UC Berkeley
UC Berkeley Previously Published Works

Title
A Fast Algorithm for Invasion Percolation

Permalink
https://escholarship.org/uc/item/8zp378fg

Journal
Transport in Porous Media, 102(2)

ISSN
0169-3913 1573-1634

Authors
Masson, Yder
Pride, Steven R

Publication Date
2014-02-04

DOI
10.1007/s11242-014-0277-8

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Transp Porous Med
DOI 10.1007/s11242-014-0277-8

A Fast Algorithm for Invasion Percolation

Yder Masson · Steven R. Pride

Received: 15 October 2013 / Accepted: 15 January 2014
© Springer Science+Business Media Dordrecht (outside the USA) 2014

Abstract We present a computationally fast Invasion Percolation (IP) algorithm. IP is a numerical approach for generating realistic fluid distributions for quasi-static (i.e., slow) immiscible fluid invasion in porous media. The algorithm proposed here uses a binary-tree data structure to identify the site (pore) connected to the invasion cluster that is the next to be invaded. Gravity is included. Trapping is not explicitly treated in the numerical examples but can be added, for example, using a Hoshen–Kopelman algorithm. Computation time to percolation for a 3D system having N total sites and M invaded sites at percolation goes as $O(M \log M)$ for the proposed binary-tree algorithm and as $O(MN)$ for a standard implementation of IP that searches through all of the uninvaded sites at each step. The relation between M and N is $M = N^{D/E}$, where D is the fractal dimension of an infinite cluster and E is Euclidean space dimension. In numerical practice, on finite-sized cubic lattices with invasion structures influenced by the injection boundary and boundary conditions lateral to the flow direction, we observe the scaling $M = N^{0.852}$ in 3D (valid through the second decimal place) instead of $M = N^{0.843}$ based on the infinite cluster fractal dimension $D = 2.53$.

Keywords Two-phase flow · Invasion percolation · Numerical simulation

1 Introduction

In this work, we consider Invasion Percolation (IP) as first introduced by Wilkinson and Willemsen (1983). IP is intended to simulate the pore-by-pore advancement of one fluid immiscibly displacing another in a porous material when the rate of injection is slow enough that viscosity effects can be neglected.

Y. Masson
Institut de Physique du Globe de Paris, Paris, France
e-mail: masson@ipgp.fr

S. R. Pride (✉)
Lawrence Berkeley National Laboratory, Berkeley, CA, USA
e-mail: srpride@lbl.gov

Published online: 04 February 2014

IP applies to the capillary fingering regime of immiscible invasion which is characterized by slow rates of advancement and emergent fractal fluid distributions up to cluster lengths L beyond which IP is no longer strictly valid. The condition for defining L is that viscous pressure drops ΔP_v over these distances must be negligible compared to capillary pressure drops $\Delta P_\mathrm{c} = \sigma/a$ across the menisci, where σ is surface tension and a is a characteristic pore size. Using Darcy's law $Q = (k/\eta)\Delta P_\mathrm{v}/L$, where Q is volumetric flux of invading fluid, k is permeability, and η is viscosity, we have

$$\frac{\Delta P_\mathrm{v}}{\Delta P_\mathrm{c}} = \frac{\eta Q L/k}{\sigma/a} \ll 1. \tag{1}$$

It is common to characterize flow speeds using the capillary number Ca that is usually defined as the viscous shear stress in a pore $\eta Q/(\phi a)$ divided by the capillary pressure σ/a or $\mathrm{Ca} = \eta Q/(\phi\sigma)$. We thus have that IP holds for all emergent invasion clusters satisfying

$$L \ll \frac{k}{\phi\, a\, \mathrm{Ca}}. \tag{2}$$

The fractal distribution of the invading fluid can be characterized as

$$V_\mathrm{I} = \phi\, a^E \left(\frac{L}{a}\right)^D \tag{3}$$

with V_I the volume of fluid injected into a sample of size L^E at percolation, where E is the Euclidean space dimension. The IP fractal dimension D without trapping is numerically observed to be $D = 2.53$ when $E = 3$ and $D = 1.89$ when $E = 2$ (e.g., Wilkinson and Willemsen 1983; Sheppard et al. 1999). Lenormand et al. (1988) performed laboratory experiments using pseudo-2D micro models (pore networks etched in glass plates) and observed a cluster fractal dimension of $D = 1.81$ at slow invasion rates. See Lovoll et al. (2004) and Toussaint et al. (2005, 2012) for interesting experiments and discussions involving a broad range of immiscible invasion scenarios.

IP is normally performed on a lattice of sites connected by bonds. In a porous material like a rock, the sites correspond to the large interstitial pores between the solid grains and the bonds to the constricted flow paths that connect the larger pores. One can distinguish between site IP and bond IP. Site IP is intended to model the scenario in which the invading fluid is wetting and the defending fluid is non-wetting (imbibition). In this case, the invading fluid advances rapidly through the narrow bonds and is possibly blocked at the entrance into the sites. The smaller the effective radius of the site, the more easily that site is invaded. All bonds connected to an invaded site are themselves immediately invaded in site IP. Bond IP is intended to model the invading fluid being non-wetting and the defending fluid being wetting (drainage) in which case it is the bonds that are blocking advancement with larger radii bonds being easier to invade. Once a bond is invaded, the site it is connecting to also becomes immediately invaded. In experimental reality (e.g., Krummel et al. 2013), thin wetting films during slow imbibition may alter fluid distributions and make emergent fluid structures in slow imbibition distinctly different from slow drainage. However, such issues are not yet entirely resolved and are not the focus of this paper.

The site IP numerical procedure is now stated. First, an invasion potential P_i is defined, which controls whether a given site on a lattice is to be invaded (e.g., Wilkinson 1984; Meakin et al. 1992; Glass and Yarrington 1996)

$$P_i = \frac{2\sigma\cos\theta_\mathrm{c}}{a_i} - \Delta\rho g (L - z_i), \tag{4}$$

where a_i is the effective radius of site i, θ_c is the equilibrium contact angle between the two fluids and the solid, $\Delta\rho$ is the fluid density contrast between the invader and the defender, g is the acceleration of gravity, L is the height of the system to be invaded, and z_i is the elevation of site i. A random number generator selects the thresholds a_i from a given probability distribution. Heterogeneity may be allowed for by using different probability distributions for the a_i over different parts of the lattice. Sites with large values of P_i are more likely to be invaded in site IP. Given the above, the site IP algorithm consists of repeating the following two steps: (1) Find the site that is neighbor to the invading cluster and that has the maximum invasion potential P_i; (2) Invade that site. When trapping is added, the efficiency of the proposed algorithm will be limited by the method used to identify trapped clusters of defending sites. Because trapping in IP is not measurably observed in 3D (e.g., Sheppard et al. 1999) and is a 2D effect, it will not be a focus here. The bond IP algorithm involves exchanging the word site for bond in the above, defining the invasion potential as $P_i = 2\sigma/b_i + \Delta\rho g(L - z_i)$, where b_i is the effective radius of bond i, and at each step invading the bond (and attached site) that is neighbor to the cluster and that has the smallest (not largest) value of P_i. No further reference to bond IP will be made. In an alternative deterministic implementation of site IP, the a_i may come from X-ray CT scans of the porespace in which case each point i of the porespace is a site assigned the local pore size a_i surrounding it. In drainage, the site adjacent to the non-wetting fluid that has the largest a_i is the next to be invaded, while in imbibition, the site adjacent to the wetting fluid that has the smallest a_i is the next to be invaded.

The simplest way to implement IP is to scan all uninvaded sites and find the one that has the maximum invasion potential and that neighbors the cluster. The goal of this paper is to give the details of a much faster IP algorithm that uses a binary tree to find the next site to invade. Translating the finite-sized scaling of Eq. 3 into the number of invaded sites at percolation M as a function of the total number of sites in the lattice N, we have $M = N^{D/E}$. Execution times (the CPU time required for the invading fluid to percolate the lattice) in the standard implementation of IP are $O(MN)$ because a search through the N sites occurs at each of the M invasion steps. By limiting the search to a dynamic list of sites that neighbor the cluster, this approach can be improved to $O(M^2)$ assuming standard search algorithms that take time $O(n)$ to search a list of n numbers. Here, we implement a binary tree that improves the execution time to $O(M \log M)$ in the large M limit (say for $M > 10^3$) which can be considered optimal. Sheppard et al. (1999) also employ a binary tree to achieve a similar efficiency and determine a range of fractal dimensions for IP on larger system sizes that can only be calculated using a fast algorithm. However, they provide no details about the algorithm that would allow it to be independently programmed. Given that there are many ways to implement a binary tree, our purpose here is to provide the details of a fast IP algorithm. A search through the recent literature (e.g., Yang et al. 2013; Chen et al. 2012; Pesheva et al. 2010) shows that standard IP with $O(MN)$ execution times is still the norm. To the best of our knowledge, details of an $O(M \log M)$ IP algorithm have never been given.

2 Data Structure

Our search algorithm uses the data structure presented in Fig. 1. In Fig. 1a, a 2D representation of an IP problem is presented. Here, a square lattice is used with the sites represented by circles. Each site is bonded to its four nearest neighbors. Sites that have already been invaded are represented in black. To keep track of the site's state, we first assign a unique index i that identifies each site (denoted in Fig. 1 to the upper left of each site). Each site is then assigned

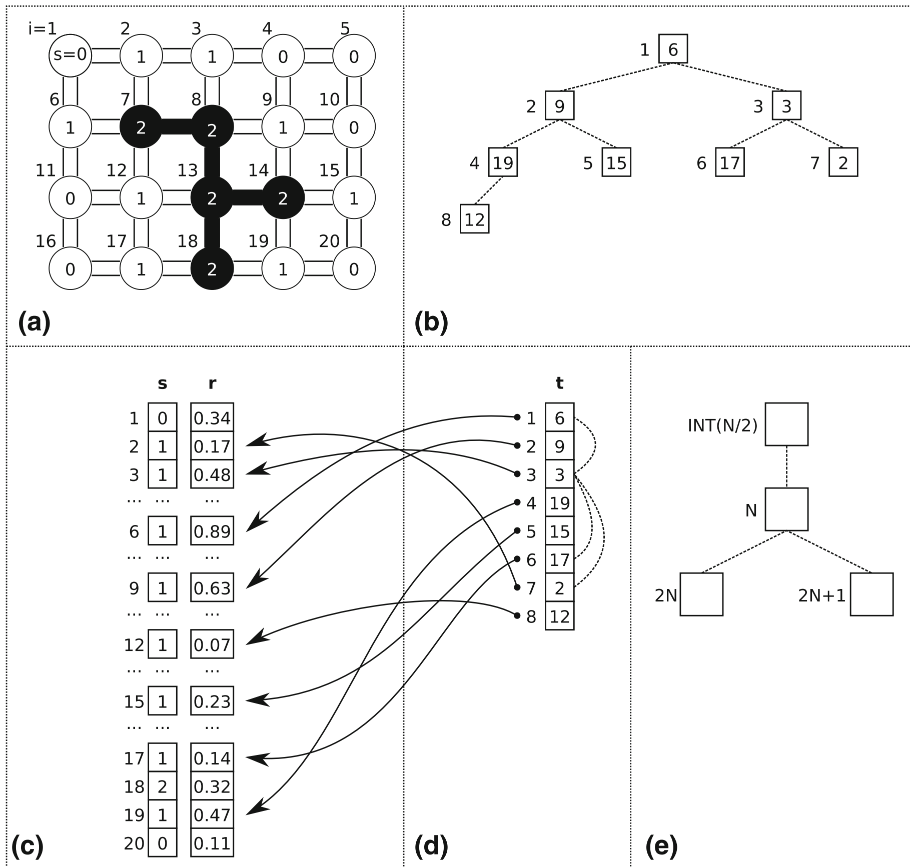


Fig. 1 Data structure used to define an efficient IP algorithm. **a** The network of sites being invaded and the index i and current state $s(i)$ associated with each site. **b** The binary-tree structure built from all the sites neighboring the cluster. **c** The arrays $s(i)$ and $r(i)$ give the state and invasion potential, respectively, for each site. **d** The array $t(j)$ gives the site associated with each node j in the binary tree. **e** The simple relation between a node in the tree $j = N$ and the parent node and children nodes associated with $j = N$

a state flag: 0, if the site has not yet been invaded; 1, if the site is neighbor to the invading cluster; and 2, if the site has already been invaded and thus a part of the cluster. These flags are stored in a state array **s** and the state of the site i is obtained by inquiring $s(i)$. In addition, the invasion potentials of the sites are stored in an array **r** so the invasion potential of site i is obtained by inquiring $r(i)$. The arrays **s** and **r** are depicted in Fig. 1c.

To avoid scanning all sites at each invasion step, a binary tree is employed (e.g., Knuth 1998). The binary-tree structure is presented in Fig. 1b. Every node in the tree corresponds to a site that is neighbor to the invading cluster (i.e., each node in the tree has a state flag of 1). Every node in the tree has one parent and two children (one left and one right). The only exception is the parent of the last node in the tree which will have only one child if the total number of nodes in the tree is even and two children if the total number of nodes is odd. The tree is filled so that the invasion potential of the site associated with a given node in the tree is always smaller than that of the parent node and greater than that of the children nodes. Therefore, the root node (the node at the top of the tree) has the maximum invasion

potential of all sites that are nearest neighbors to the cluster and always corresponds to the next site to invade. The site index associated with each node in the binary tree is stored in a 1D array **t** as shown in Fig. 1d. In this array, the tree nodes are assigned a unique index j by spanning the tree from top to bottom and from left to right as shown in Fig. 1b, d. The site associated with each node j is obtained by inquiring $t(j)$ as shown in Fig. 1d. The invasion potential of site $t(j)$ associated with node j can be obtained by inquiring $r(t(j))$. Last, as shown in Fig. 1e, a given node in the tree with index $j = N$ has a parent node with index $j = \text{int}(N/2)$, where $\text{int}(x)$ is the greatest integer function (produces the greatest integer that is smaller than the argument x), a left child node with index $j = 2N$ and a right child node wth index $j = 2N + 1$. These relations are given by the dotted lines in Fig. 1d.

Given such a data structure, a fast invasion algorithm can be developed.

3 The Proposed IP Algorithm

The algorithm is based on two distinct procedures that are used successively to update the tree structure each time a site is invaded. One procedure (Update Root) is used to remove one node from the tree, and the other (Add Branch) is used to add new nodes to the tree.

3.1 Update Root Procedure

(a) An example of this procedure is sketched in Fig. 2a. First, the site corresponding to the root node in the tree is invaded (step 1 in Fig. 2a). This site's status is changed to "invaded," i.e., $s(t(1)) = 2$. The root node pointer then needs to be updated. We check which of the sites corresponding to the two children nodes 2 and 3 has the greatest invasion potential [in the example, $r(t(2)) > r(t(3))$]. The root node is updated (step 2 in Fig. 2a) so it now points to the child having the higher invading potential [i.e., $t(1) = t(2)$ in the example]. We then repeat this operation to update node 2 from its children (step 3 in Fig. 2a). This continues until we reach a dead end at say node $j = L$; i.e., this node has no children (step 5 in Fig. 2a). In order to keep the tree complete, node $j = L$ is updated to point towards the site that was previously associated with the last node in the tree (step 6 in Fig. 2a). Since the last node in the tree is no longer used, the size of the tree is decreased by one node.

(b) We now need to move the site associated with the last modified node $j = L$ into its correct position within the tree. This procedure is sketched in Fig. 2b. First, the invading potential $r(t(L))$ of the site $t(L)$ associated with the $j = L$ node is compared to the potential $r(t(\text{int}(L/2)))$ associated with the parent node $j = \text{int}(L/2)$. If $r(t(L)) < r(t(\text{int}(L/2)))$, we stop here. If $r(t(L)) > r(t(\text{int}(L/2)))$, we swap the sites, i.e., $t(L) \rightleftharpoons t(\text{int}(L/2))$ as depicted in step 7 in Fig. 2b. We then go up one level in the tree and repeat the operation until the invading potential associated with the current node is smaller than the one associated with its parent (step 8 in Fig. 2a).

3.2 Add Branch Procedure

This procedure is used to add new sites to the tree and is presented in Fig. 3. If there are N nodes in the tree, we first create a new node $j = N + 1$ at the end of the tree (step 1 in Fig. 3) so that there are now $N_a = N + 1$ nodes in the tree. The node value is set to point towards the site i adjacent to the invasion cluster that we want to add, i.e., $t(N_a) = i$. We then associate the new site to its correct position in the tree. This is done by comparing the invading potential $r(t(N_a))$ of the current node $j = N_a$ to that of the parent node $r(t(\text{int}(N_a/2)))$ and swapping

Fig. 2 Schematic of the Update Root procedure. See the text for a description of this procedure

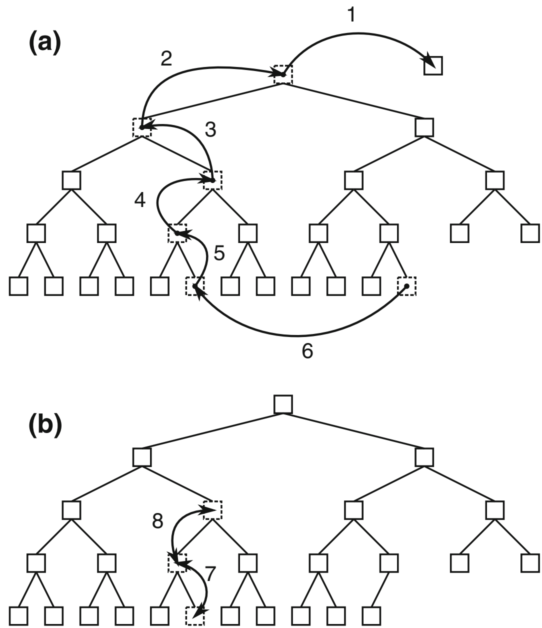
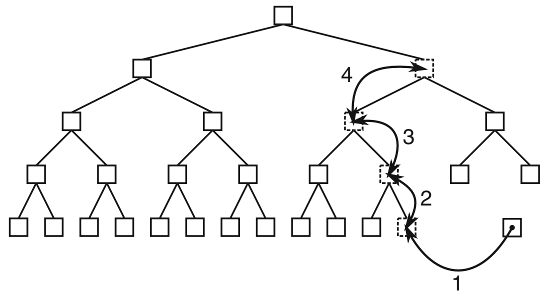


Fig. 3 Schematic of the Add Branch procedure. See the text for a description of this procedure



node positions if $r(t(N_a)) > r(t(\text{int}(N_a/2)))$. This step is repeated until the added site is at the right place in the tree (i.e., it has an invading potential smaller than the next parent node as depicted in steps 2, 3, and 4 in Fig. 3).

3.3 Proposed Algorithm

The above procedures are used in the following algorithm:

1. Set up the injection sites, i.e., set $s(i) = 2$ for all i where we want to inject fluid.
2. Find all the nearest neighbors to the injection sites.
3. Build the initial tree by successively adding all the neighboring sites to the tree using an Add Branch procedure.
4. Start a loop and continue until percolation steps 5 to 8 below:
5. Invade the next site by changing the status of the site associated with the root node from $s(t(1)) = 1$ to $s(t(1)) = 2$.
6. Find all the nearest uninvaded neighbors to $t(1)$ and change their status from $s(i) = 0$ to $s(i) = 1$.

7. Use the Update Root procedure to update the root node of the tree.
8. Add to the tree the nearest neighbors defined in step 6 using the Add Branch procedure.
9. End the loop the first time $s(i) = 2$ for an i on the exit surface.

3.4 Adding Trapping

To include the main 2D effect of clusters of defending sites becoming completely surrounded by invaded sites, we first search for such trapped clusters at each step using the Hoshen and Kopelman (1976) algorithm which efficiently identifies and separately labels all isolated clusters on a lattice. Details of the algorithm will not be elaborated upon here. A nice analysis of algorithms that identify isolated clusters is given by Babalievski (1998). The trapped sites of defending fluid so identified cannot be invaded on physical grounds and thus their associated nodes in the binary tree must be permanently removed using a procedure similar to the Update Root procedure but starting not at the root node but at the node to be eliminated. In 3D, trapped clusters of defending sites do not significantly develop.

4 Efficiency

The time complexity of the binary-tree algorithm presented in this paper can be understood by considering the height h of the tree as measured by the number of levels (or floors) within the tree. Assuming that the total number of nodes in the tree is proportional to the number of invaded sites M, we have $M = 2^h$ (plus what ever nodes that do not entirely fill a floor). In the invasion history leading up to the observed M at percolation, there is 1 comparison (operation) made when there is only 1 floor present, 2 comparisons made 2 times when there are 2 floors, 3 comparisons made 4 times when there are 3 floors present and, in general, n comparisons made $2^{(n-1)}$ times when there are n floors present. Thus, with each comparison taking time Δt, the CPU time T as a function of M goes as

$$T(M) = \Delta t \sum_{n=1}^{h(M)} n 2^{(n-1)}, \tag{5}$$

where $h(M) = \log_2 M$. In the large h limit, the sum can be evaluated by multiplying by $\Delta n = 1$ and converting the sum to an integral to obtain

$$T_\infty(M) = \frac{\Delta t}{2} \int_1^h n 2^n \, \mathrm{d}n \tag{6}$$

$$= \frac{\Delta t}{2(\ln 2)^2} \left[(h \ln 2 - 1) 2^h - 2(\ln 2 - 1) \right] \tag{7}$$

$$= \frac{\Delta t}{2(\ln 2)^2} \left[M \ln M - M - 2(\ln 2 - 1) \right] \sim O(M \ln M). \tag{8}$$

In comparing $T(M)$ to the approximation $T_\infty(M) \sim O(M \log M)$, they become asymptotically indistinguishable from each other for $M > 10^3$.

To verify these predicted execution times, Fig. 4 gives the actual times to percolation determined using the Fortran 95 function CPU_TIME for various lattice (cluster) sizes after averaging the percolation times over 40 different realizations of the lattice (i.e., over different random thresholds). The average percolation time $T(M)$ for the binary-tree fast algorithm

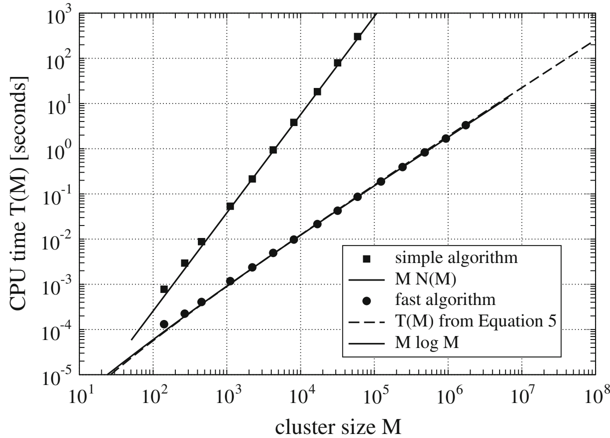


Fig. 4 CPU time from start to percolation as a function of the observed cluster size M in 3D cubic lattices. Each given time observation using both the simple (*solid squares*) and fast (*solid circles*) algorithms is averaged over 40 different realizations of the random thresholds. For cluster sizes less than roughly 100, the asymptotic execution time for the binary tree $O(M \log M)$ begins to be noticeably differ from Eq. 5. In the prediction of $O(MN)$ for the execution time of the simple algorithm, the function $N(M)$ was obtained from the fit given in Fig. 5

observed numerically for each percolation cluster of size M (the square symbols) is plotted in the figure and compared to the estimate given by Eq. 5 and to $M \log M$. A nice agreement between the time complexity prediction and the actual numerical results is seen. We also see that the prediction made using Eq. 5 is equivalent to $M \log M$ for $M > 10^3$. Also given in Fig. 4 is the time to percolation using the simple algorithm that searches all sites at each invasion step. As expected, these times scale as $O(MN)$ as demonstrated in the figure. Similar results hold in 2D.

In order to obtain the function $N(M)$ as well as to know $M(N)$ over a continuous range of lattice sizes N, we fit the discretely observed $M(N)$ obtained using IP with a power law as shown in Fig. 5. The best least-squares fit is $M(N) = 0.407N^{0.852}$ which is a slightly more massive cluster than the theoretically expected value of $M(N) \propto N^{D/E} = N^{0.843}$. The reason for the difference is the boundary effects associated with invading our finite-sized domains. The invasions were obtained by injecting over the entire $z = 0$ face of the cube and stopping percolation when invading fluid first gets to $z = N^{1/3}$. In the x and y directions perpendicular to the direction of invasion, we use periodic boundary conditions. From our fit of $M(N) \propto N^{0.852}$, we obtain $N(M) \propto M^{1.174}$ which was used in predicting the $O(MN)$ execution time for the simple algorithm in Fig. 4. Also shown in Fig. 5 are the cluster sizes obtained using the simple and fast algorithms which overlap exactly. In the end, what prevents going to larger lattice sizes with the fast algorithm is not CPU time, but allocatable memory availability.

Last, Sheppard et al. (1999) report observing execution times of $O(M^{1.24})$ for their binary-tree algorithm over a similar range of system sizes (but they do not give a plot). This can be understood, at least partially, if we rewrite $M \log_2 M$ as a power law M^α and identify the exponent α as $\alpha = 1 + \log(\log M - \log 2)/\log M$. When we try to fit a power law over the range $10^5 < M < 10^6$ to our fast-algorithm execution times, we obtain $T(M) \sim O(M^{1.085})$, which is faster than the Sheppard et al. (1999) result, but consistent with the prediction of $\alpha = 1 + \log(\log 6 - \log 2)/\log 6 = 1.095$.

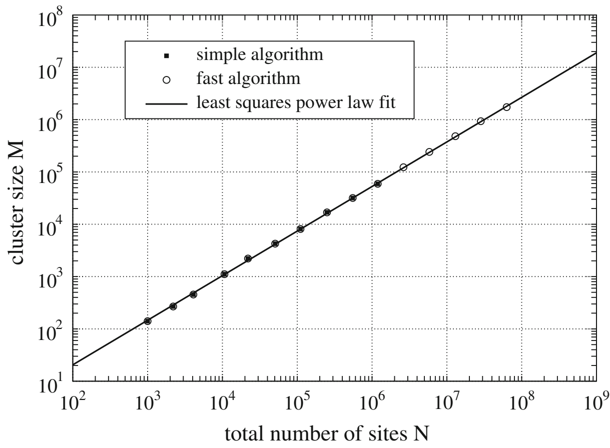
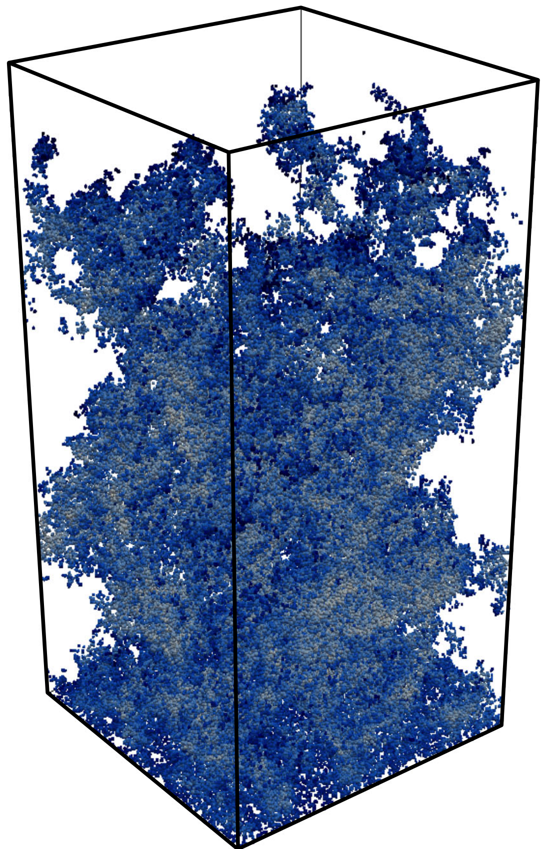


Fig. 5 Cluster size M observed at percolation in 3D cubic systems having total sites N. The *open circles* are the sizes determined using the fast binary-tree algorithm while the *solid squares* are the sizes determined using the simple algorithm. Both algorithms give the same results; however, the fast algorithm can explore considerably larger system sizes in a reasonable amount of time. The given power-law fit is $M(N) = 0.407N^{0.852}$

Fig. 6 A typical IP cluster with no gravity $g = 0$. Dimension $256 \times 256 \times 512$. Total CPU time 0.57 s



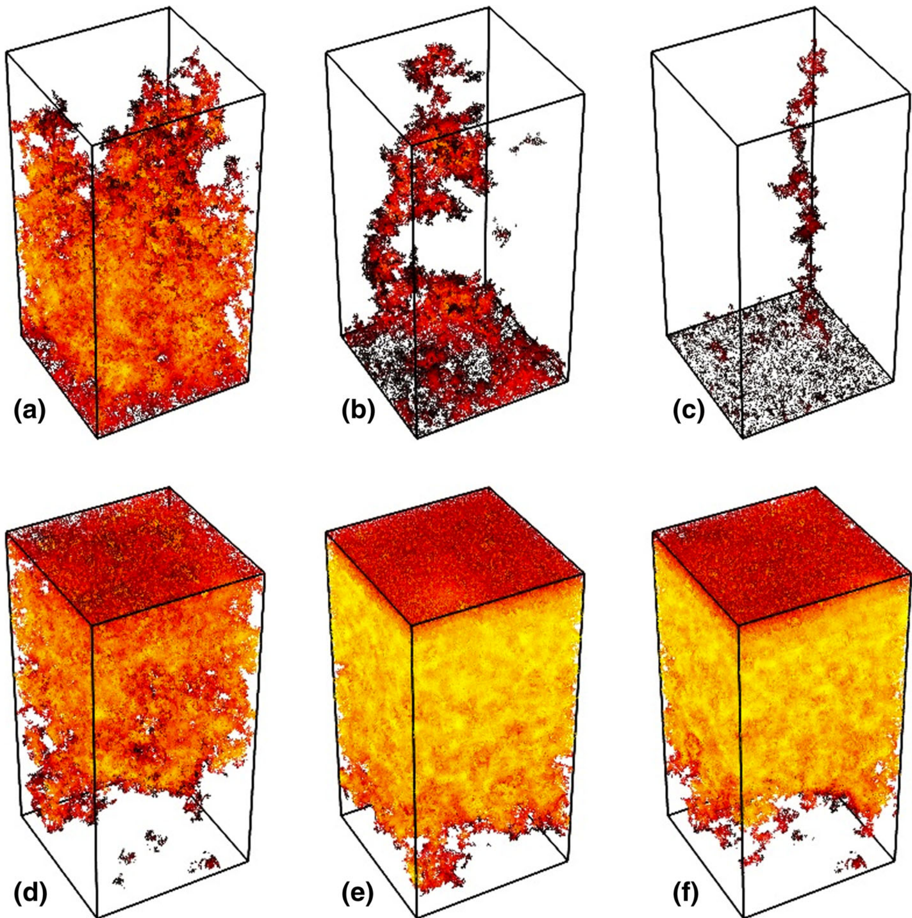


Fig. 7 Effect of gravity. **a** $B_0 = 0.00$ injection from bottom. **b** $B_0 = 0.0001$ injection from bottom. **c** $B_0 = 0.001$ injection from bottom. **d** $B_0 = 0.00$ injection from top. **e** $B_0 = 0.0001$ injection from top. **f** $B_0 = 0.001$ injection from top

5 Numerical Examples

Although the purpose of this article is to give the details of a fast IP algorithm and demonstrate its efficiency, we conclude by giving two simple examples of site IP. A 3D cluster at percolation in a lattice of size $256 \times 256 \times 512$ is shown in Fig. 6. The CPU time to produce the given cluster is 0.57 s using the fast algorithm and would have taken roughly 3×10^6 s (or over 1 month) using the simple algorithm. All sites on the bottom face of the prism are used for injection and the simulation stops when the cluster touches the top face. Although the mass (volume) of the invaded fluid follows, at least approximately, the $M = N^{D/E} = N^{0.84}$ scaling of Eq. 3, the size of the individual patches of defending sites, that are of interest in many applications, do not have probability distributions that are simple power laws (c.f., Masson and Pride 2011).

In Fig. 7, the well-known gravitational fingering effect is demonstrated. When the invading fluid is lighter than the defending fluid (i.e., $\Delta\rho > 0$) and when invasion is occurring from

lower to higher in the gravity field, a finger of invading fluid that gets a distance d ahead of the front will have a bouyancy-induced pressure drop across the lead meniscus given by $\Delta P_g = \Delta\rho g d$. Comparing this to $\Delta P_c = \sigma/a$, the instability occurs when

$$\frac{\Delta P_g}{\Delta P_c} = \frac{a\Delta\rho g d}{\sigma} \gg 1. \tag{9}$$

Defining the dimensionless Bond number as $\text{Bo} = \Delta\rho g a^2/\sigma$, fingers that develop through normal capillary fingering that satisfy $d \gg a/\text{Bo}$ will accelerate and grow as shown in Fig. 7a–c. When the invasion occurs from higher to lower in the gravity field, the fingering is suppressed and more of the defending fluid is displaced compared to if gravity is not acting at all ($\text{Bo} = 0$). This is shown in the sequence of Fig. 7d–f.

6 Conclusions

A fast algorithm for IP is given that uses a binary-tree structure to quickly find the next site to invade without having to scan through all uninvaded sites that neighbor the cluster. Computation time scales as $O(M \log M)$ as expected for a binary tree both in 3D and 2D, where M is the size of the invasion cluster at percolation. This can be compared to execution times of $O(MN)$ for a simple algorithm, where N is the total number of sites in the lattice. Based on execution times quoted by Sheppard et al. (1999) who also employ a binary tree for finding the next site to invade, their implementation is at least slightly different from that given in this paper. Sheppard et al. (1999) do not give any details about their algorithm and one of the main purposes of the present paper is to provide enough detail about the algorithm that it can be independently programmed. Last, the algorithm is straightforward to parallelize even if the present paper was focused on using a single processor. For example, one could divide the lattice sites between the processors and have one binary tree per processor. The root nodes at each processor would be compared to determine the next site to invade. The neighbors to the invaded site would then be communicated to the appropriate processors and the process continued.

Acknowledgments This material is based upon work supported as part of the Center for Nanoscale Control of Geologic CO2, an Energy Frontier Research Center and as part of the LBNL Geophysics Cluster, both funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Award Number DE-ACO2-05CH11231. Y. Masson has recently been supported through the European Community's 7th Framework Program (FP-7-IDEAS-ERC), ERC Advanced Grant (WAVETOMO).

References

Babalievski, F.: Cluster counting: the Hoshen–Kopelman algorithm versus spanning tree approaches. Int. J. Mod. Phys. C **9**, 43–60 (1998)

Chen, F., Shinosky, M., Aitken, J., Yang, C.C., Edelstein, D.: Invasion percolation model for abnormal time-dependent dielectric breakdown characteristic of low-k dielectrics due to massive metallic diffusion. Appl. Phys. Lett. **101**, 242, 904 (2012)

Glass, R.J., Yarrington, L.: Simulation of gravity fingering in porous media using a modified invasion-percolation model. Geoderma **70**, 231–252 (1996)

Hoshen, J., Kopelman, R.: Percolation and cluster distributions. I. Cluster multiple labeling technique and critical concentration algorithm. Phys. Rev. B **14**, 3438–3445 (1976)

Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, 2nd edn. Addison-Wesley, Reading, MA (1998)

Krummel, A.T., Datta, S.S., Munster, S., Weitz, D.A.: Visualizing multiphase flow and trapped fluid configurations in a model three-dimensional porous medium. AIChE J. **59**, 1022–1029 (2013)
Lenormand, R., Touboul, E., Zarcone, C.: Numerical models and experiments on immiscible displacements in porous media. J. Fluid Mech. **189**, 165–187 (1988)
Lovoll, G., Meheust, Y., Toussaint, R., Schmittbuhl, J., Maloy, K.J.: Growth activity during fingering in a porous Hele–Shaw cell. Phys. Rev. E **70**, 026301 (2004)
Masson, Y.J., Pride, S.R.: Seismic attenuation due to patchy saturation. J. Geophys. Res. **116**, B03, 206 (2011)
Meakin, P., Feder, J., Frette, V., Jossang, T.: Invasion percolation in a destabilizing gradient. Phys. Rev. A **46**, 3357–3368 (1992)
Pesheva, N., Stefanov, I., Slavtchev, S.: Application of the invasion percolation model to water–gas flows in artificial soils with plants. Transp. Porous Med. **83**, 319–331 (2010)
Sheppard, A.P., Knackstedt, M.A., Pinczewski, W.V., Sahimi, M.: Invasion percolation: new algorithms and universality classes. J. Phys. A Math. Gen. **32**, L521–L529 (1999)
Toussaint, R., Lovoll, G., Meheust, Y., Maloy, K.J., Schmittbuhl, J.: Influence of pore-scale disorder on viscous fingering during drainage. Europhys. Lett. **71**, 583 (2005)
Toussaint, R., Maloy, K.J., Lovoll, G., Meheust, Y., Jankov, M., Schafer, G., Schmittbuhl, J.: Two-phase flow: structure, upscaling and consequences for macroscopic transport processes. Vadose Zone J. (2012). doi:10.2136/vzj2011.0123
Wilkinson, D.: Percolation model of immiscible displacement in the presence of buoyancy forces. Phys. Rev. A **30**, 520–531 (1984)
Wilkinson, D., Willemsen, J.F.: Invasion percolation: a new form of percolation theory. J. Phys. A Math. Gen. **16**, 3365–3376 (1983)
Yang, Z., Niemi, A., Fagerlund, F., Illangasekare, T.: Two-phase flow in rough-walled fractures: comparison of continuum and invasion-percolation models. Water Resour. Res. **49**, 993–1002 (2013)