

[15] J. Hoshen, On the application of the enhanced Hoshen–Kopelman algorithm for image analysis, Pattern Recogn. Lett. 12 (1998) 575–584.

[16] I.J. Tsang, I.R. Tsang, D. Van Dyck, Cluster diversity and entropy on the percolation model: the lattice animal identification algorithm, Phys. Rev. B 62 (5) (2000) 6004–6014.
[17] A.N. Burkitt, D.W. Heermann, Parallelization of a cluster algorithm, Comput. Phys. Commun. 54 (2–3) (1989) 201–209.
[18] J.M. Constantin, M.W. Berry, B.T. VanderZanden, Parallelization of the Hoshen–Kopelman algorithm using a finite state machine, Int. J. Supercomput. Appl. High Perform. Comput. 11 (1) (1997) 34–48.
[19] M. Flanigan, P. Tamayo, Parallel cluster labeling for large-scale Monte-Carlo simulations, Physica A 215 (4) (1995) 461–480.
[20] J.M. Teuler, J.C. Gimel, A direct parallel implementation of the Hoshen–Kopelman algorithm for distributed memory architectures, Comput. Phys. Commun. 130 (1–2) (2000) 118–129.
[21] E.T. Gawlinski, H.E. Stanley, Continuum percolation in two dimensions: Monte Carlo tests of scaling and Universality for non-interacting discs, J. Phys. A 14 (1981) L291–L299.
[22] S.B. Lee, S. Toruato, Monte Carlo study of correlated continuum percolation: Universality and percolation thresholds, Phys. Rev. A 41 (10) (1990) 5338–5344.
[23] B. Lorenz, I. Orgzall, H.O. Heuer, Universality and cluster structures in continuum models of percolation with two different radius distributions, J. Phys. A 26 (1993) 4711–4722.
[24] S.H. Strogatz, Exploring complex networks, Nature 410 (2001) 268–276.
[25] R.J. Gaylord, P.R. Wellin, Computer Simulations with Mathematica-Explorations in Complex Physical and Biological Systems, Springer, Santa Clara, 1995.
[26] H. Goult, J. Tobochnik, An Introduction to Computer Simulation Methods: Applications to Physical Systems, 2nd Edition, Addison-Wesley, Reading, MA, 1996.
[27] D. Stauffer, A. Aharony, Introduction to Percolation Theory, 2nd Edition, Taylor and Francis Ltd, London, 1994.
[28] T.W. Patzek, Fundamentals of Multiphase Flow in Porous Media, 1st Edition, Berkeley, U.C. Berkeley, 1998.
[29] T.W. Patzek, Verification of a complete pore network simulator of drainage and imbibition, Soc. Petro. Eng. J. 6 (2001) 144–156.
[30] P.G. de Gennes, Percolation—A new unifying concept, La Recherche 7 (1976) 919–927.
[31] D. Wilkinson, J.F. Williamsen, Invasion percolation: a new form of percolation theory, J. Phys. A 16 (1983) 3365–3376.
[32] M.M. Dias, D. Wilkninson, Percolation with trapping, J. Phys. A 19 (1986) 3131–3146.
[33] A. Al-Futaisi, T.W. Patzek, The impact of wettability alteration on two-phase flow characteristics of sandstones—A quasi-static description, Water Resour. Res. 2002, accepted for publication.