A fast two-step algorithm for invasion percolation with trapping

Yder J. Masson

Computers & Geosciences 90 (2016) 41–48

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

A fast two-step algorithm for invasion percolation with trapping

Yder Masson

Institut de Physique du Globe de Paris, 1 rue Jussieu, 75005, Paris, France

ARTICLE INFO

Article history:
Received 22 September 2015
Received in revised form
2 February 2016
Accepted 3 February 2016

Keywords:
Invasion percolation
Trapping
Cluster labeling
Algorithm
Hoshen Kopelman
Two phase flow

ABSTRACT

I present a fast algorithm for modeling invasion percolation (IP) with trapping (TIP). IP is a numerical algorithm that models quasi-static (i.e. slow) fluid invasion in porous media. Trapping occurs when the invading fluid (that is injected) forms continuous surfaces surrounding patches of the displaced fluid (that is assumed incompressible and originally saturates the invaded medium). In TIP, the invading fluid is not allowed to enter the trapped patches. I demonstrate that TIP can be modeled in two steps: (1) Run an IP simulation without trapping (NTIP). (2) Identify the sites that invaded trapped regions and remove them from the chronological list of sites invaded in NTIP. Fast algorithms exist for solving NTIP. The focus of this paper is to propose an efficient solution for step (2). I show that it can be solved using a disjoint set data structure and going backward in time, i.e. by un-invading all sites invaded in NTIP in reverse order. Time reversal of the invasion greatly reduces the computational complexity for the identification of trapped sites as one only needs to investigate sites neighbor to the latest invaded/un-invaded site. This differs from traditional approaches where trapping is performed in real time, i.e. as the IP simulation is running, and where it is sometimes necessary to investigate the whole lattice to identify newly trapped regions. With the proposed algorithm, the total computational time for the identification and the removal of trapped sites goes as $O(N)$, where N is the total number of sites in the lattice.

1. Introduction

Invasion percolation (IP) is an algorithmic model introduced by Wilkinson and Willemsen (1983) to model biphasic fluid migration in porous materials. IP considers the scenario of two immiscible fluids. The first fluid is slowly injected inside a random porous medium and displaces the second fluid that originally saturates the pore space. This process occurs naturally in various contexts, for example, when oil, water or gas migrate through reservoir rocks. In IP, the pore space is modeled as a lattice of sites that corresponds to larger interstitial spaces or pores between grains, connected by bonds, that represent smaller throats connecting neighboring pores. The invasion proceeds by invading the sites one by one. As the invasion goes, it may happen that some clusters formed by the defending fluid become completely surrounded by the invading fluid, this is called trapping. There are two IP model variants: invasion percolation without trapping (NTIP), where trapped sites are allowed to be invaded, and, invasion percolation with trapping (TIP), where the defending fluid is assumed incompressible and sites belonging to trapped clusters cannot be invaded. From an algorithmic point of view, trapping brings additional complexity to the IP procedure as it requires extra operations to search for trapped clusters. The focus of this paper is to propose an efficient algorithm for identifying the trapped sites in TIP.

Regardless of trapping, one often distinguishes between site IP and bond IP that intend to model two different types of displacement named drainage and imbibition (e.g. Lenormand and Bories, 1980; Chandler et al., 1982). Site IP intends to model imbibition where a wetting fluid (e.g. water) is invading a medium originally saturated by a non-wetting fluid (e.g. oil). Bond IP intends to model the opposite situation called drainage where a non-wetting fluid is invading a medium originally saturated by a wetting fluid. In practice, IP models have mainly been used for slow drainage in porous and fractured media, this is because the use of the IP for imbibition is not well justifiable, due to additional physical mechanisms (e.g., film flow and snap-off) which often accompany imbibition and are not taken into account in IP. I refer the reader to Løvoll et al. (2005) and Toussaint et al. (2005, 2012) for extensive discussions of various invasion scenarios. From an algorithmic point of view, there is no difference between bond IP and site IP and they can be modeled using a unique IP algorithm on appropriate lattices (e.g. Patzek et al., 2001). For the sake of clarity, I only consider site IP in the present study.

The site TIP procedure (Wilkinson and Willemsen, 1983) consists of the following steps:

1. Define a lattice of sites connected by bonds, injection sites (i.e. sites at which the invading fluid is injected), sink sites (i.e. sites

E-mail address: yder.masson@cal.berkeley.edu

http://dx.doi.org/10.1016/j.cageo.2016.02.003
0098-3004/

at which the defending fluid is free to escape), and set all sites to invadable.
2. To all sites in the lattice, assign an invasion potential (e.g. Wilkinson, 1984; Meakin et al., 1992; Glass and Yarrington, 1996)

$$P_i = \frac{2\sigma \cos(\theta_c)}{a_i} - \Delta\rho g (L - z_i) \quad (1)$$

where a_i is the effective radius of site i, θ_c is the equilibrium contact angle between the wetting fluid and the solid, $\Delta\rho$ is the density contrast between the two fluids, 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. Practically, the radii a_i are chosen randomly from a given probability distribution.
3. Invade the sites one by one until the invading fluid percolates, i.e. repeat the following three steps until the invading fluid reaches a sink site:
(a) Identify the trapped clusters formed by the defending fluid that are not connected to a sink and set all sites belonging to these clusters to un-invadable.
(b) Among all invadable sites, find the site that is neighbor to the invading fluid and that has the maximum invasion potential.
(c) Invade that site.

The site NTIP procedure is similar to the TIP procedure except that step 3(a) in the above sequence is not performed.

The NTIP problem can be solved efficiently using a binary tree data structure (e.g. Knuth, 1998) that maintains an up-to-date list of sites that neighbor the invader. This data structure permits us to find the site with the largest invasion potential in $O(1)$ operation and to insert new neighbor sites in $O(\log(n))$ operations, where $n \leq N$ is the number of active sites in the list (e.g. Schwarzer et al., 1999; Sheppard et al., 1999; Masson and Pride, 2014). I refer to Masson and Pride (2014) for a detailed implementation of NTIP using a perfectly balanced binary tree that strictly guarantees $O(M \log M)$ execution time, where M is the number of sites invaded at percolation time.

Algorithms for TIP can be constructed by modifying existing fast algorithms that solve the NTIP problem. A classic approach is to identify the trapped clusters formed by the defender at each time step. The trapped sites are then flagged to prevent further invasion (i.e. step 3(a) in the above procedure). This can be done using cluster labeling algorithms such as the classic (Hoshen and Kopelman, 1976) algorithm, or a similar but more generic disjoint-set data structure (e.g. Knuth, 1998; Cormen et al., 2001). A nice analysis of algorithms that label isolated clusters is given by Babalievski (1998). Labeling clusters at each time step is however highly inefficient because it requires to scan the entire lattice M times which gives a computation time of $O(MN)$ for the trapping part of TIP, where N is the total number of sites in the lattice. A better approach is to first investigate the neighbors of each newly invaded site to check for trapping which is ruled out in most cases. If trapping is possible, a different algorithm is used to update the cluster labeling as necessary (e.g. Schwarzer et al., 1999; Sheppard et al., 1999; Meakin, 1991). In this work, I propose an alternative approach where clusters are labeled only once at the end of the IP invasion.

The aim of this paper is to detail a comprehensive and computationally efficient algorithm for solving the TIP problem. A search through the recent literature (Chen et al., 2012; Yang et al., 2013) shows that less efficient algorithms with execution time $O(MN)$ are still widely used, further, I found no study that provides the reader with a detailed fast TIP algorithm that can easily be turned into code. The present paper together with Masson and Pride (2014) should allow the graduate student or people less familiar with IP to get started quickly with adequate algorithms. The solution I propose for TIP is based on the simple observation that trapping can be treated a posteriori through time-reversal of the invasion sequence obtained in NTIP. Surprisingly, to my knowledge, this simple and efficient approach has not been reported in the literature. Numerical results show better performance than previously proposed algorithms.

2. The proposed TIP algorithm

In this section, I first show that TIP can be solved by post-processing the NTIP solution. Then, I detail an efficient two-steps algorithm for solving TIP. Finally I analyze the performance of the proposed algorithm.

2.1. Algorithm principle

Consider a NTIP sequence as illustrated in Fig. 1 and imagine a site belonging to a trapped region is invaded, e.g. as in Fig. 1b. We observe that the interface between the two fluids outside the trapped region (i.e. which encloses the newly invaded site) is not modified by this invasion. Therefore, this will not affect the way sites will be invaded outside this trapped region, i.e. which one of these sites will be invaded and in what order. In other words, the sites invaded in NTIP are the same as those invaded in TIP plus some extra sites belonging to trapped regions at invasion time. Based on this observation, I formulate the following proposition:

Proposition 1. *Given two IP simulations, a NTIP simulation S_{NT} and a TIP simulation S_T, both performed using the exact same setup (i.e. lattice, realization of invasion potentials, injection sites, sink sites, and boundary conditions), let L_T be the list of sites invaded in S_T sorted by increasing invasion time and L_{NT} be the list of sites invaded in S_{NT} sorted by increasing invasion time.* When removing the sites that invaded trapped regions from L_{NT}, we obtain L_T.

It follows from Proposition 1 that the TIP problem can be decomposed into two sub-problems that can be solved sequentially: (1) Solve the NTIP problem. (2) Identify and remove the sites that invaded trapped regions in NTIP.

One inefficient way to obtain the TIP invasion sequence knowing the NTIP solution is to proceed as for traditional TIP. That is, redo the invasion and, at each invasion step, check whether or not the newly invaded site belongs to a trapped region. A simple improvement to this procedure is to identify all clusters connected to a sink at the end of NTIP. In this case, all sites connected to a sink do not need to be re-investigated when searching for trapped clusters. I now show that further improvement can be achieved through time-reversal of the NTIP simulation.

Consider the situation where a new site is invaded in NTIP. There are three and only three possible scenarios regarding the evolution of the clusters formed by the defending fluid:

1. The number of clusters stays the same (see e.g. Fig. 1a).
2. One cluster of size one is suppressed (see e.g. Fig. 1d).
3. One cluster is split into multiple clusters (see e.g. Fig. 1e).

Now imagine the opposite situation where we go backward in time and un-invade a site, there are again three possible scenarios:

1. The number of cluster stays the same (see e.g. Fig. 1a).
2. A new cluster of size one is created (see e.g. Fig. 1d).
3. Multiple clusters are merged together (see e.g. Fig. 1e).

Notice the important difference between the two situations (i.e. going forward in time and going backward in time): on the one

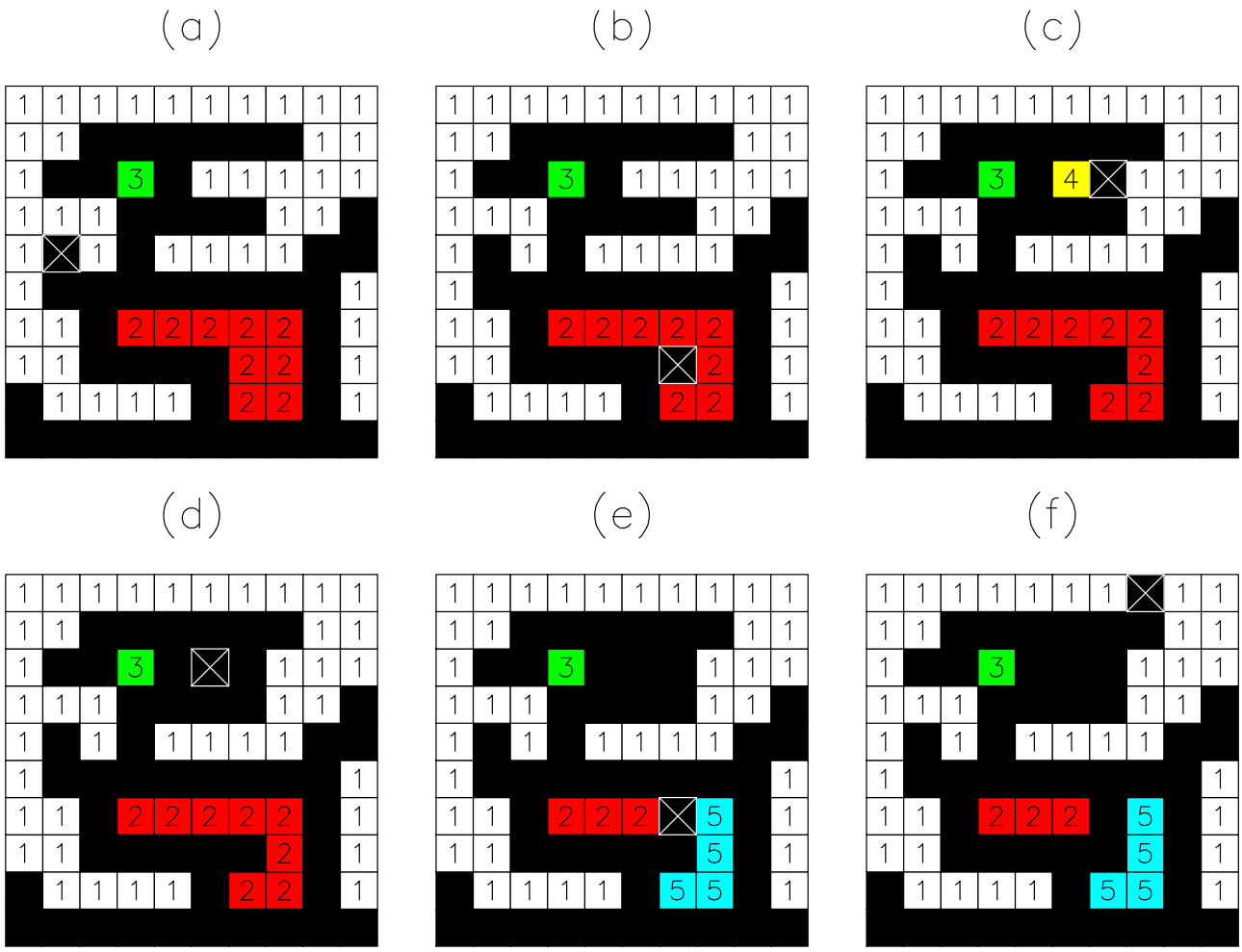

Fig. 1. Snapshots showing the last six invasion steps of a 2D site invasion percolation simulation without trapping (NTIP) on a square lattice. The sites filled with the invading fluid are pictured in black. The remaining sites are occupied by the defending fluid. The invading fluid is injected from the bottom side and periodic boundary conditions are used in the horizontal direction. The defending fluid forms multiple clusters that are labelled with distinct numbers and represented in colors. The fluid contained in the white cluster labelled with the value one is free to escape from the top of the domain (i.e the sink). The remaining clusters are trapped by the invading fluid. At each one of the invasion steps (a, b, c, d, e and f) a single site is invaded and it is marked by a white cross. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

hand, when un-invading a site, it is possible to determine which one of the three scenarios occurred by simply looking at the neighbors to the newly un-invaded site. This is because when connecting multiple patches, we are sure that they are effectively merged to form a unique cluster. On the other hand, when invading a site, we may need to investigate the whole lattice to label the cluster formed by the defending fluid. This is because, when the defending fluid is locally split into multiple patches, there is no guarantee that these patches are not still connected far away from the newly invaded site. Therefore, an efficient way to identify sites that invaded trapped region in NTIP is to undo the invasion sequence, i.e. un-invade all sites in reverse order while bookkeeping the cluster evolution. As an example, consider what happens when undoing the invasion sequence in Fig. 1, i.e. starting with Fig. 1f and ending with Fig. 1a:

In Fig. 1f, the site marked by a cross has two neighbors belonging to cluster 1. (Notice that all sites in the lattice have four neighbors: up, down, left and right.) When un-invaded, this site is added to cluster 1 that is connected to a sink. Therefore, this site would have been invaded in TIP.

In Fig. 1e, the site marked by a cross has two neighbors belonging to cluster 2 and cluster 5. Un-invading this site effectively merges the two clusters. Their union gives cluster 2 in Fig. 1d that is a trapped cluster. This means that the site would have not been invaded in TIP.

In Fig. 1d, the site marked by a cross has no neighbors filled with the defending fluid. Un-invading this site creates a new cluster, i.e. cluster 4 in Fig. 1c. Cluster 4 is a trapped cluster so the site would have not been invaded in TIP.

in Fig. 1c, the site marked by a cross has two neighbors belonging to cluster 1 and cluster 4. When this site is un-invaded, the two clusters are merged into cluster 1 in Fig. 1b. Cluster 1 is connected to a sink so the site would have been invaded in TIP.

In Fig. 1b the site marked by a cross has three neighbors belonging to cluster 2 which is trapped. Hence, the site would have not been invaded in TIP.

In Fig. 1a the site marked by a cross has three neighbors belonging to cluster 1 that is connected to a sink. Therefore, the site would have been invaded in TIP.

This process goes on and on up until all sites have been un-invaded. We can verify that when ignoring the invasions in Fig. 1b, d and e, we obtain the invasion sequence corresponding to TIP pictured in Fig. 2. In the next section, I detail an algorithm that follows this procedure and applies to arbitrary lattices.

2.2. Practical implementation

The procedure described in the previous section can be implemented using a disjoint set data structure (see e.g. Cormen

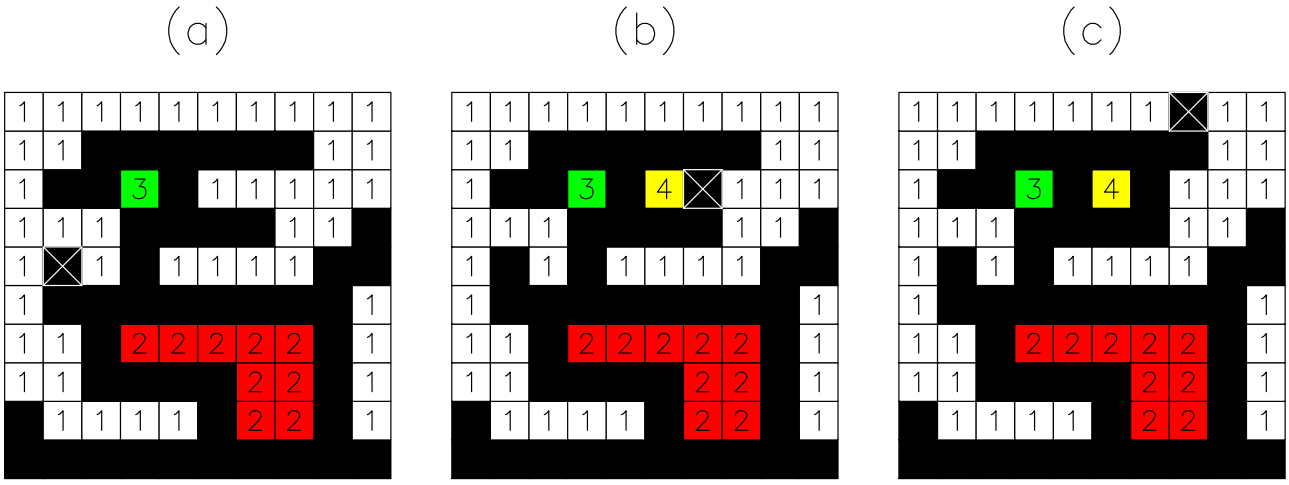

Fig. 2. Snapshots showing the last three invasion steps of a 2D site invasion percolation simulation with trapping (TIP) on a square lattice. The random realization of invasion potentials, the injection area and the boundary conditions are assumed to be similar to those in Fig. 1. Note that the sites invaded in (a), (b) and (c) are invaded in the same order as the sites invaded in Fig. 1(a), (c) and (f), respectively. The sites invaded in Fig. 1(b), (d) and (d) are not invaded here as they belong to trapped clusters.

et al., 2001) that is well known from computer scientist and detailed in Appendix A for completeness. I propose the following algorithm for solving the TIP problem:

1. *Setup the TIP simulation*: Define a lattice consisting of N sites connected by bonds. Assign a unique index i to all sites in the lattice. Define injection sites at which the invading fluid is injected and sink sites at which the defending fluid is free to escape the system. Initially, all sites are filled with the defending fluid. Initialize the number of sites $M_T=0$ invaded in the TIP simulation and define a one-dimensional array $L_T(t)$ that will contain the indices of the sites invaded during the TIP simulation.
2. *Solve the NTIP problem*: Using the same setup as for the TIP simulation, run a NTIP simulation and recast the result in a one-dimensional array $L_{NT}(t)$ that return the index of the site invaded at time t, where $1 \le t \le M_{NT}$ and M_{NT} is the total number of sites invaded in the NTIP simulation. I refer to Masson and Pride (2014) for a detailed algorithm solving the NTIP problem.
3. *Label the clusters formed by the defending fluid*: Define a one-dimensional label array $l(i)$ of dimension N. At first, no sites have a label, i.e. $l(i) = \text{no\_label}\ \forall\, i$. Initialize the number of clusters $N_c=0$. For all pairs of neighbor sites (i,j) where both i and j are filled with defending fluid (at the end of the NTIP simulation) do:

If site i has a label and site j has a label **Then**
  $l(j) = \text{UNION}(l(i), l(j))$
If site i has a label and site j has no label **Then**
  $l(j) = \text{FIND}(l(i))$
If site i has no label and site j has a label **Then**
  $l(i) = \text{FIND}(l(j))$
Else
  $l(i) = l(j) = \text{CREATE\_CLUSTER}(N_c)$
End If

and, for all isolated sites j filled with defending fluid and that have no neighbors filled with defending fluid do:

$l(j) = \text{CREATE\_CLUSTER}(N_C)$.

Once this is done, all sites i filled with defending fluid have a label $l(i)$, and, $\text{FIND}(l(i))$ returns the label of the cluster that contains site i.

4. *Assign a unique label to all clusters connected to a sink*: That is, create a label

$l_s = \text{CREATE\_CLUSTER}(N_c)$

for the sink cluster, and then, merge all clusters connected to a sink. That is, for all sink sites j do:

$l_s = \text{UNION}(l_s, l(j))$.

From now on, if we have $\text{FIND}(l(i)) = l_s$, it means that site i is connected to a sink.

5. *Un-invade all sites one by one in reverse order*: For all sites stored in L_{NT}, starting with site $i = L_{NT}(M_{NT})$ and ending with site $i = L_{NT}(1)$, at each step t when un-invading site $i = L_{NT}(t)$, do:

If site i has no neighbor labeled **Then**
  $l(i) = \text{CREATE\_CLUSTER}(N_C)$
Else If site i has unique neighbor j labeled **Then**
  $l(i) = \text{FIND}(l(j))$
Else If site i has n neighbors $j_1, j_2, \ldots, j_n$ labeled **Then**
  For $k=1$ to $n-1$ **Do**
    $l(i) = \text{UNION}(l(j_k), l(j_{k+1}))$
  End For
End If

then, check whether or not the newly un-invaded site i is connected to a sink. If it is connected to a sink, add it to the TIP list L_T:

If $\text{FIND}(l(i)) = \text{FIND}(l_s)$ **Then**
  $M_T = M_T + 1$
  $L_T(M_T) = i$
End if

6. *Flip the array* $L_T(t)$, i.e., put its elements in reverse order so it contains the TIP sequence of invaded sites sorted by chronological invasion time.

2.3. Efficiency

Time consuming operations for the algorithm presented in Section 2.2 lie in steps 2, 3 and 5.

When using fast algorithms (e.g. Sheppard et al., 1999; Masson and Pride, 2014) for solving the NTIP problem in step 2, the CPU time to percolation for systems having N total sites and M_{NT} invaded sites at percolation is at most $O(M_{NT}\log(M_{NT}))$.

Using the disjoint set data structure (or the equivalent Hoshen–

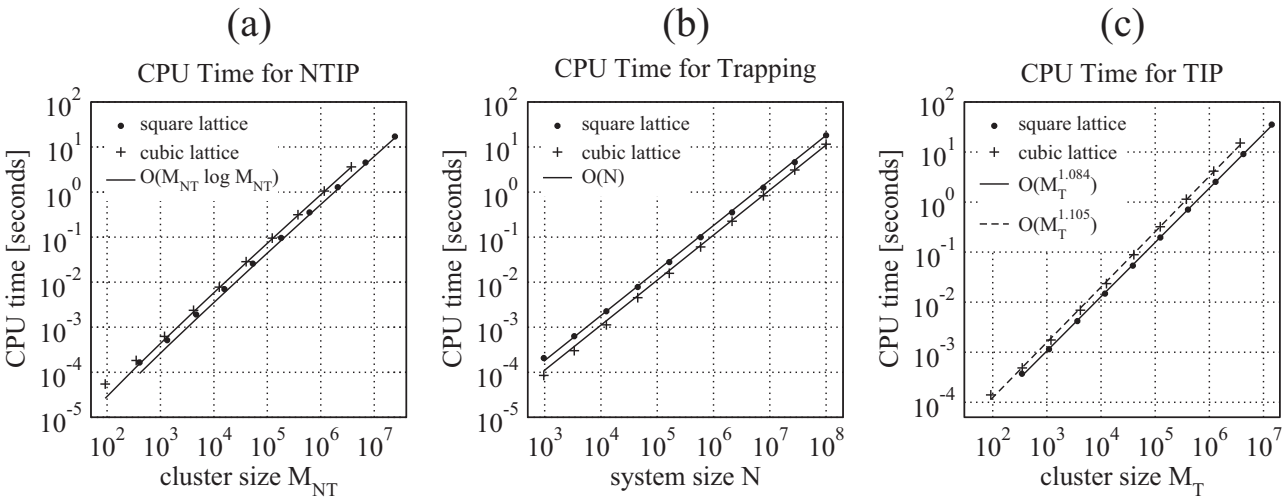

Fig. 3. (a) Observed (symbols) and predicted (solid lines) CPU time for NTIP plotted as a function of the observed cluster size M_{NT}. (b) Observed (symbols) and predicted (solid lines) CPU time for the identification and removal of the trapped sites plotted as a function of the total number of sites in the lattice N. (c) Observed (symbols) CPU time for TIP plotted as a function of the cluster size M_T. The solid lines correspond to power law least square fits, the CPU time for TIP on a 2D square lattice goes as $O(M_T^{1.084})$ and the CPU time for TIP on a 3D cubic lattice goes as $O(M_T^{1.105})$.

Kopelman Algorithm) to label the clusters formed by the defending fluid in step 3, the CPU time scales linearly with the total number of sites filled with the defending fluid, i.e. $O(N - M_{NT})$ (see e.g. Hoshen and Kopelman, 1976). In step 5, the CPU time needed to un-invade all sites is proportional to the total number of sites in the invading cluster and goes as $O(M_{NT})$.

Combining these results shows that, in the worse case, the total CPU time for solving the TIP problem goes as $O(M_{NT}\log(M_{NT})) + O(N)$.

In Fig. 3a and b, I compare the observed CPU times to the predicted values. To assess the CPU time scaling of the algorithm, I performed a total of 10^8 TIP simulations on both 3D cubic lattices and 2D square lattices of dimension $N = L^{d-1} \times 2L$, with N ranging from 10^3 to 10^8, where d is problem dimension and L is the number of sites in a given direction. For better statistics, I used a varying number of simulations n ranging from $n = 10^6$ for systems of size $N = 10^3$ to $n = 10^2$ for system of size $N = 10^8$. The estimated coefficient of variation for the measurements varies between 500 (for the smallest systems) and 10^{-1} (for the largest systems). I measured both the average CPU time for solving the NTIP problem (i.e. steps 1 and 2 in the algorithm of Section 2.2) and for the treatment of trapping (i.e. steps 3–6 in the algorithm of Section 2.2). Measurements were performed using the fortran intrinsic timing routine **CPU_TIME**. For big enough systems (say $N > 10^5$), I observe a very good agreement between the predicted CPU times and the numerical estimates. In Fig. 3c, I present the total CPU time needed to model TIP as a function of the cluster size M_T (that is the total number of sites invaded once the trapped sites have been removed). In the worse case (i.e. for 3D cubic lattices), I observe execution times of $O(M^{1.105})$ which is faster than the $O(M^{1.24})$ reported by Sheppard et al. (1999) for systems of comparable sizes. Schwarzer et al. (1999) report an execution time of 20 000 sites per second for NTIP and 250 s to grow clusters with M=500 000 in TIP, which means that their implementation of TIP is roughly ten times slower than their implementation of NTIP (for M=500 000). Using my implementation of the algorithm in Section 2.2, I observe that TIP is only two (in 2D) to four (in 3D) times slower than NTIP, regardless of the system size.

3. Numerical examples

To illustrate outputs produced by the algorithm in Section 2.2, I present some examples of site IP simulations in 2D and 3D. An advantage of the proposed algorithm is that the results for both TIP and NTIP are available at the end of the simulations. This permits easy comparison at no extra cost. Fig. 4 shows 2D clusters at percolation time for square lattices of size 128×256 and illustrates the influence of gravity on trapping. Notice that all panels present results obtained for both NTIP and TIP. Taken together, the light gray and the dark red sites give the ensemble of sites invaded in NTIP, while, the dark red sites taken alone are the sites invaded in TIP. When the invasion is occurring from lower to higher in the gravity field, and, when the invading fluid is buoyant (i.e. lighter than the defending fluid), we observe the emergence of a finger of invading fluid. This is because, as the invading fluid rises, the buoyancy-induced pressure drops $\Delta P_g = \Delta\rho g d$, where d is the height of the site, gets larger than the capillary pressure drops $\Delta P_c = \sigma/a$ at the menisci. Such finger like instabilities as observed in Fig. 4d, e and f occur when

$$\frac{\Delta P_g}{\Delta P_c} = \frac{B_0 d}{a} \gg 1, \tag{2}$$

where $B_0 = \Delta\rho g a^2/\sigma$ is the dimensionless Bond number. When B_0 gets small enough, as in Fig. 4a, (b) and (c), gravity tends to stabilize the invading front which results in an increased density of invaded sites in NTIP, and, in an increased occurrence of trapping in TIP.

In Fig. 5, I model fluid migration through an idealized 3D geological structure. The 3D model consists of a cubic lattice with dimensions $512 \times 256 \times 256$ where the pore size distribution (i.e. a_i in Eq. (1)) has been constructed as follows: (1) To mimic sedimentary rock bedding, I computed a random medium with gaussian correlation function (e.g. Klimes, 2002) where the correlation lengths a_x and a_y in the two horizontal directions are a lot larger than the one a_z in the vertical direction. (2) I created stratification by shifting the mean values of the random medium within seven horizontal layers. (3) I added a realization of the white noise to the resulting medium so the pore size distribution is locally random. (4) I modeled geological deformation by first folding the entire medium and then by shifting two blocks apart from a fault plane. (5) I increased the pore size along the fault plane to mimmic higher permeability. The invading fluid is injected from a point centered in the bottom layer and is lighter than the defending fluid. The following values have been used to compute the invasion potential in Eq. (1): $\sigma = 0.0728\ \mathrm{J\,m^{-2}}$, $\theta_c = \pi$, $10^{-5}\ \mathrm{m} < a_i < 10^{-4}\ \mathrm{m}$,

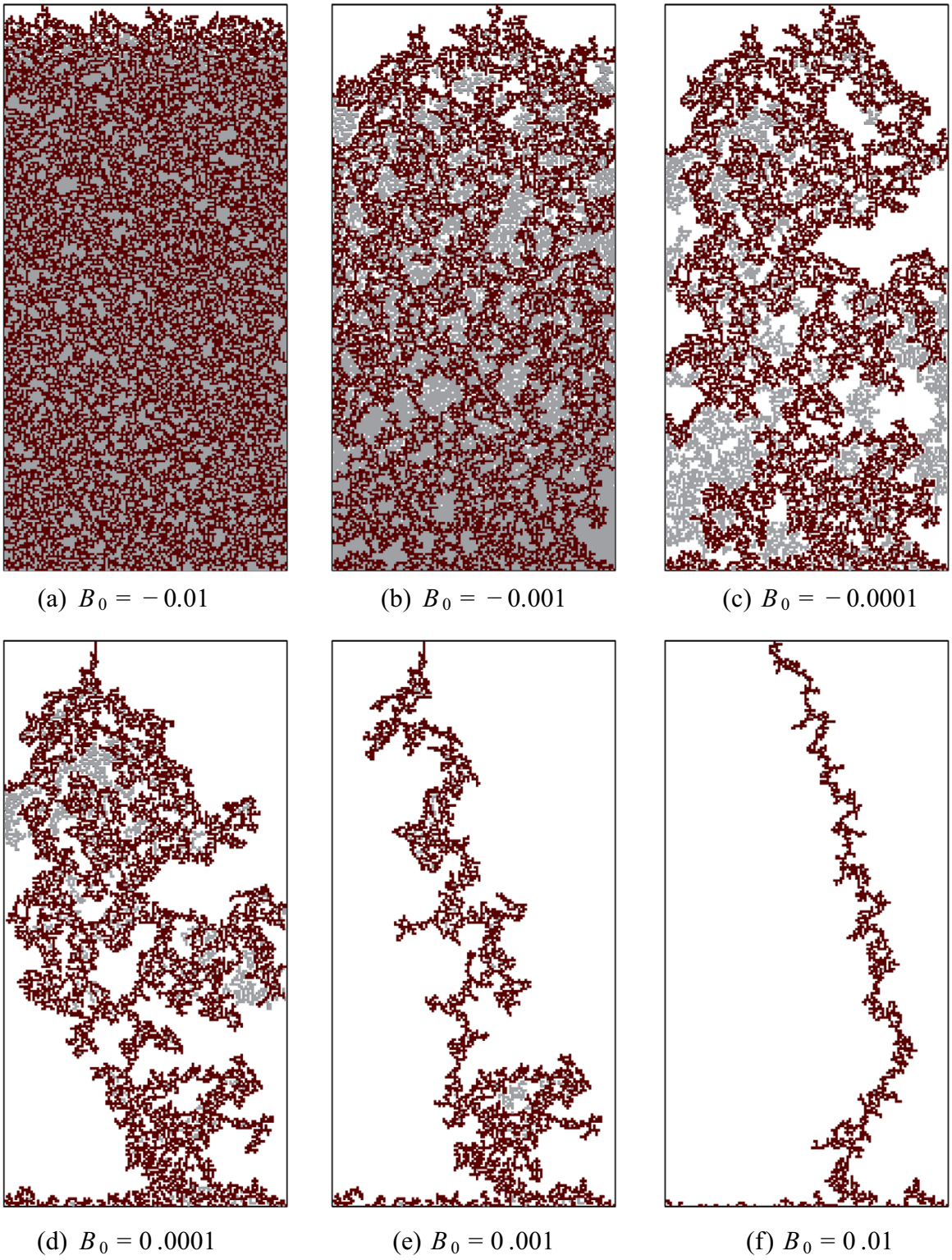

Fig. 4. Result of various 2D IP simulations with and without trapping and varying the influence of gravity. The sites in red represent the sites invaded when modeling TIP. The sites in grey represent the sites that invaded trapped region when simulating NTIP. Taken together the red and grey sites give the set of sites invaded when modeling NTIP. The number B_0 at the bottom of each panel is the Bond number, a negative value of B_0 is used to signify that the direction of the gravity field is reversed. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

$g = 9.81\ \mathrm{m\ s^{-2}}$, $\Delta\rho = 0.2\ \mathrm{kg\ m^{-3}}$, and $z_i = k_i$ (m), where k_i is the vertical index of site i in the lattice.

Fig. 5 a shows a 3D view of the cluster formed by the invading fluid at percolation time, i.e. when the invader reaches the side wall to the left. Fig. 5b corresponds to a 2D projection of the same cluster onto the xz plane. In Fig. 5a and b, the rainbow colors indicate the time at which the sites have been invaded. As time goes, we observe that the invading fluid is rising due to gravity. When the upfront finger of invading fluid reaches a layer with larger than average invasion potentials, the invading fluid spreads toward the sides. Because the layers are dipping down, the invasion potentials in the layers are decreasing toward the sides and

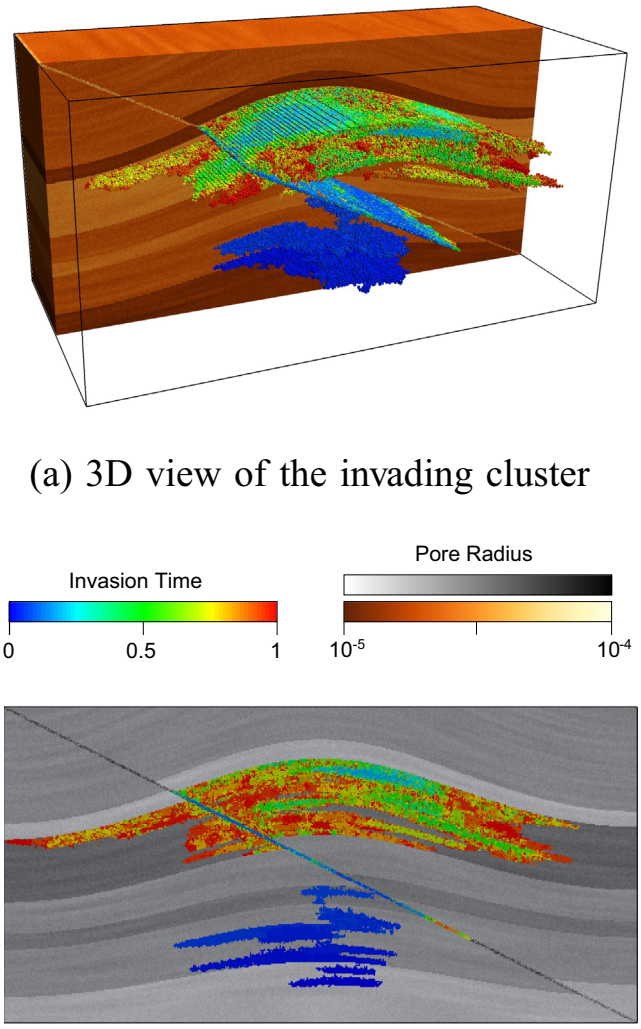

Fig. 5. Migration and accumulation of oil through an idealized anticline reservoir. The simulation is performed on a cubic lattice with dimension $512 \times 256 \times 256$. The invading fluid (i.e. oil) is injected from a point in the bottom layer and the defending fluid (i.e. water) is free to escape from the side walls (i.e. to the right and to the left in this figure). Periodic boundaries are applied in the y direction perpendicular to the projection plane xz. The rainbow colors represent the time at which the sites have been invaded (top and middle panels) or trapped (bottom panel). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

gravity eventually blocks the invasion. When the side invasion fronts get blocked, the invasion starts again at higher altitude where sites have a larger invasion potential due to the buoyancy induced pressure gradient. This process occurs many time until the invading fluid reaches the fault plane where sites have larger than average invasion potentials. The invading fluid then migrates along the fault plane until it reaches a layer with very low invasion potential that acts as an anticline trap. Finally, the invading front moves downward, spreads, and the invading fluid progressively fills the anticline. The simulation stops when the invading fluid finds an escape, i.e. when it reaches the left wall in this case.

Fig. 5 c shows a 2D projection of the trapped clusters at percolation time. The rainbow colors represent the time at which the sites got trapped. We observe that trapping is marginal when the fluid in rising freely but gets very significant when the invading front is blocked and the fluid accumulates. It is well known that in the absence of gravity trapping occurs only in 2D and is very marginal in 3D. However the present example clearly indicates that trapping needs to be accounted for in 3D when gravity is acting.

4. Conclusions

I proposed a two steps algorithm for a fast solution of the TIP problem. The first step consists of solving the NTIP problem and execution time scales as $O(M \log M)$ when using a binary tree data structure (e.g. Masson and Pride, 2014). The second step consists of identifying trapped sites a posteriori and execution time goes as $O(N)$. I measured better performance than previously proposed algorithms (Schwarzer et al., 1999; Sheppard et al., 1999). My algorithm has the advantage that both TIP and NTIP are modeled which allow for comparison at no additional cost. Readers interested in getting an implementation of the proposed algorithm written in fortran 2003 may contact me via email.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7-IDEAS-ERC)/ERC Advanced Grant (WAVETOMO). Readers interested in getting a program based on the proposed algorithm and written in fortran 2003 may contact me via email at the following address: yder.masson@cal.berkeley.edu.

Appendix A. UNION/FIND (disjoint set data structure)

In this appendix, I detail a possible implementation of the disjoint-set data structure used in Section 2.2 for labeling the cluster formed by the defending fluid. Further information can be found in e.g. Knuth (1998) and Cormen et al. (2001).

In IP, the sites filled with defending fluid form multiple clusters. The objective is to define a data structure to store the assignment of sites to clusters. The data structure needs to support the following three operations:

- **CREATE_CLUSTER**: Define a new cluster.
- **FIND**: Find the cluster that contains a given site.
- **UNION**: Merge two clusters.

To create this data structure, each site i is given a cluster label $l(i)$ and one defines a one-dimensional array L of dimension N_c, where N_c is the current number of cluster labels. Each cluster with label l

```
Function: CREATE_CLUSTER(N_c)
  N_c = N_c + 1
  L(N_c) = N_c
  Return: N_c
End Function: CREATE_CLUSTER
```

Function: FIND(l)	Function: UNION (l, m)
While $L(l) \neq l$	u = FIND(l)
$l = L(l)$	v = FIND(m)
End While	$L(u) = v$
Return: l	Return: v
End Function: FIND	End Function: UNION

Fig. A1. A basic implementation of the disjoint set procedures. The CREATE_CLUSTER procedure defines a new cluster or class with label $l = N_c + 1$. The FIND procedure returns the canonical cluster label associated with the input label (l). The UNION procedure sets an equivalence between the two input labels (l,m) which effectively merges two clusters.

```
Function: CREATE_CLUSTER(N_c)
  N_c = N_c + 1
  L(N_c) = N_c
  r(N_c) = 1
  Return: N_c
End Function: CREATE_CLUSTER
```

	Function: UNION(l, m)
Function: FIND(l)	u = FIND(l)
$m = l$	v = FIND(m)
While $L(m) \neq m$	If $r(u) = r(v)$ Then
$m = L(m)$	$r(u) = r(u) + 1$
End While	$L(v) = u$
While $L(l) \neq l$	Return: u
$n = L(l)$	Else If $r(u) > r(v)$ Then
$L(l) = m$	$L(v) = u$
$m = n$	Return: u
End While	Else
Return: m	$L(u) = v$
End Function: FIND	Return: v
	End If
	End Function: UNION

Fig. A2. An optimized implementation of the disjoint set procedures. The modified FIND procedure performs path compression while the modified UNION procedure uses weighted union. These procedures should be used instead of their simpler counterpart in Fig. A1.

points to a parent cluster with label $L(l)$. It is possible for some clusters named root clusters to be their own parent, i.e. we have $l = L(l)$. This data structure effectively creates trees of cluster labels. Using this data structure, the CREATE_CLUSTER, FIND and UNION procedures can be implemented very simply as shown in Fig. A1. Invoking $l(i)$ = CREATE_CLUSTER(N_c) effectively creates a new cluster with label $l = N_c + 1$ containing site i and sets it up as a root cluster. For a given cluster label l, FIND(l) simply follows the L (l) links up until it reaches the root cluster label and returns it. FIND($l(i)$) returns the label of the cluster containing site i. UNION(l, m) points l to m, effectively merging the two clusters. For a pair of site i and j, UNION($l(i)$,$l(j)$) merges the cluster that contains site i with the cluster that contains site j.

The FIND and UNION procedures as implemented in Fig. A1 are slow, they scale linearly with the depth of the label trees (i.e. the number of label $L(l)$ we need to investigate to reach the root label). They can be improved in two different ways. A first improvement called “path compression” is to allow the procedure FIND to collapse the tree of labels. A second improvement called “weighted union” or “union by rank” consists of modifying the UNION procedure so that the label trees are merged based on their rank. This has for effect to keep the tree's depth small and this requires an extra area $r(i)$ storing the rank of the label trees. Optimized procedures using path compression and union by rank are given in Fig. A2.

References

Babalievski, F., 1998. Cluster counting: the Hoshen–Kopelman algorithm versus spanning tree approaches. Int. J. Mod. Phys. C 9 (01), 43–60.

Chandler, R., Koplik, J., Lerman, K., Willemsen, J.F., 1982. Capillary displacement and percolation in porous media. J. Fluid Mech. 119, 249–267.

Chen, F., Shinosky, M., Aitken, J., Yang, C.-C., Edelstein, D., 2012. Invasion percolation model for abnormal time-dependent dielectric breakdown characteristic of low-k dielectrics due to massive metallic diffusion. Appl. Phys. Lett. 101 (24), 242904.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., et al., 2001. Introduction to Algorithms vol. 2. MIT Press, Cambridge.

Glass, R., Yarrington, L., 1996. Simulation of gravity fingering in porous media using a modified invasion percolation model. Geoderma 70 (2), 231–252.

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

Klimes, L., 2002. Correlation functions of random media. Pure Appl. Geophys. 159, 1811–1831.

Knuth, D.E., 1998. The Art of Computer Programming: Sorting and Searching vol. 3. Pearson Education.

Lenormand, R., Bories, S., 1980. Description d'un mecanisme de connexion de liaision destine a l'etude du drainage avec piegeage en milieu poreux. CR Acad. Sci. 291, 279–282.

Løvoll, G., Méheust, Y., Måløy, K.J., Aker, E., Schmittbuhl, J., 2005. Competition of gravity, capillary and viscous forces during drainage in a two-dimensional porous medium, a pore scale study. Energy 30 (6), 861–872.

Masson, Y., Pride, S.R., 2014. A fast algorithm for invasion percolation. Transp. Porous Media 102 (2), 301–312.

Meakin, P., 1991. Invasion percolation on substrates with correlated disorder. Physica A: Stat. Mech. Appl. 173 (3), 305–324.

Meakin, P., Feder, J., Frette, V., Jo, T., et al., 1992. Invasion percolation in a destabilizing gradient. Phys. Rev. A 46 (6), 3357.

Patzek, T.W., et al., 2001. Verification of a complete pore network simulator of drainage and imbibition. SPE J. 6 (02), 144–156.

Schwarzer, S., Havlin, S., Bunde, A., 1999. Structural properties of invasion percolation with and without trapping: shortest path and distributions. Phys. Rev. E 59 (3), 3262.

Sheppard, A.P., Knackstedt, M.A., Pinczewski, W., Sahimi, M., 1999. Invasion percolation: new algorithms and universality classes. J. Phys. A: Math. Gen. 32 (49), L521.

Toussaint, R., Løvoll, G., Méheust, Y., Måløy, K.J., Schmittbuhl, J., 2005. Influence of pore-scale disorder on viscous fingering during drainage. Europhys. Lett. 71 (4), 583.

Toussaint, R., Måløy, K.J., Méheust, Y., Løvoll, G., Jankov, M., Schäfer, G., Schmittbuhl, J., 2012. Two-phase flow: structure, upscaling, and consequences for macroscopic transport properties. Vadose Zone J. 11, 3.

Wilkinson, D., 1984. Percolation model of immiscible displacement in the presence of buoyancy forces. Phys. Rev. A 30 (1), 520.

Wilkinson, D., Willemsen, J.F., 1983. Invasion percolation: a new form of percolation theory. J. Phys. A: Math. Gen. 16, 3365.

Yang, Z., Niemi, A., Fagerlund, F., Illangasekare, T., 2013. Two-phase flow in rough-walled fractures: comparison of continuum and invasion-percolation models. Water Resour. Res. 49 (2), 993–1002.