Patch-based iterative conditional geostatistical simulation using graph cuts

Xue Li[1,2], Gregoire Mariethoz[2], DeTang Lu[1], Niklas Linde[3]

[1]University of Science and Technology of China, Department of Modern Mechanics, 230027, Hefei, Anhui, China

[2]University of Lausanne, Institute of Earth Surface Dynamics, Lausanne 1015, Switzerland

[3]University of Lausanne, Applied and Environmental Geophysics Group, Institute of Earth Sciences, Lausanne 1015, Switzerland

Abstract

Training image-based geostatistical methods are increasingly popular in groundwater hydrology even if existing algorithms present limitations that often make real-world applications difficult. These limitations include a computational cost that can be prohibitive for high-resolution 3D applications, the presence of visual artifacts in the model realizations, and a low variability between model realizations due to the limited pool of patterns available in a finite-size training image. In this paper, we address these issues by proposing an iterative patch-based algorithm which adapts a graph cuts methodology that is widely used in computer graphics. Our adapted graph cuts method optimally cuts patches of pixel values borrowed from the training image and assembles them successively, each time accounting for the information of previously stitched patches. The initial simulation result might display artifacts, which are identified as regions of high cost. These artifacts are reduced by iteratively placing new patches in high-cost regions. In contrast to most patch-based algorithms, the proposed scheme can also efficiently address point conditioning. An advantage of the method is that the cut process results in the creation of new patterns that are not present in the training image, thereby increasing pattern variability. To quantify this effect, a new measure of variability is developed, the merging index, quantifies the pattern variability in the realizations with respect to the training image. A series of sensitivity analyses demonstrates the stability of the proposed graph cuts approach, which produces satisfying simulations for a wide range of parameters values. Applications to 2D and 3D cases are compared to state-of-the-art multiple-point methods. The results show that the proposed approach obtains significant speedups and increases variability between realizations. Connectivity functions applied to 2D models transport simulations in 3D models are used to demonstrate that pattern continuity is preserved.

1. Introduction

Characterization of geological formations plays an important role in many hydrogeological applications. The geostatistical description of 2D or 3D property fields are widely used, for example, for heterogeneity representation [*De Marsily et al.*, 2005; *Klise et al.*, 2009] and transport inversion [*Hermans et al.*, 2012; *Lee and Kitanidis*, 2014]. Over the past decades, a series of geostatistical techniques have been developed to reproduce geological structures and account for geological uncertainty in numerical subsurface models [*Deutsch and Journel*, 1992; *Koltermann and Gorelick*, 1996].

Classical geostatistical approaches to reservoir characterization follow two main avenues: two-point semivariogram-based techniques and object-based methods. However, they both present limited abilities to reproduce complex spatial patterns. The two-point semivariogram has the advantage to quantify spatial variability in a mathematically tractable way [*Goovaerts*, 1998; *Isaaks and Srivastava*, 1989]. However, its application is limited to multi-Gaussian systems and hence cannot describe high-order statistics and realistic connectivity patterns [*Journel*, 1993]. Object-based methods sample geological shapes to form a simulated field [*Deutsch and Tran*, 2002; *Deutsch and Wang*, 1996]. They can produce realistic heterogeneity patterns, but it is not always possible to condition the simulations to dense datasets [*Allard et al.*, 2006; *Skorstad et al.*, 1999]. Multiple-point geostatistics (MPS) was developed to avoid such limitations [*Guardiano and Srivastava*, 1993; *Journel*, 2005; *Krishnan and Journel*, 2003].

The fundamental element of MPS is the use of training images, which are numerical descriptions of prior models in the form of images [*Hu and Chugunova*, 2008; *Mariethoz and Caers*, 2014]. While this raises questions related to the choice of a training image [*Pérez et al.*, 2014; *Suzuki and Caers*, 2008], the applications of training-image based MPS approaches have grown increasingly appealing in the last years. MPS approaches extract spatial structures of high complexity from the training image in the form of conditional distributions to generate stochastic realizations. Thus MPS realizations exhibit the same type of patterns as those found in the training image [*Guardiano and Srivastava*, 1993]. This idea was implemented in the SNESIM algorithm [*Strebelle*, 2002] by using a tree structure to store data events. Similar algorithms include GROWTHSIM which introduces a random-neighbor path [*Eskandari and Srinivasan*, 2010] or HOSIM which uses spatial cumulants for pattern extraction [*Mustapha and Dimitrakopoulos*, 2011]. Although training-image based methods have been used in many hydrogeological applications [*Hermans et al.*, 2015; *Hu and Chugunova*, 2008; *Huysmans et* *al.*, 2013; *Mahmud et al.*, 2015; *Michael et al.*, 2010], they suffer from limitations inherent to the simulation algorithms. Some important points that limit the applicability of these methods are a high computational cost, the difficulty to reproduce certain types of patterns, and most importantly the limited variability that can be recovered from a finite size training image [*Emery and Lantuéjoul*, 2014].

Several attempts have been made to decrease the computational burden of MPS. The use of multiple grids can reduce CPU cost [*Strebelle*, 2003]. Search methods using lists [*Straubhaar et al.*, 2011], or a further improved structure by combining lists and trees [*Straubhaar et al.*, 2013] decreases RAM consumption but results in higher CPU cost. Another MPS technique with low RAM requirements is Direct Sampling (DS) [*Mariethoz et al.*, 2010]. Based on Shannon sampling, DS generates simulations by sampling node-by-node directly from the training image without storing patterns. Unfortunately, the CPU cost of DS can still be high [*Meerschman et al.*, 2013]. With the continued development of hardware, parallelization strategies have been adopted that can achieve significant acceleration for stochastic simulations [*Huang et al.*, 2013a; *Huang et al.*, 2013b; *Mariethoz*, 2010; *Tahmasebi et al.*, 2012b; *Walsh et al.*, 2009]. Among the most efficient MPS algorithms are patch-based methods, whereby instead of simulating a single node from a probability model, entire patterns are pasted into the simulation grid. Several algorithms have been developed to accomplish this [*Chatterjee et al.*, 2012; *Honarkhah and Caers*, 2010; *Mahmud et al.*, 2014; *Rezaee et al.*, 2013; *Tahmasebi et al.*, 2012a; *Tahmasebi et al.*, 2014; *Zhang et al.*, 2006]. They offer significant computational gains but tend to have a lower patterns diversity than pixel-based methods [*Mahmud et al.*, 2014; *Tan et al.*, 2014], which is problematic for applications where sweeping a wide model space is important [*Caers*, 2011], such as inverse problems [*Laloy et al.*, 2016; *Lochbühler et al.*, 2014; *Saibaba and Kitanidis*, 2015].

A historical overview and comparison of MPS and computer graphics reveals that these two research fields share the purpose of stochastically generating images that present similar properties as the training image, and that they separately evolved similar algorithms such as those based on the Markov random field assumption for pixel growth [*Efros and Leung*, 1999; *Guardiano and Srivastava*, 1993] or the application of tree structures [*Strebelle*, 2002; *Wei and Levoy*, 2000]. The main difference between these two research fields is that the data-conditioning problem is not addressed in computer graphics. A successful application of a texture synthesis algorithm for conditional MPS was achieved by *Parra and Ortiz* [2011] who used a non-causal search neighbor based on the method of *Wei and Levoy* [2000]. More recently, the image quilting method [*Efros and Freeman*, 2001] was adapted for geostatistical conditional simulation under the name of Conditional Image Quilting (CIQ) [*Mahmud et al.*, 2014]. By identifying the minimal overlap error and generating an optimal cut of the patches, this algorithm avoids introducing vertical and horizontal artifacts that are often present when applying patch-based methods [*Arpat and Caers*, 2007]. However, CIQ only considers two overlapping patches with either vertical or horizontal boundaries. As such, it is limited in that it cannot consider patches of arbitrary shape, and is also not able to consider sequences of more than two overlapping patches. Moreover, it requires using a unilateral path (i.e. starting on the top left corner of the grid and proceeding with the simulation row by row), which results in difficulties for hard data conditioning [*Tahmasebi et al.*, 2012a]. Another drawback of CIQ, and patch-based algorithms in general, is the large amount of verbatim copy, which refers to large areas of the simulation being directly borrowed from the training image. As a result, the variability between realizations is reduced, since it is inversely proportional to the amount of verbatim copy [*Mariethoz and Caers*, 2014].

In computer graphics, the graph cuts approach was proposed as an alternative to the quilting approach for cutting patches *Kwatra et al.* [2003]. It is only very recently that the concept of graph cuts made its way into geostatistics. The general idea of conditional graph cuts was first described by [*Mariethoz and Lefebvre*, 2014], but not implemented nor tested until a conference presentation in 2015 [*Li and Mariethoz*, 2015], followed by an application in the context of geophysical inversion to iteratively update parts of a parameter field, hence forming a Markov chain of realizations that sample a posterior distribution [*Zahner et al.*, 2016]. While this last application was successful in solving inverse problems, it is restricted to simulation outputs with a size smaller than the training image and it has not been investigated for direct conditioning to point data or for 3D cases. In this paper, we further develop the graph cuts approach into a complete geostatistical simulation method that enables conditioning, artifact minimization and 3D simulation. The graph cuts framework offers an efficient way of defining optimal cuts through the grid according to the chosen cost function and allows keeping a record of past cuts that can be used within an iterative simulation strategy. This allows removing artifacts caused by previous cuts and increasing the variability by introducing new cuts. Since the first submission of this manuscript, another graph-cuts based method has been proposed [*Tahmasebi and Sahimi*, 2016a; b] that was developed independently of our work. It presents similar concepts, but with different implementation details. For example in our paper a different strategy is used for ensuring local conditioning. We also propose a new merging index to quantify the pattern diversity generated by the simulation algorithm. Exhaustive tests are carried out, including parameter sensitivity analysis as well as comparison with other MPS algorithms.

The next section presents the main concepts of the graph cuts-based simulation method developed in this paper. Then we outline the new algorithm for cases of unconditional and conditional simulation. A parameter sensitivity analysis and applications to 2D and 3D cases are demonstrated in section 3.

2. Methodology

2.1.Graph cuts principle

The basic concept that underlies the method presented in this paper is that a Cartesian grid (i.e., a 2D or 3D matrix of property values) can be represented as a graph, where each pixel is represented as a vertex which is connected to its neighbors by edges (i.e., two adjacent vertices in the grid representation are connected by an edge in the graph representation). While both grid and graph representations are equivalent, using a graph representation allows using efficient methods and algorithms that have been developed in graph theory. Herein, we are focusing on graph cuts methods, which are designed to partition a graph in an optimal manner according to a cost function. In the context of a patch-based MPS method, the graph representation allows using graph cuts to cut patches such as to minimize artifacts and overlap errors.

Let us consider a graph $G(\mathbf{V}, \mathbf{E})$ with vertices $\mathbf{V}$ and edges $\mathbf{E}$. Optimal graph cuts are derived by using a flow analogy, whereby the graph is seen as analogous to a pipe network containing a source and a sink, and a set of non-terminal vertices. Each edge has a non-negative cost (a capacity) and, following the flow analogy, a fluid can flow from the source vertex towards the sink vertex (note that flow is used here as an analogy only and it does not represent a physical model).

A cut separates the vertices of the graph into two sets: one attached to the source and the other attached to the sink. The cost of cutting an edge is equal to its capacity. The min-cut problem consists in finding the cut that has the minimum total cost among all possible cuts throughout the graph. The Ford-Fulkerson method [*Ford and Fulkerson*, 1956] offers a fast implementation based on the max-flow/min-cut theorem. This theorem specifies that the minimum cost cut can be found by identifying the smallest capacity edges that constrain the total flow through the graph (i.e., the bottlenecks in the flow system). Figure 1 shows a simple example of a graph construct with the min-cut highlighted in green and the lines widths reflecting the capacity of the edges.

Once the cut is identified, the nodes are labeled as belonging to either the source or the sink. This technique is equally applicable to 2D surfaces or 3D volumes since the labeling process is only influenced by the connection of vertices in the graph, which can represent a grid of any dimensionality.

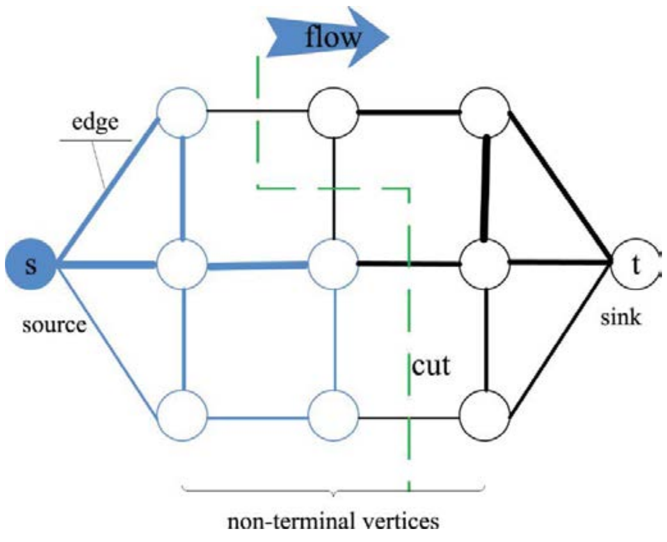


Figure 1. Example illustrating the structure of a graph, with the min-cut of optimal cost highlighted in green (modified from Kwatra 2003). The blue vertex labeled s denotes the source; the white vertex labeled with t denotes the sink, and the un-labeled vertices denote non-terminal vertices. The width of the edges represents the flow capacity. The max-flow/min-cut method is the analogue of a flow process from the source to the sink, through non-terminal vertices. The minimum cost cutis shown as the green line.

Many algorithms such as the push-relabel strategy [*Goldberg and Tarjan*, 1988] or the augmenting path strategy have been developed for solving min-cut/max-flow problems efficiently. The fast augmenting path algorithm proposed by Boykov and Kolmogorov [*Boykov and Kolmogorov*, 2004] is used in this paper.

2.2. Using graph cuts for patch-based geostatistical simulation

Similar to the applications of graph cuts in computer graphics [*Efros and Freeman*, 2001; *Lasram et al.*, 2012] patch-based MPS algorithms extract patches from a training image and sequentially assign the selected patches to the simulation grid. A distance function is used to find a patch that corresponds to the overlap with the previously simulated patches [*Arpat and Caers*, 2007; *Tahmasebi et al.*, 2012a; *Zhang et al.*, 2006].

The main attraction of using graph cuts in this context is to find the best cut of the overlap that maximizes the spatial coherence when stitching the patches in the simulation. To this end, the grid points in the overlap area between two patches *A* and *B* constitute the vertex set. Nodes representing adjacent pixels are connected by edges. Since the cut has to pass somewhere through the overlap area, source and sink nodes are connected to nodes representing grid cells that should be constrained by the old and new patch respectively. Once the minimum cut is determined, the nodes that are labeled as being attached to the source are attributed the values in patch *A* and the nodes attached to the sink are attributed the values in patch *B* (see Figure 2a). More specifically, the overlapping part of *A* and *B* is considered as a grid graph in which any two adjacent vertices *u* and *v* of the overlap are connected by an edge *e*(*u*,*v*). The value at vertex *u* is denoted as *A*(*u*) and *B*(*u*). The absolute difference between vertices values is denoted $\delta(u)$:

$$\delta(u, A, B) = \left| A(u) - B(u) \right|. \tag{1}$$

The capacity (cost) of the edge connecting *u* and *v* is defined as the sum of the absolute differences of the two vertices:

$$C_E(u,v,A,B)=\delta(u,A,B)+\delta(v,A,B)\,. \tag{2}$$

Using these costs, graph cuts can be applied to optimally stitch the patches together in order to update the simulation grid. In contrast to quilting-based methods, [e.g. *Mahmud et al.*, 2014; *Tahmasebi et al.*, 2012a], here the old cuts can be included in the overlap and influence the subsequent cuts, following the strategy proposed by *Kwatra et al.* [2003]. The main idea is to add a new vertex (termed seam vertex in this paper) in the graph where a cut passes, in order to store the quality of the old cut. If a seam vertex indicates a poor quality cut, it suggests that when a new patch is considered at this location it is preferable that the parts of the old cut containing high cost seam vertices (and the surrounding poor quality patterns) are erased and replaced. Figure 2 illustrates this with a situation where there is an initial cut between patches *A* and B (Figure 2b), and a third patch *C* needs to be added that accounts for the existing cut (Figure 2c). Once a cut is made between *A* and B, additional seam vertices are added along the cut (dashed line on Figure 2c). Figure 2d illustrates the construction of an edge with the addition of seam vertex *m*, connected to the adjacent vertices *u* and *v*. Two edges *e*(*u*,*m*) and *e*(*m*,*v*) connect each new seam vertex to the pre-existing adjacent vertices and replace the previous edge *e*(*u*,*v*).

If a subsequent patch *C* overlaps this cut, a new edge *e*(*m*,*t*) is connected to the sink terminal (*t*) of the new patch *C* and assigned the capacity of the old cost. The capacity of *e*(*u*,*m*) is calculated using:

$$C_E(u,m,A,C)=\delta(u,A,C)+\delta(v,A,C)=C_E(u,v,A,C)\,, \tag{3}$$

and the capacity of *e*(*m*,*v*) is defined in a similar way:

$$C_E(m,v,B,C) = \delta(u,B,C) + \delta(v,B,C) = C_E(u,v,B,C)\,. \tag{4}$$

According to the max-flow/min-cut theorem, when patch *C* overlaps the seam vertices, at most one of these three edges (*e*(*m*,*v*), *e*(*u*,*v*) or *e*(*m*,*t*)) has to be cut in order to find the cut with the lowest cost. This results in the four possible cases that are listed in Table 1.

The process of adding and cutting patches can be iterated to further reduce the overall cost of the cuts throughout the simulation. Since the cost of the cut reflects the differences between patches, a larger value implies a higher probability of discontinuities and artifacts. By searching for high cost values, one can define the location of new patches to be added that will reduce any remaining artifacts.

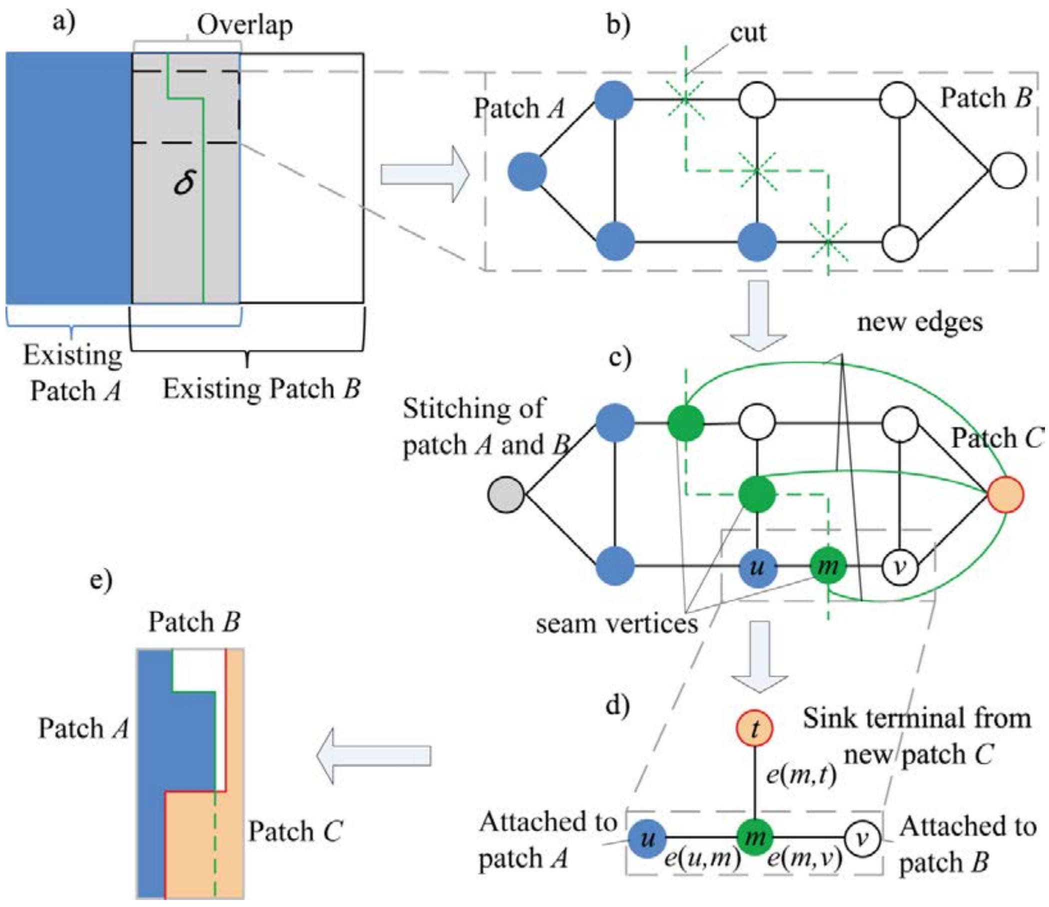


Figure 2. The process of incorporating the previous cut in the subsequent graph cut problems. a) Overlap (in gray) with previous cuts (in green); b) an enlarged view of

previous cuts; c) additional seam vertices and edges; d) an enlarged view of a seam vertex; e) cuts combining three patches.

Table 1. Four possible cases of graph cuts that account for previous cuts

Edge cut when placing a new patch	Labels of (u,m,v) (0=attached to source terminal, 1=attached to sink)	Changes	Cost of edge that is cut
$e(m,t)$	(0,0,0)	old cut and the seam vertex are preserved	$C_E(u,v,A,B)$
e(u,m)	(0,1,1)	the seam vertex is preserved but its cost is updated	$C_E(u,v,A,C)$
e(m,v)	(0,0,1)		$C_E(u,v,B,C)$
None	(1,1,1)	remove the old cut and the seam vertex	None

2.3. Unconditional simulation

In the case of unconditional simulation, the proposed algorithm consists of two steps. The first step is to simulate an initial grid by tiling and cutting patches, but with cuts defined by the graph cuts algorithm. If the realization presents significant local artifacts, a second step consists in iteratively adding and cutting patches to improve the continuity of patterns and remove artifacts.

2.3.1. Initial simulation step

Graph cuts allow for flexibly arranging patches of arbitrary shape along a random path. This flexibility is very valuable when conditioning or iteratively reworking a realization. However, the two-steps strategy adopted here only calls for an initial approximatively conditioned realization which will be further conditioned and improved later. For this first step, it is sufficient to proceed as in other patch-based methods such as CIQ or CCSIM [e.g. *Arpat and Caers*, 2007; *Chatterjee and Dimitrakopoulos*, 2012; *Honarkhah and Caers*, 2010; *Mahmud et al.*, 2014; *Tahmasebi et al.*, 2012a] which use square patches, a fixed overlap size and a unilateral path similar to other patch-based algorithms. Numerical tests (nor shown here) have showed that using a unilateral path is preferable in the first step as it allows good preservation of the training image patterns [*Daly*, 2004], albeit with poorer conditioning. The conditioning is then taken care of in the second step, and this is when a non-unilateral path is used. The procedure for this first step is as follows:

1) Input the training image, the simulation grid, initialize parameters (patch size p, overlap size o, number of candidates ε), and define a list for the seam vertices to store the location and cost of cuts.
2) Start the simulation by randomly choosing a patch from the training image and placing it in the simulation grid.
3) Iterate until the entire grid is simulated:
    a) Search the training image for subsequent patches according to the similarity of the overlap area. The ε candidates that best fit the overlapping part of the patch are selected using the distance function:

$$d_o = \frac{1}{N}\sum_{i=1}^{N}\delta_i \,, \tag{5}$$

where N is the number of pixels in the overlap area and δ_i is the error value at vertex i calculated in Eq. (1). Patches with a lower distance value have a better fitto the overlap.

b) Uniformly select one of the ε best fitting candidate patches to use in the simulation grid.
c) If old cuts are included in overlap grid, add corresponding seam vertices and assign C_E to their edges.
d) Use the graph cut algorithm to define the optimal cut between old and new patches. After the cut has been applied, the remaining part of the patch is denoted partial patch, and is pasted in the simulation grid. The seam vertices list and the cut map are updated according to Table 1.

Figure 3 illustrates the process and presents the corresponding cost map which tracks the cost of each seam vertex.

Note that for patches 2 and 3 (Figure 3e-3l), the result is similar to what would be obtained using Image Quilting, because in this initial simulation stage we chose to consider square patches and a unilateral patch. For patch 4, the combination of vertical and horizontal overlaps would be handled in Image Quilting by performing two independent cuts. In contrast, Graph Cuts is able to define a single optimal cut through the combined overlap shape (figure 3n).

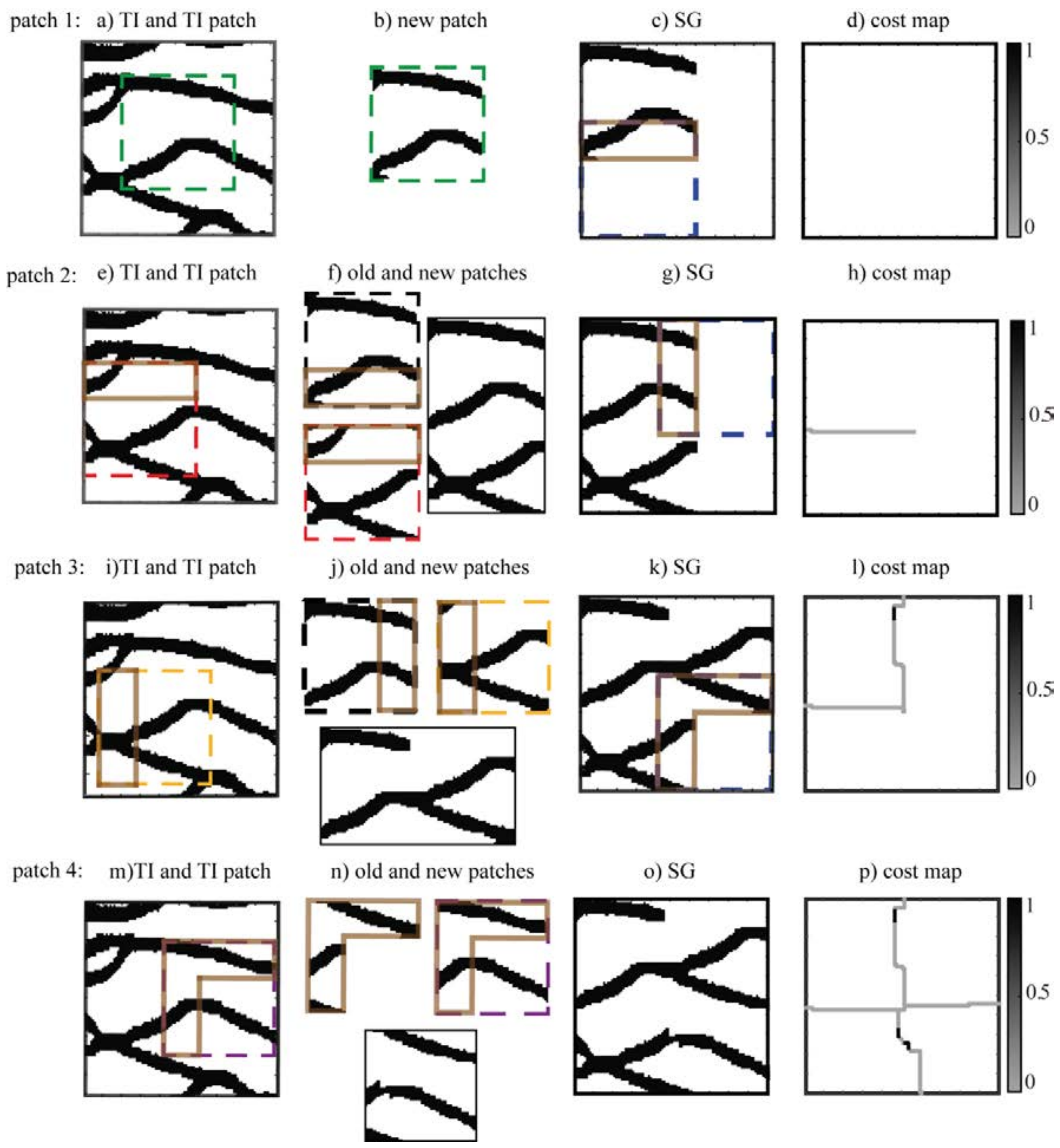


Figure 3. The initial simulation step illustrated by a realization consisting of 4 patches shown in each row. For the simulation of each patch, an overlap (shown with solid brown lines) is defined. One best matching patch is extracted from the Training Image, shown with colored rectangles. A cut is defined inside the overlap, and the resulting partial patch is pasted in the Simulation Grid. The cost maps show the location and cost for each cut.

2.3.2. Second simulation step: iterative re-simulation of patches

A high value in the cost map represents a high likelihood of discontinuities and artifacts (see Figure 3k-l and o-p). Such discontinuities can be removed by placing new patches that are centered on the high cost values. In order to increase spatial continuity in the following cuts, a rejection/acceptance condition is defined. If the new cut reduces the mean cost of cuts, the update is accepted, otherwise it is rejected. The complete procedure to accomplish this iterative re-simulation is as follows:

1. Define a stopping average cost value $C_{\min}$ and a size range *r*.
2. Iterate until the mean cost of the cuts over the grid, $\overline{C}$, is lower than $C_{\min}$, or until a maximum number of iterations *l* has been reached:
   a) Scan the cost map to identify vertices of high cost C_E (here defined as higher than the mean cost of the initial simulation, section 2.3.1) and label the connected components of the high cost regions (The regions with a cost above 80% of the highest cost value is here defined as a high cost region). This is in contrast to [*Zahner et al.*, 2016] who use high cost regions to define the terminals of the graph cut problem.
   b) Randomly choose one among the ε largest connected components as the center of the new patch. The size of the new patch is defined as the size of the chosen connected component multiplied by *r* in each dimension.
   c) Use Eq. (5) to find the ε closest matches in the training image
   d) Run the graph cuts algorithm to define which part of the patch should be pasted in the simulation grid.
   e) Accept the new patch and the associated cut only if it reduces $\overline{C}$.

Note that *r* should be larger than 1 in order to include in the new patch the entire connected component of large error values (*r*=1.3 is used in this paper).

Figure 4 shows one iteration starting from the model in Figure 3. It results in improved structure continuity due to the addition of a new patch in the lower part of the domain, thereby resulting in a cost reduction in this area.

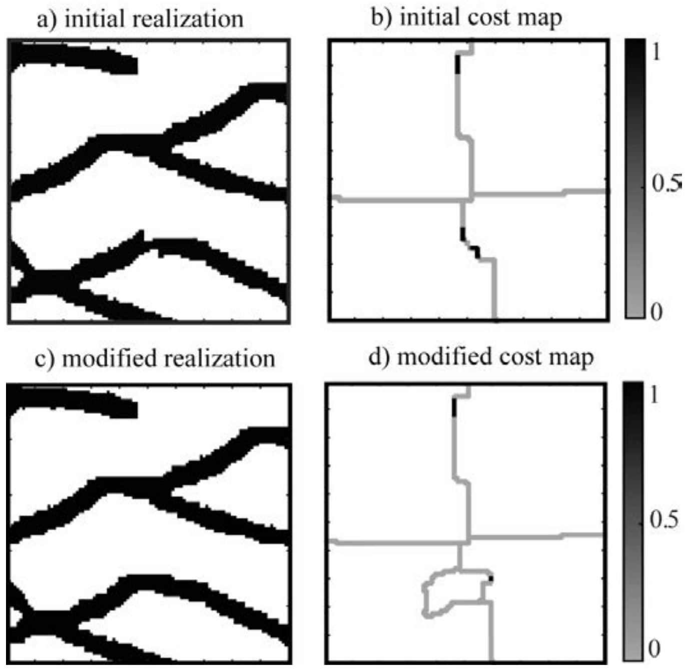


Figure 4. One iteration of patch replacement. a) Initial realization (same as Figure 3k; b) initial cost map (same as Figure 3l); c) Result after patch replacement (note that the artifact on the channel in the lower portion of the image has disappeared); d) cost map of c.

2.4. Conditional algorithm

Conditioning to point data can be challenging in patch-based simulation, especially when several conditioning data fall within a patch, resulting in a data configuration that may not be present in a finite size training image. By iteratively cutting and stitching patches, the two-step simulation described above can be extended to address point data conditioning.

2.4.1. Conditional distance

For conditional simulation, the formulation of the distance between patches is modified such that the search for a subsequent patch accounts for both 1) overlap similarity and 2) conditioning data matching. To do so, we modify the distance function in Eq. (5) to obtain an approximately conditioned realization, which will be later modified for exact conditioning. The modified distance function d_m is reformulated as:

$$d_m = (1-w)d_o + wd_c, \text{ with } d_c = \frac{1}{N_c}\sum_{c=1}^{N_c}\delta_c \qquad (6)$$

with δ_c denoting the absolute difference between conditioning data and the simulation and N_c the number of conditioning points inside the patch. The relative importance of the two terms is determined by the weight w in the interval [0,1]. A value of w=0 means that the conditioning data are not taken into account. Conversely, w=1 results in the conditioning data being honored as much as possible while ignoring continuity between patches.

2.4.2. Exact conditioning

While the use of Eq.(6) allows selecting patches that are generally coherent with the conditioning data values, it does not ensure that all conditioning data are honored exactly if a continuous training image is used or if several conditioning data lie in the same patch, especially when using large patches or when the density of conditioning data is high [*Mahmud et al.*, 2014; *Zhang et al.*, 2006]. This will inevitably result in some of the conditioning data not being honored. Increasing w can result in a better conditioning, but this comes at the price of decreased pattern continuity where patches overlaps, which is not desired. One solution consists in splitting the patch into smaller patches, each one containing less conditioning data, hence making it easier to find a match [*Tahmasebi et al.*, 2012a]. In this work, we take a different route that exploits the flexibility of the graph cuts approach.

By considering the non-matched conditioning data as artifacts, improved conditioning can be accomplished by iteratively replacing patches analogously to how cut artifacts were treated in section 2.3.2. After a first simulation step that uses the distance in Eq. (6), the conditioning data are divided into two groups: the matched and the non-matched data. For continuous variables, if the exact same value as the conditioning data is absent in the training image, a value within a specified error tolerance is defined as being an “exact” match. A 5% tolerance is used for the tests presented in this paper. At each non-matched conditioning position, a new patch is selected from the training image and used to define a new cut that is centered on the conditioning point. The procedure for exact conditioning is:

Iterate until all conditioning data are honored:

1. Randomly choose one non-matched conditioning datum as the center of the patch.
2. Use a two-stage search in the training image.
    a) Find patches with center values that are the same (or within the specified threshold) as the conditioning datum and select them as candidates.
    b) Calculate the distance between the simulation patch and the candidates using Eq.(6). One of the ε candidates with lowest distance is randomly chosen as the new patch.
3. Define $u=\{u_j,\ j=1,\ \ldots,\ N_c\}$ as the conditioning positions inside the overlap area between the old patch A and the new patch B, with values $A(u)$ and $B(u)$, the conditioning data values as $F(u)$. If a point u_j satisfies

$$A(u_j) = F(u_j) \quad and \quad B(u_j) \neq F(u_j)\,, \tag{7}$$

this point is defined as the source because it is the value of the old patch that is correct and should be preserved. Conversely, if a point u_j satisfies

$$A(u_j) \neq F(u_j) \quad and \quad B(u_j) = F(u_j), \tag{8}$$

it is the new patch that contains the correct conditioning data value and the point is therefore defined as sink.

Note that if both the old and new patch have the correct (or incorrect) value at a conditioning position, this position is defined as unlabeled since conditioning is preserved (or needs to be removed) in both patches.

4. Update the simulation, seam vertices list and the non-matched conditional data list.

Note that the terminal definition described in point 3 aims at better conditioning since it freezes the conditioning data that are correctly matched. But if only a single point is defined as terminal, a cut consisting of an isolated point may occur, leading to poor local conditioning. One solution to the problem of isolated points is to extend the terminal according to the conditioning error. Instead of the conditioning data only, the neighbors of the conditioning data that are within an error tolerance are also taken as terminal. Such a zone is found in the interval [E_l, E_h], which is calculated by:

$$E_l = (1 - q)\delta_{u_i}, \tag{9}$$

$$E_h = (1 + q)\delta_{u_i}, \tag{10}$$

where δ_{u_i} is the cost at the conditioning point u_i calculated by Eq. (1), and q denotes a tolerance (expressed as a percentage) when setting the location of the terminals. As a result, the source and sink terminals will be further apart when q is large. On the other hand, δ_{u_i}

represents a relatively small value because it is based on patches similar to target conditioning, therefore ensuring that the terminal is not excessively large. The extended terminal results in a better local conditioning but a higher possibility of artifacts which can be removed by subsequent iterative improvements without adverse effects on the final results. In this paper we use a value of q=50% for all tests.

The conditioning procedure is illustrated in Figure 5. One old patch centered on a non-matched conditioning datum is taken from the previous simulation (Figure 5a) and used to find a similar patch that matches this conditioning datum (Figure 5b). Four conditioning data are contained in this patch and are separated in two groups as shown in Figure 5c. Only the conditioning datum represented by a square has a different value between the old and the new patch. This point is defined as terminal as described in step 3 above. The comparison between defining the single conditioning datum ("simple terminal", Figure 5d) and the extended connected component (Figure 5g) is shown in Figure 5e-f and Figure 5h-i. A conditional pattern is cut with the extended terminal that avoids the isolated cuts and generally improves conditioning. Consequently, this strategy is used in this paper for the iterative processes whenever conditioning data are present.

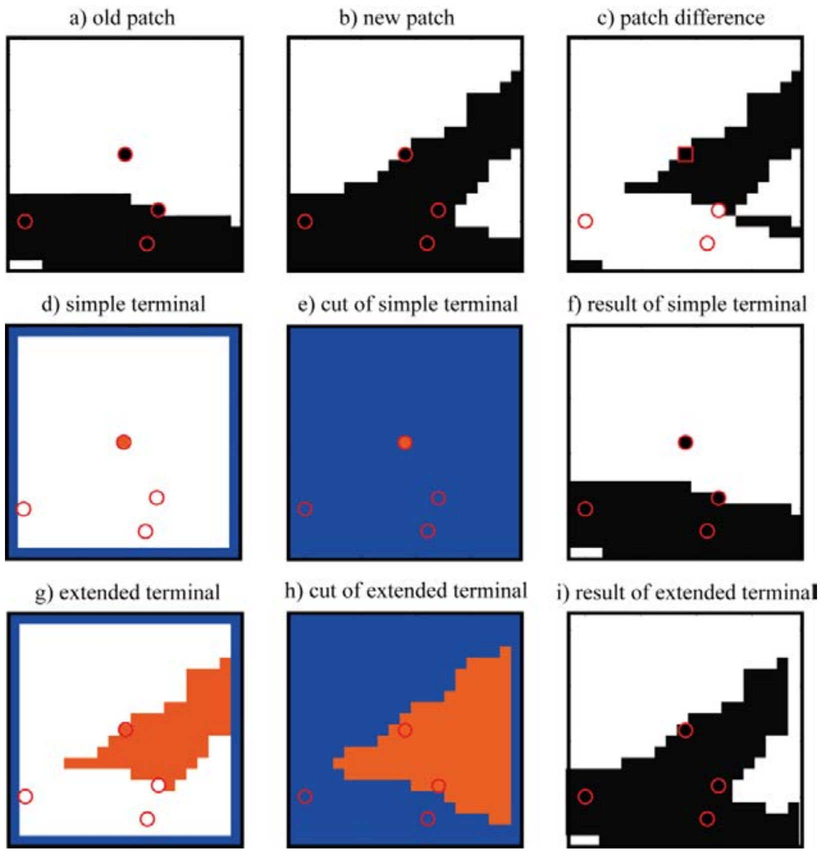

a) old patch
b) new patch
c) patch difference
d) simple terminal
e) cut of simple terminal
f) result of simple terminal
g) extended terminal
h) cut of extended terminal
i) result of extended terminal

Figure 5. Conditioning procedure and comparison of different terminal definitions. Points belonging to source and sink are shown in blue and red respectively. a) Old patch from previous simulation with a non-matched conditioning datum in the center; b) new patch from the training image; c) the difference between the old and the new patch; d) simple terminal with the non-matched conditioning data as terminal; e) cut with simple terminal of d); f) result of using the simple terminal; g) extended terminal; h) cut with extended terminal of g); i) result of using the extended terminal. In d) and g), blue indicates vertices belonging to the source terminal, red indicates vertices belonging to the sink terminal and white indicates non-terminal vertices. These non-terminal vertices are separated into either source or sink with the max-flow/min-cut method in e) and h).

Accurate conditioning often results in an increased total cost around the conditioning data. This is a consequence of a trade-off between a faithful reproduction of the training image patterns and the conditioning to data that are not fully consistent with the training image. One advantage of the proposed method is that these high cost cuts are removed by using the iterative replacement of patches described in section 2.3.2, with the distance defined in Eq. (6) for the selection of patches.

Figure 6 illustrates the process of a conditional simulation on the same training image as in Figure1a. The conditioning data are displayed in Figure 6a. Figure 6b shows the initial simulation where data conditioning is imperfect, with a large number of conditioning data that are not satisfied. After 6 conditioning iterations (see section 2.4.2), implying the addition of 6 new patches, the simulation shown in Figure 6c satisfies all conditioning data, but at the price of increased discontinuities. By adding more patches at the remaining high cost locations, the spatial continuity is improved. Figure 6d shows the final result after 20 iterations that seek to improve the spatial continuity. Figure 6e displays the evolution of the conditioning error and the mean cost over the iterations. Note that an initial increase in the mean cost is required to achieve conditioning. The mean cost is reduced in subsequent iterations. This behavior is also observed when simulating continuous variables (not shown).

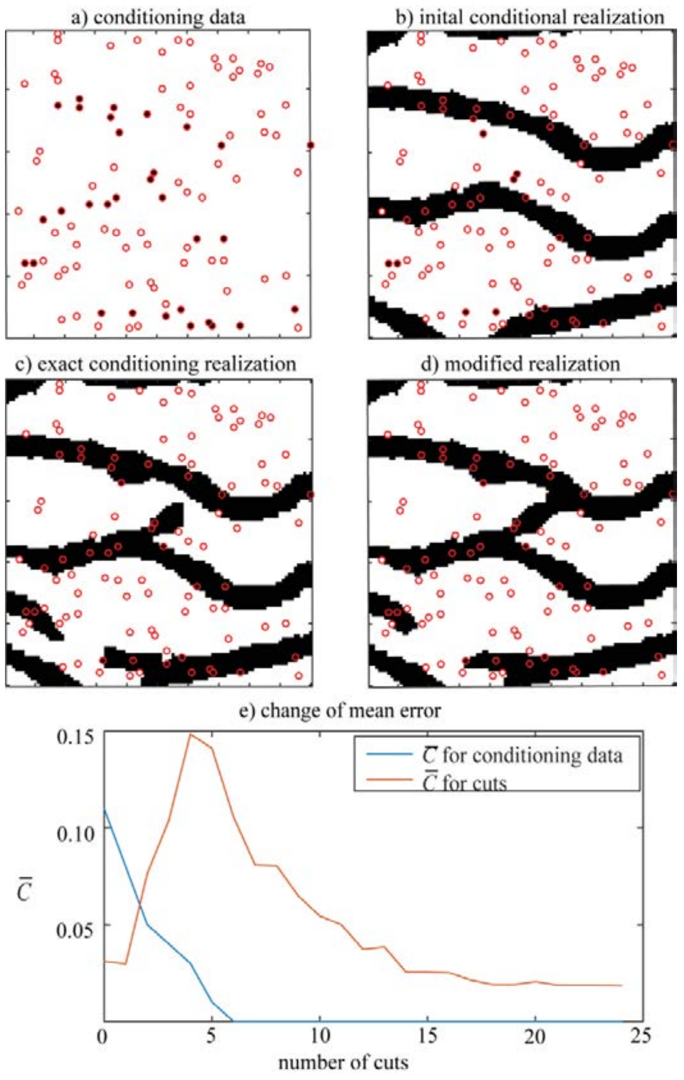


Figure 6. The iterative procedure for a conditional simulation. a) 100 conditioning data; b) initial conditional simulation with 11 non-matched conditioning data; c) simulation conditional to all data (6 iterations); d) simulation after further modifications of high cost areas (20 iterations); e) mean error for conditioning data and mean cost for all cuts, as a function of the iteration number.

3. Numerical tests

3.1. Parameter sensitivity

The main parameters of the proposed method include the patch size *p*, the overlap size *o* and the number of replicates *ε*. This section determines optimal values for these parameters using different test cases and shows their impact on unconditional and conditional simulation results. The machine used has 8 Gb of RAM and an Intel i5-4210M with 2.60 GHz CPU.

Besides the quality of the reconstructed patterns, we also focus on avoiding the occurrence of verbatim copy, which refers to a situation in which values that are next to each other in the training image are also next to each other in the simulation, thereby resulting in an artificial reduction in the variability. Verbatim copy can be measured with index coherence maps [*Honarkhah and Caers*, 2010; *Mariethoz and Caers*, 2014]. An index coherence map indicates for each simulated pixel its original location in the training image. By analyzing index coherence maps, it is possible to estimate the proportion of the simulation that is a verbatim copy of the training image. To this end, we introduce a merging index *M* that quantifies the variability in a realization based on the index coherence map:

$$M = \frac{S-1}{R-1}, \tag{11}$$

Where *S* is the number of patches identified from the index coherence map and *R* is the number of patches used during the simulation. Coherent patches (i.e. next to each other in the training image and in the realization) are identified as a single patch in the index coherence map although they have been placed after one another during the simulation. As a consequence, $S \leq R$. The merging index $M \in [0,1]$ therefore measures the fraction of cuts that creates variability, 0 denoting a complete reproduction of the training image and 1 meaning that each cut adds variability to the realization.

In this section we use a training image based on a satellite image of the Lena delta, Russia (Figure 7a) of size 500 by 500 pixels (size of scene approx. 10km by 10km). The corresponding map of pixel indices is shown in Figure 7b, which corresponds to the index of each point in the TI calculated by:

$$I = I_x + I_y \times \dim x + I_z \times \dim x \times \dim y \,, \qquad (12)$$

where I is the pixel index, I_x, I_y and I_z (default as 0 for 2D cases) are the coordinate in each direction and dim x, dim y and dim z are the dimensions in each direction (i.e. there is a horizontal and vertical gradient in Figure 7b).

The realizations have the same size as the training image. We applied l=200 iterative patch modifications and found no improvements in 8 out of 10 realizations. One or two new patches were accepted in two of the realizations, meaning that only minor improvements were possible. This shows that in the unconditional case, the graph cuts is sufficient to produce very low overlap errors already in step 1. Therefore, we only show the step 1 realizations and the corresponding CPU time.

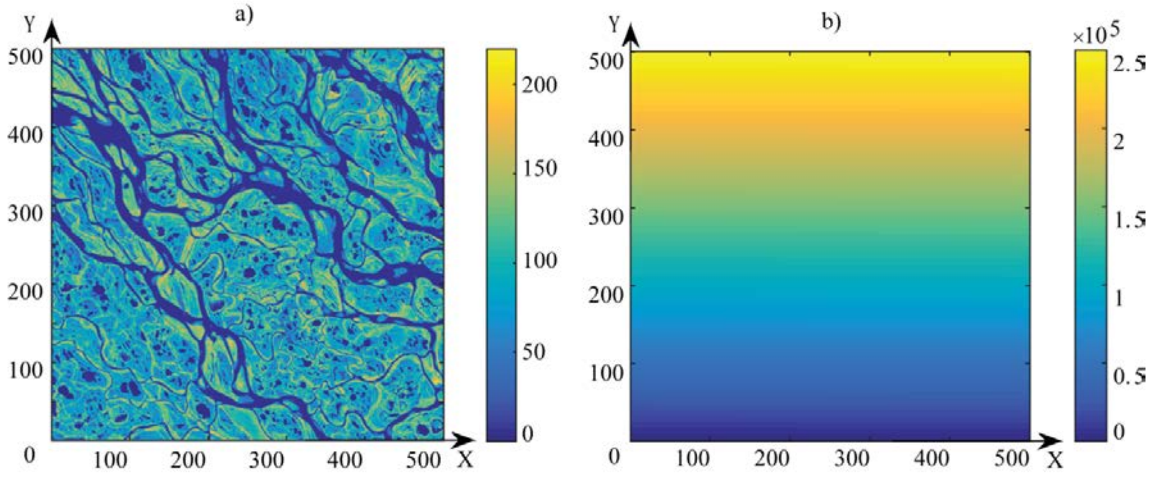


Figure 7. a) Training image (Lena Delta, Russia). b) linearly increasing indices of the pixels the training images (there is a vertical as well as a horizontal gradient). Index coherence maps are obtained by mapping these indices in the realizations.

3.1.1. Number of candidates

Increasing the number of candidates is the most straightforward means to increase the variability of patterns in the realizations. A sensitivity analysis is carried out with fixed parameter values *p*=85 (pixels), *o*=25 (pixels) and varying number of replicates ε=1, 10, 50. Figures 8a-c show the resulting realizations. The corresponding index coherence maps are shown in Figure 8d-f. In the index coherence maps, the lateral variation in color is difficult to see, therefore red boundaries are used to identify coherent patches: the patches on either side of a red line are not adjacent in the training image. It shows that increasing the number of replicates can introduce more possible candidate patches, which avoids the occurrence of verbatim copy of the training image. The merging index computed over 10 realizations, displayed in Table 2, leads to the conclusion that too small ε values result in low merging index values, indicating verbatim copy. On the other hand, there is little benefit in setting a large ε value (more than 50 in this case) as it only brings minor improvement in the merging index, at the risk of using suboptimal patches in the simulations.

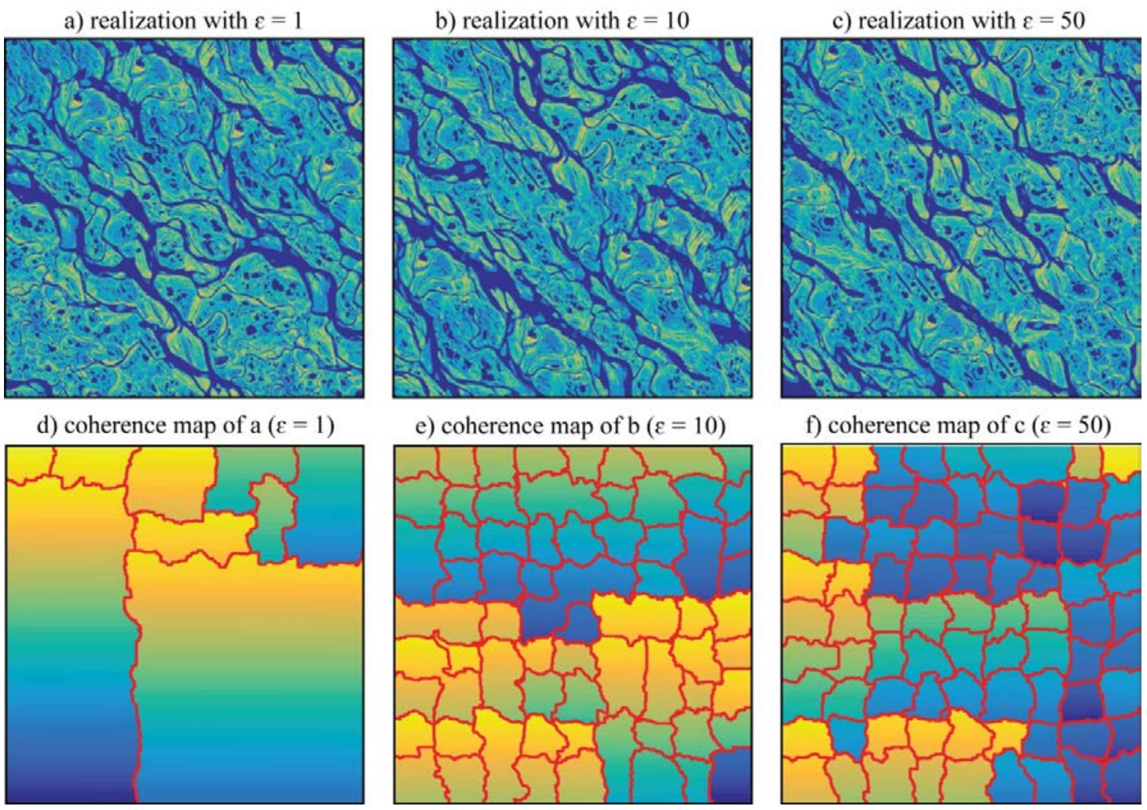


Figure 8.a)-c) Realizations with *p*=80, *o*=25 and ε =1, 10, 50; d)-f) index coherence maps for a)-c) with the red boundaries showing coherent patches.

Table 2. Average merging index for 10 realizations with fixed *p*=80, *o*=25 and varying ε.

Nb. of candidates (ε)	1	10	50
Nb. of cuts	81	81	81
Average nb. of patches identified from index coherence map	9.2	66.5	78.7

Average merging index (*M*)	0.1025	0.8188	0.9713
Average CPU time (*s*)	23.79	24.34	24.42

3.1.2. Patch size

The patch size is a critical parameter for many other patch-based simulation techniques. Figure 9a-c show realizations with different patch sizes *p*=40, 80, 120, while the other parameters are maintained at *o*=25 and *ε*=10. The corresponding index coherence maps are displayed in Figure 9d-e. Table 3 shows the merging index computed over 10 realizations, along with the average CPU time for each simulation. With *p*=40, 1521 small patches are used, which do not allow a good reproduction of the connectivity. Moreover, only 53.27% of the patches are introducing variability. A larger patch size can capture large-scale patterns and also results in an increased creation of new borders. It also has the advantage that the CPU time is drastically reduced. However, it is clear that using larger patches also reduces the diversity of small-scale patterns, which are identical as in the training image.

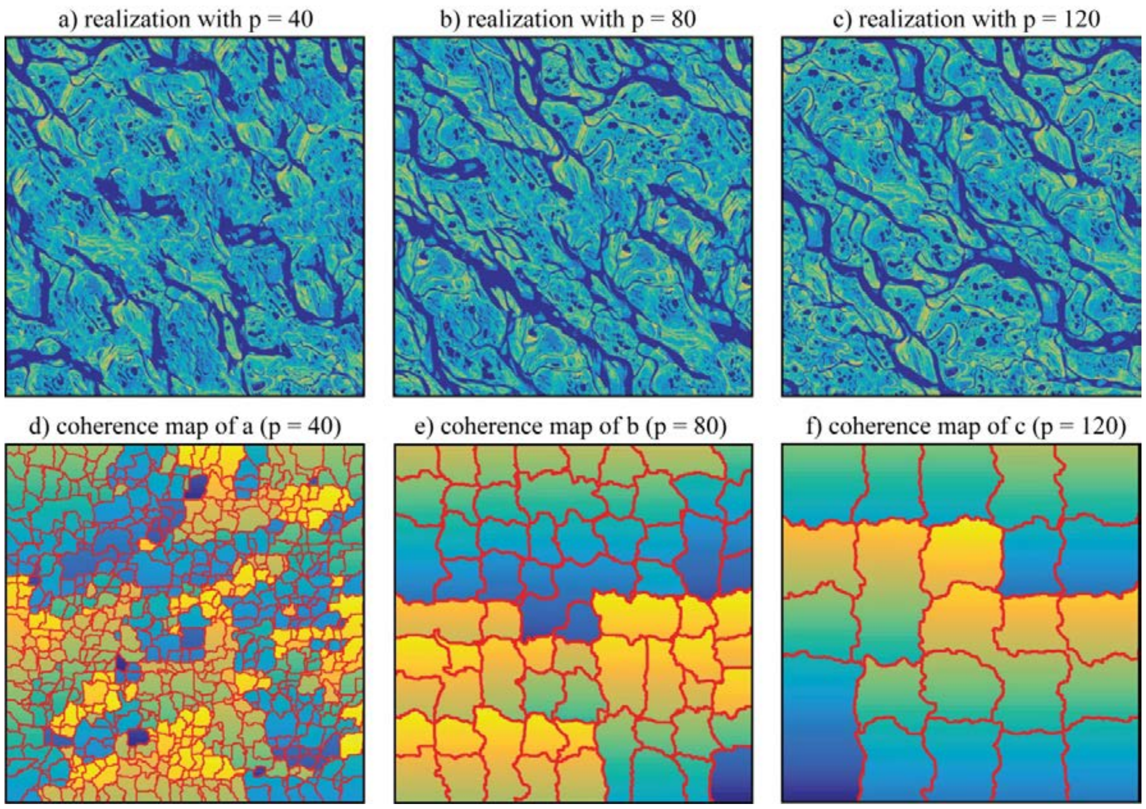


Figure 9.a)-c) Realizations with *o*=25, *ε*=10, and *p*=40, 80, 120 respectively; d)-f) the index coherence maps for a)-c) respectively with the red boundaries showing coherent patches.

Table 3. Average merging index for 10 realizations with *o*=25, *ε*=10 and varying *p*.

Patch size (*p*)	40	80	120
Nb. of cuts	1521	81	25
Average nb. of patches identified from index coherence map	810.7	66.5	21.5
Average merging index (*M*)	0.5327	0.8188	0.8542
Average CPU time (s)	164.36	24.34	11.09

3.1.3. Overlap size

The sensitivity analysis is now carried out with respect to overlap size *o*=5, 25, 45 and keeping fixed values of *p*=80 and *ε*=10. Figure 10 shows corresponding realizations and their index coherence maps. With a very small overlap size, the shapes of the cuts are constrained to be horizontal or vertical as shown in Figure 10d. It decreases the flexibility needed to respect the continuity as shown in Figure 10a with a high merging index (Table 4). A large overlap can result in the adjacent patch in the training image to have an extremely small distance (eq. 5). This can lead to a lower merging index from *o*=5 to *o*=25 as shown in Table 4. While there is only a small difference in merging index between *o*=25 and *o*=45 a larger overlap leads to higher computational cost, which is explained by the graph cuts problem being more difficult to solve as the larger overlap results in a larger graph. In this case the largest variability is obtained with *o*=5, however this variability comes at the price of clearly degraded spatial structures.

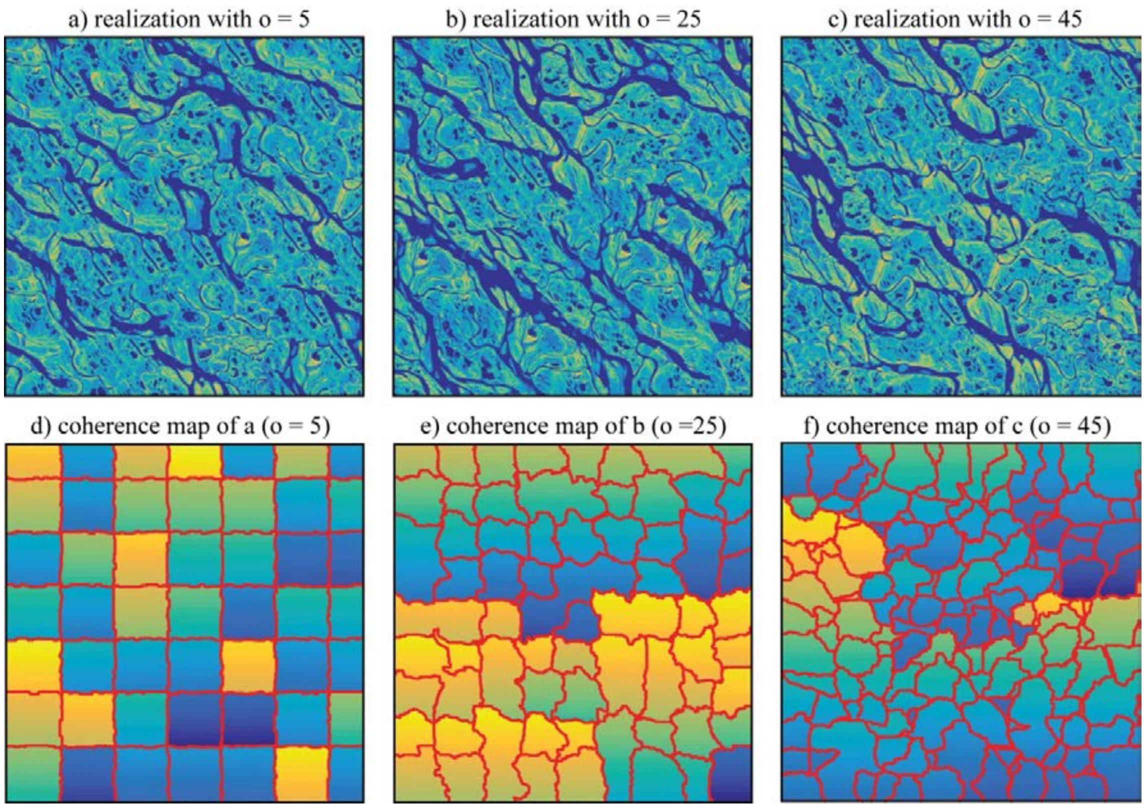


Figure 10.a)-c) Realizations with *p*=80, *ε*=10, and *o*=5, 25, 45 respectively; d)-f) the index coherence maps for a)-c) respectively with the red boundaries showing coherent patches.

Table 4. Average merging index for 10 realizations with *p*=25, *ε*=10 and varying *o*.

Overlap size (*o*)	5	25	45
Nb. of cuts	49	81	169
Average nb. of patches identified from index coherence map	48.1	66.5	139.5
Average merging index (*M*)	0.9813	0.8188	0.8244
Average CPU time (s)	7.13	24.31	79.36

3.2. Conditional simulation

Conditional simulation is tested a 250 by 250 binary training image [*Strebelle*, 2002] shown in Figure 11a. To evaluate the results of the proposed method, we compare them with Direct Sampling (DS) realizations. This pixel-based approach is able to provide accurate conditioning [*Mariethoz et al.*, 2010]. Here we use the following parameters for DS: a distance threshold of 0.05 when comparing data events, a maximum fraction of scanned training image of 0.8 and data events defined as the 40 closest informed neighbors. These parameters were defined through a sensitivity analysis. Figure 11b is an unconditional DS realization from which 100 pixel values are extracted in Figure 10c as conditioning data.

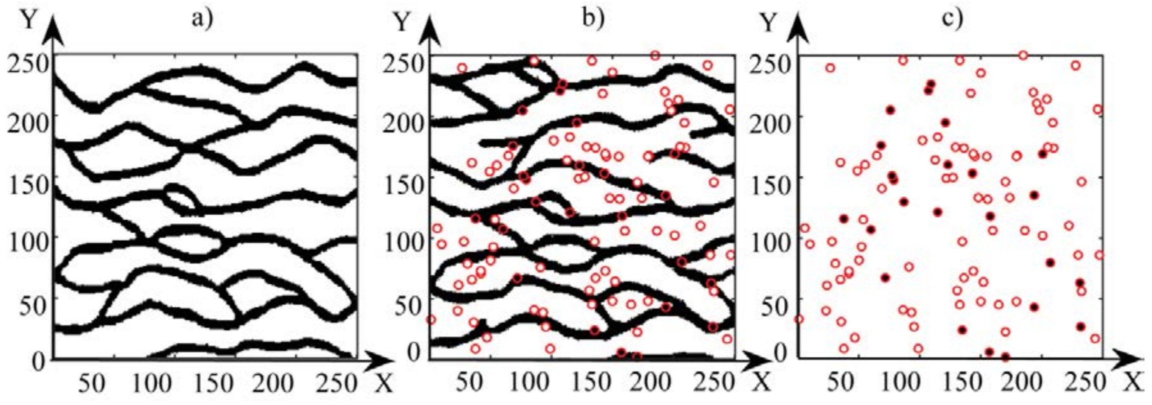


Figure 11. a) Training image; b) unconditional DS realization; c) 100 conditioning data

According to section 3.1.1, the size of the patch should be large enough to capture large-scale structures. Thus p = 40, 60 and 80 is used for conditional simulations (the value of p=80 is too large, but useful to illustrate the iterative conditioning). As expected, it is difficult to satisfy all conditioning constrains with large patches. The number of non-matched conditioning data for each initial realization shown in Figure 12a-c is 1, 3, and 5 respectively which leads to different number of cuts for each conditioning step as shown in Table 5 for *step 2*. Figure 12d-f displays the final conditioned results with stopping criterion $C_{\min}$ =0.01 (the stopping criterion is very small to illustrate the ability of iterative improvement of the proposed method), where all conditioning data are honored.

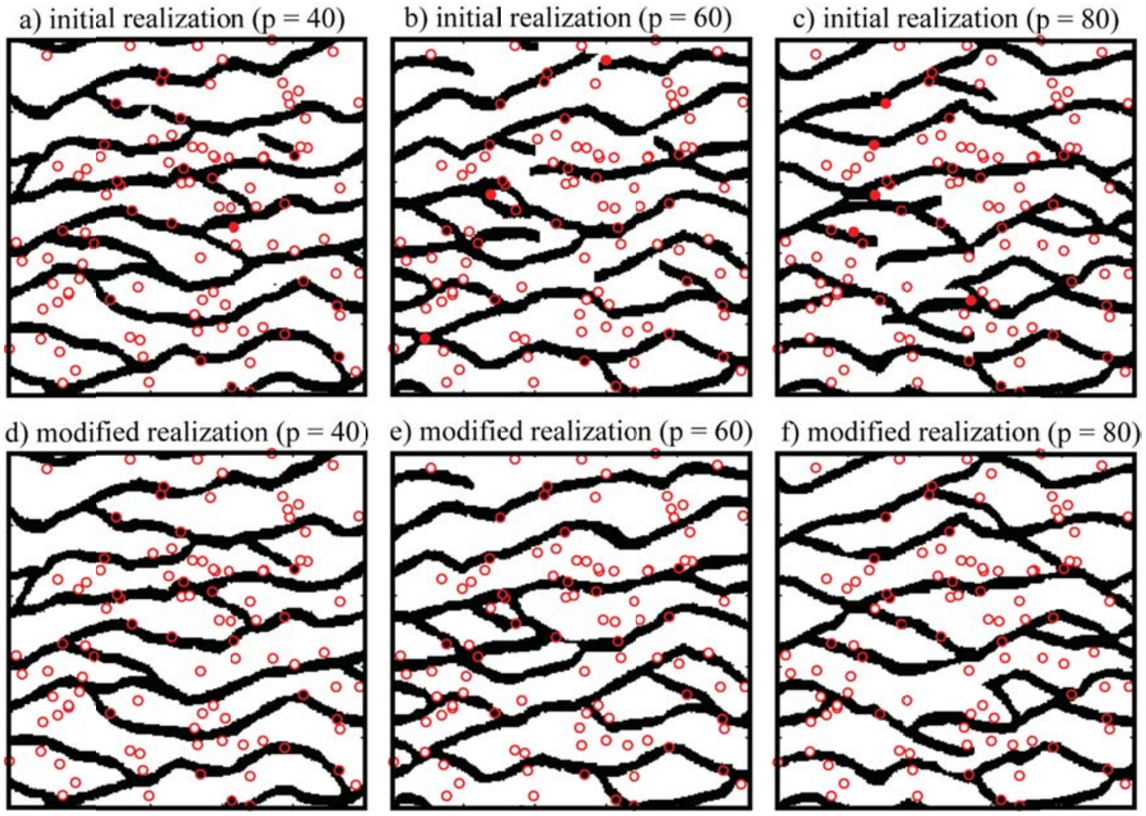

a) initial realization (p = 40)
b) initial realization (p = 60)
c) initial realization (p = 80)
d) modified realization (p = 40)
e) modified realization (p = 60)
f) modified realization (p = 80)

Figure 12. Realizations obtained by the proposed Graph Cuts approach based on conditioning to 100 hard data with *o*=15, *ε* = 10, and *w* = 0.5. a)-c) Initial realizations with *p*=40, 60, and 80 respectively; d)-f) realizations after conditioning and artifacts removal for a)-c) respectively, with a mean cost of 0.01 as stopping criterion. The open circles indicate matched conditioning data and the solid red circles denote non-matched conditioning data.

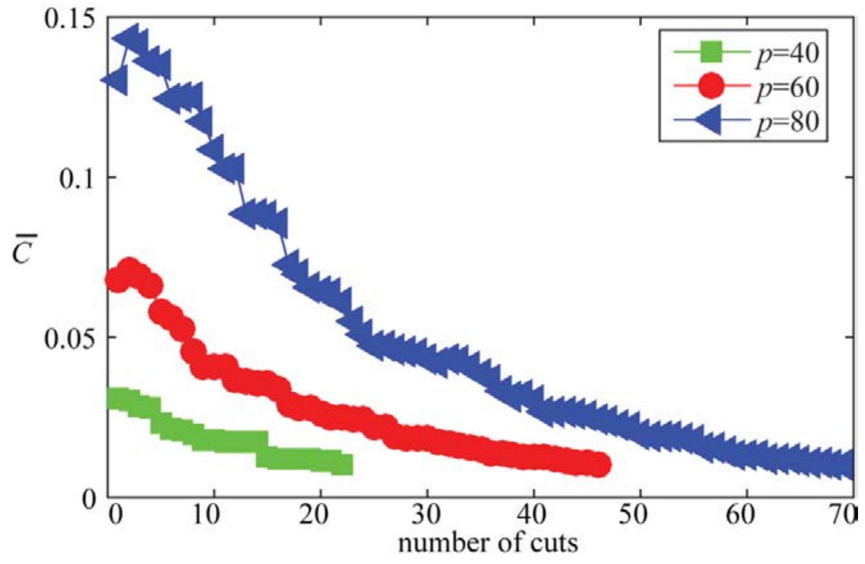


Figure 13. Mean cost of each cut with different patch sizes, as a function of the iteration number.

Table 5.Time for conditional simulation with *o*=15, *ε*= 10, *w* = 0.5, $C_{\min}$ =0.01, and varying patch sizes

patch size (pixel)	40		60		80	
	Nb. of cuts	time (s)	Nb. of cuts	time (s)	Nb. of cuts	time (s)
Step 1: initial realization	100	6.848	36	2.941	16	1.582
Step 2: exact conditioning	1	0.108	3	0.608	5	1.406
Step 3: reducing $\overline{C}$	21	6.022	43	8.737	59	10.266
Total time(s)	12.798		12.286		13.254	

As shown in Figure 13 and in Table 5, $\overline{C}$ increases in *step 1* with increasing patch size and more re-simulation iterations are needed to reduce this cost to the target stopping value. Note that with all parameter values, the final realizations (at the end of step 3) converge to results of similar mean cost, with all conditioning data honored. Although not detailed here, tests showed equally low sensitivity for parameters p, o and ε, thanks to the iterative nature of the algorithm. The convergence of the iterative process for a large range of parameter values is a clear advantage compared to other MPS methods where the results are greatly dependent on a set of parameters that are difficult to choose for non-expert users.

In Figure 14, conditional simulations obtained with Conditional Graph Cuts (with p=40, o=15, ε= 10 and $C_{\min}$=0.01) and DS are compared. Figure 13a-b show the mean of 50 conditional realizations with both methods. It is seen that with DS there are some features that tend to appear in all realizations (continuous black channels in the average map), whereas this effect is less present with Conditional Graph Cuts.

A comparison is also carried out using connectivity functions [*Pardo-Igúzquiza and Dowd*, 2003; *Renard and Allard*, 2013]. The connectivity functions of channels (in black) in the X- and Y-direction are shown in Figure 14c-d. In the X-direction, the connectivity functions of both methods are distributed around that of the training image. In the Y-direction, a similar ensemble of connectivity functions is obtained for the Conditional Graph Cuts realizations, while less connectivity is produced by the Direct Sampling method.

We use multidimensional scaling (MDS) to investigate the variability between realizations obtained with both methods. Given a dissimilarity matrix **D** between model realizations, a MDS representation displays the ensemble of models as a set of points in a possibly high-dimensional Euclidean space, arranged in such a way that their respective distances are preserved [*Scheidt and Caers*, 2009]. **D** can be computed using several appropriate measures of distance; here we use the Hausdorff distance [*Dubuisson and Jain*, 1994] which is an adequate similarity measure for the channels scenario [*Jung et al.*, 2013; *Suzuki and Caers*, 2008]. The coordinates of the points are in high dimension, but for representation they are projected in a 2D space in Figure 14e. There is an overlap of both sets of points, which means a similar ability to reproduce patterns, although none of the sets of models produced are exactly centered on the training image. The points of Conditional Graph Cuts are slightly further from the training image and it has a somewhat wider spread than that of DS (visible in outlier realizations, also identifiable in the connectivity functions). This confirms that Conditional Graph Cuts may generate new patterns from the training image and shows at least as much variability between realizations, despite the DS being pixel-based and Conditional Graph Cuts being patch-based, which usually results in less diversity in the patterns produced [*Mariethoz and Caers*, 2014]. It appears that the iterative nature of the Conditional Graph Cuts algorithm allows generating new patterns (and therefore variability) through the cutting and stitching of new patches. The mean CPU time is 42.37 s for a DS realization and it is 3.54 s for a Conditional Graph Cuts realization with all conditioning data honored and $\overline{C}$=0.067.

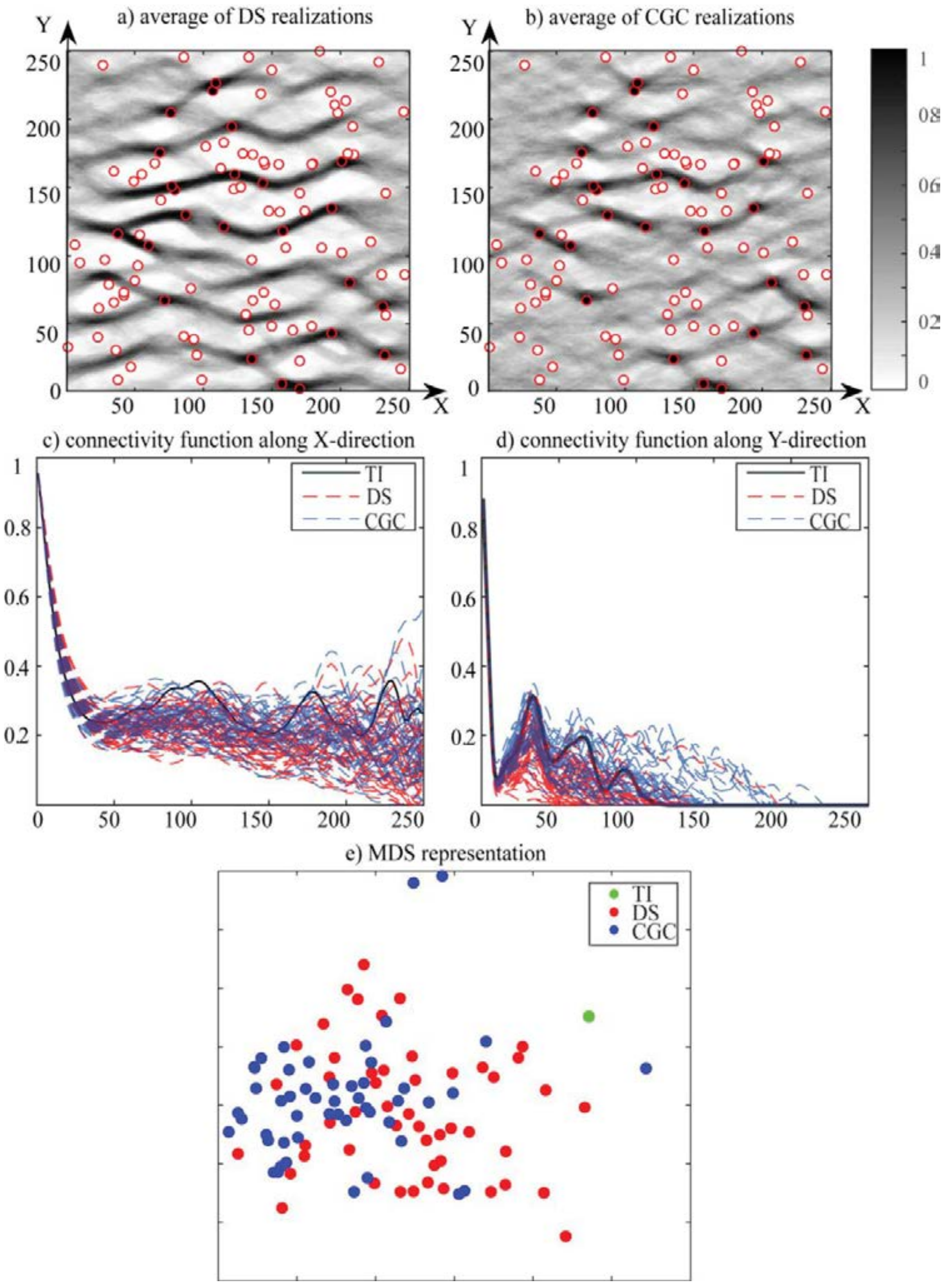


Figure 14. a) Average of 50 DS realizations with threshold=0.05, scanning fraction=0.8 and maximum neighborhood points=40; b) average of 50 conditional graph cuts realizations with *p*=40, *o*=15, ε = 10 and $C_{\min}$ = 0.01; c) connectivity function of channel along X-direction; d) connectivity function along Y-direction; e) MDS representation.

3.3. A continuous test case

In this section we use a continuous training image obtained by imaging flume experiment results [*Paola et al.*, 2009]. The image size is 500 by 200 pixels. We use 50 conditioning data randomly sampled from an unconditional DS realization to create the 2D application shown in Figure 15a and 15b. All the simulations in this section have the same size as the training image. 50 conditional realizations using Conditional Graph Cuts (with $p = 50$, $o = 15$, $\varepsilon = 10$, $w = 0.2$, $l = 200$ and $C_{\min} = 0.15$) and 50 realizations using DS (threshold = 0.04, scanning fraction = 0.8 and maximum neighborhood points = 40) are generated and compared in Figure 15. The two methods produce visually similar realizations. Conditional Graph Cuts introduces more variability between realizations, as shown in the average maps of Figure 15e-f (e.g. top right corner of figure), whereas DS seems to result in some structures that are the same in all realizations. This is also visible in the standard deviation maps of Figure 15g-h. They are related to the occurrence of systematic verbatim copy of the training image. These sharply delimited areas denote specific patterns being systematically laid down in the vicinity of compatible patterns formed by the conditioning data. However such lack of variability is local and in the vicinity of the data. One can identify areas of high variance caused by a low number of possible patches that can be laid at a given location. The well-defined shape of these high-variance zones indicates that they correspond to artifacts rather than a desired uncertainty quantification.

The average simulation time is significantly reduced from 2150 s for DS to 14 s for conditional graph cuts, representing a speedup of 153. Note that the difference in CPU cost between DS and Graph Cuts is much larger than for the channels example in the previous section. This is due to the training image being continuous and complex, which requires the DS to scan a larger fraction of the training image to find an acceptable pattern match [*Meerschman et al.*, 2013]. In comparison, with a categorical training image certain patterns are frequent and can be found after scanning a small portion of the training image. In contrast, Conditional Graph Cuts performs in all cases a full convolution of the training image with the overlap area, whether it is continuous or categorical. Therefore, its computational cost is less sensitive to the type of variable and the complexity of the structures.

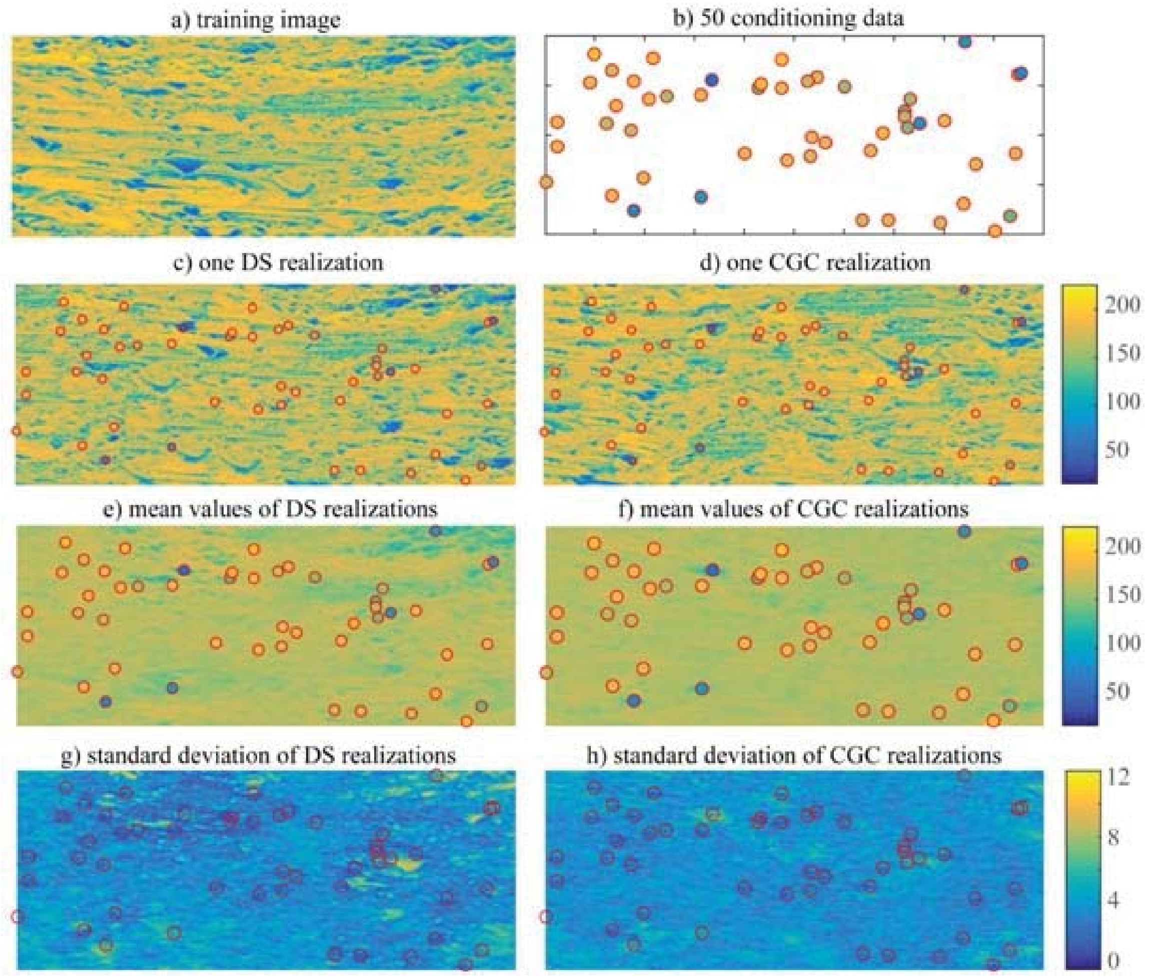


Figure 15. Comparison of 50 DS realizations (with threshold=0.04, scanning fraction=0.8 and maximum neighborhood points= 40) and 50 conditional Graph Cuts realizations (with *p* = 50, *o* = 15, *ε* = 10, *l* = 200 and *w* = 0.2).a) Training image; b) 50 conditioning data; c) one DS realization; d) one Conditional Graph Cuts realization; e) average of 50 DS realizations; f) average of 50 Graph Cuts realizations; g) standard deviation map of 50 DS realizations; h) stand deviation of 50 Graph Cuts realizations.

3.4.3D application

The 3D performance of graph cuts is assessed against CIQ (described in section 1) with respect to CPU efficiency and against both CIQ and DS with respect to patterns variability. CIQ has been thoroughly compared with DS [*Mahmud et al.*, 2014] and presents a speedups of over 50. 3D training images of different sizes are used for this test. The largest training image (Figure 16a) has dimensions of 340 by 200 by 80 and was obtained using the method of *Jha et al.* [2014]. Two smaller training images with size 100 by 100 by 80 and 50 by 50 by 40 are subsets of this large image (Figure 16b and Figure 16c). Three output size, that is 50 by 50 by 40, 100 by 50 by 40, and 100 by 100 by 80 are considered for the tests on each training image.

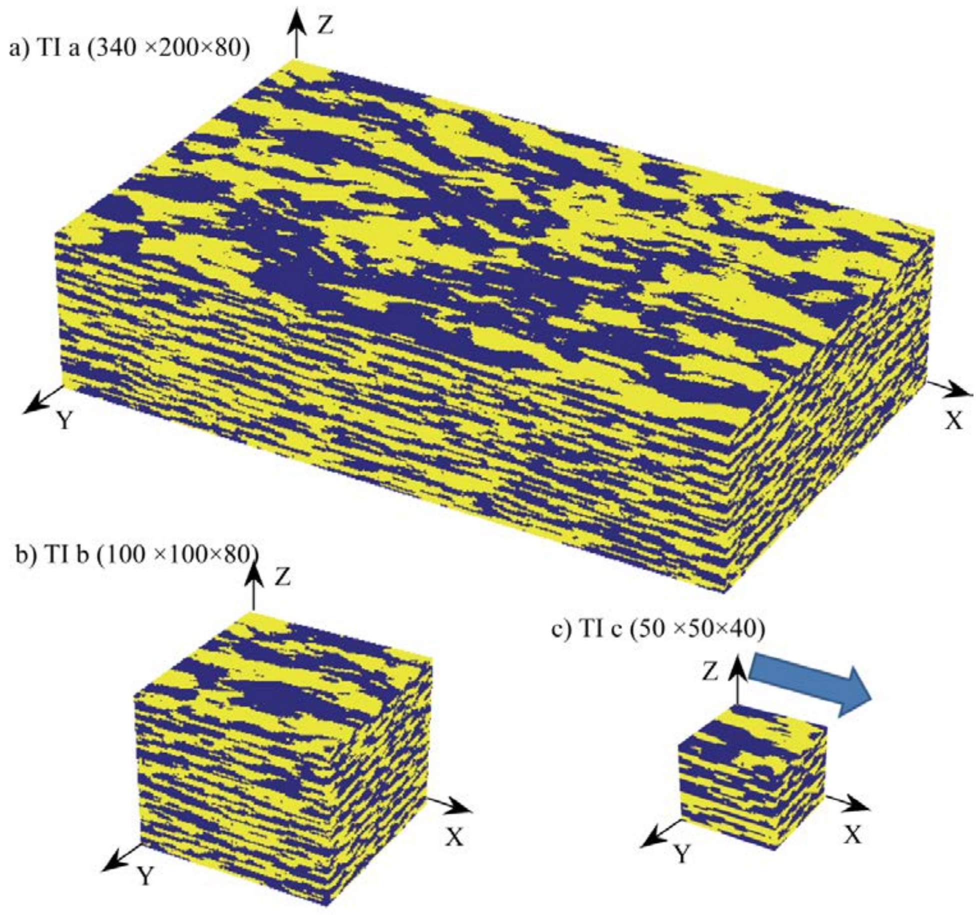


Figure 16. The three training images used. The smaller training images are taken from the large one. The arrow shows the flow direction used for the transport experiment. In all images, voxels have a size of 1m by 1m by 1m.

The same graph cuts parameters are used for both IQ and GC (square patches of size p=30, o=10 in each direction and ε=10). The CPU times for the whole simulation (T_{total}) are displayed in Table 6, along with the time used for scanning the training images (T_{scan}), also given as a percentage $P_T = T_{\text{scan}}/T_{\text{total}}$*100%. We do not use DS in the CPU performance comparison as it is much slower than both GC and IQ.

The results in Table 6 show that the scan of the training image takes most of the time for both methods, and that this scanning time increases with the size of the training image. Note that the computer function used to scan the training image is identical in IQ and GC. In comparison, the percentage of scanning time of GC decreases for larger output size since it requires more time to perform the cuts and to record the seam vertices. However, despite of the additional CPU time GC is globally faster than IQ. The reason is that IQ splits the overlaps into several smaller cuboids, each of which requires a scan of the training image. Conversely, GC can process patches of arbitrary shape, hence reducing the number of training image scans required (N_{scan}), as shown in Table 6. The increasing of output size leads to an increasing percentage of non-cubic overlaps, as a results, the requirements of scanning number of scanning time of IQ increase faster than that of GC.

Table 6. CPU performance for Image Quilting and Graph Cuts (time in seconds per realization). In order to generalize the assessment of CPU time, different training image sizes are considered, as well as different simulation sizes for each training image.

	Output size	IQ			GC			speedup
		T_{total}	T_{scan}	N_{scan}	T_{total}	T_{scan}	N_{scan}	

TI a	50 by 50 by 40	522	517	19	300	284	7	1.74
		P_T = 99.07%			P_T = 94.72%			
	100 by 50 by 40	1605	1595	61	850	799	19	1.89
		P_T = 99.41%			P_T = 93.93%			
	100 by 100 by 80	11172	11126	467	5272	4863	99	2.12
		P_T = 99.59%			P_T = 92.26%			
TI b	50 by 50 by 40	150	146	19	73	56	7	2.05
		P_T = 97.74%			P_T = 76.04%			
	100 by 50 by 40	452	447	61	204	153	19	2.22
		P_T = 98.75%			P_T = 75.07%			
	100 by 100 by 80	3145	3123	467	1311	926	99	2.40
		P_T = 99.28%			P_T = 70.58%			
TI c	50 by 50 by 40	29	28	19	15	8	7	1.93
		P_T = 96.49%			P_T = 51.67%			
	100 by 50 by 40	90	87	61	51	21	19	1.76
		P_T = 96.88%			P_T = 40.53%			
	100 by 100 by 80	632	614	467	409	112	99	1.55
		P_T = 97.22%			P_T = 27.40%			

Figures 17 shows individual realizations obtained with the different methods, as well as comparative metrics. In Figure 17e it is visible that the experimental variograms do not allow distinguishing between the methods tested. To test the reproduction of connectivity patterns, we use flow and transport modeling on the realizations. With TI c (Figure 16c), 30 realizations of the same size as the training image are generated with all methods (DS parameters: threshold=0.04, scanned fraction = 0.3 and 25 closest neighbors). The facies models are converted to hydraulic parameters by assigning to facies 0 and 1 conductivity values of 10^{-8} m/s and 10^{-3} m/s, and porosity values of 0.01 and 0.1, respectively. A dynamic model is set with a steady-state flow from left (X = 0) to right (X = 50) as shown by a blue arrow in Figure 16c. The head boundary conditions are set as H=1 on the left and H=0 on the right side of the domain (gradient of 0.02). With a conservative tracer, a transient transport regime is defined by setting up an initial concentration of 0 on the entire domain and a concentration of 1 on the left boundary, meaning that the contaminant is progressively pushed into the model. This transient setting is run for 250 hours using a groundwater finite element code [*Cornaton and Perrochet*, 2006], and the mass flux on the right side of the domain is recorded, corresponding to the contaminant breakthrough curve averaged over the outflowing boundary (Figure 17f). With this setting, a rapid breakthrough corresponds to a high degree of connectivity of the channelized structures. The results are shown in Figure 17. While all methods are globally similar in terms of transport, in this case the GC realizations are slightly more connected than those of the other methods, which is indicated by faster breakthrough. The iterative modification step of the GC results in retaining long-range patterns, which translates into a better reproduction of the connectivity properties,

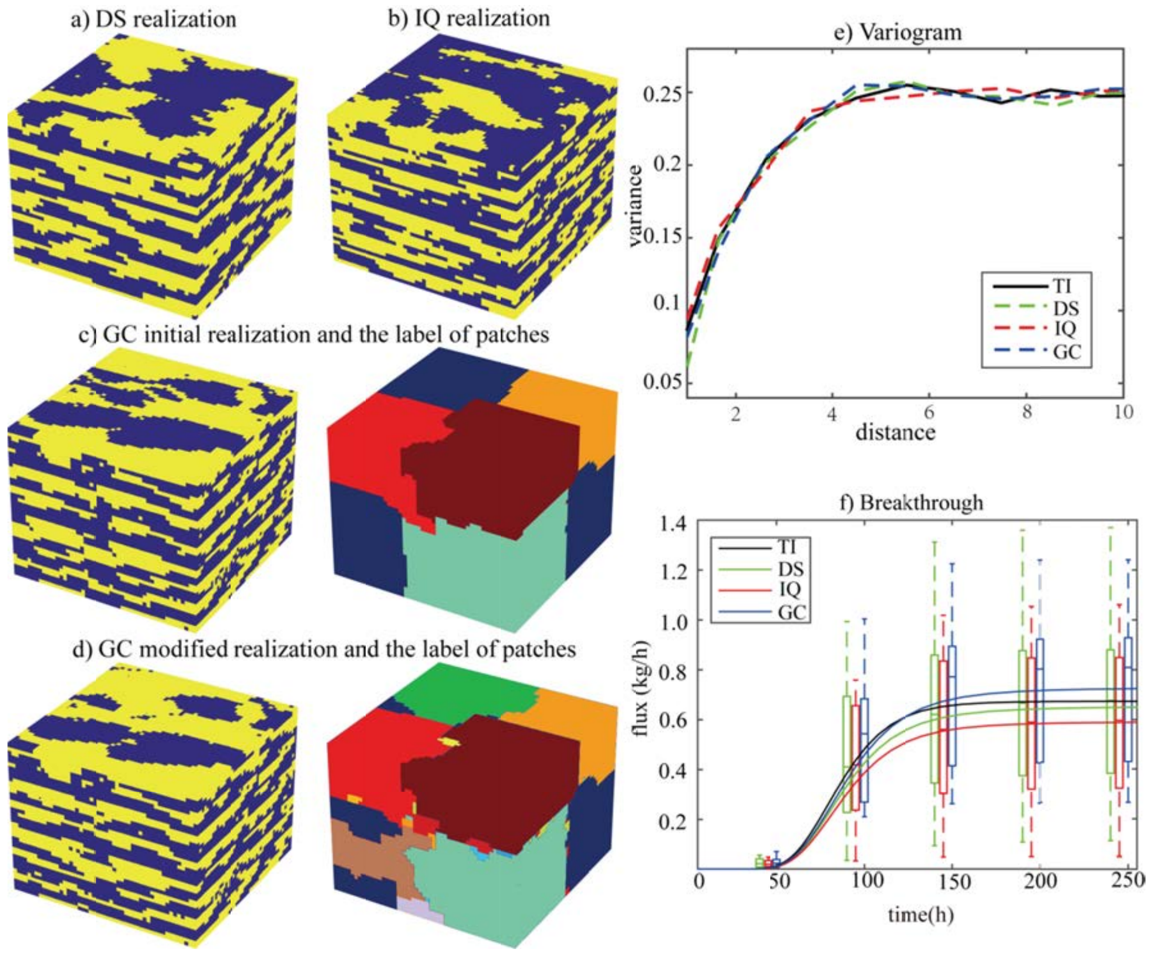


Figure 17. Comparison of 3D unconditional realizations and corresponding variograms and contaminant breakthrough curves. a) One DS realization; b) one IQ realization; c) one initial GC realization and the patch labels; d) One modified GC realization after *l*=50 iterations and the patch labels. e) ensemble variograms of 30 realizations generated by each method; f) ensemble breakthrough curves.

4. Discussion and conclusion

In this paper, a graph cut-based algorithm is presented for multiple-point geostatistical simulation with point conditioning. This algorithm is an adaptation of an efficient graph cut technique used in computer graphics and initially proposed by *Kwatra et al.* [2003], which accounts for the information carried by previously placed patches. It is based on the generation of cuts in patches of arbitrary size and shape, which are optimal in terms of minimizing an overlap error.

For conditioning, we design a sequential simulation strategy that takes place in several steps: an initial simulation step followed by a modification step that involves several iterations. Acknowledging that an initial realization can present artifacts, the iterations are used to successively remove such artifacts. The iterative modification can also be used to achieve exact conditioning, which is usually a challenge for patch-based MPS. It tracks the mismatch to conditioning data and uses additional patches at locations where there is residual error. An extension of the terminals is proposed to preserve local conditioning. In some of the test cases, the initial simulation is already satisfying in terms of both data conditioning and absence of artifacts, and then few if any iterative patches replacements are required. While it is not investigated in this paper, it may be possible to accommodate for uncertain conditioning data by adjusting the stopping criterion for iterative conditioning, for example allowing for a tolerance proportional to the measurement error.

While graph cuts have been used in computer graphics to obtain visually appealing textures, it remained to be seen whether this could translate into a satisfying level of variability of the models generated. Our tests showed that it is the case, and that graph cuts can be used to design a patch-based simulation algorithm requiring few parameters that only needs a limited tuning to achieve good results. The iterative re-simulation can improve a poor initial simulation caused by suboptimal parameterization. Moreover, the proposed algorithm provides at least as much variability as pixel-based methods because the patch cutting procedure generates new patterns. This is confirmed by 3D flow and transport modeling applied to the generated fields.

The question of patterns diversity is important because it is often difficult to find or to build training images that are large enough to represent an appropriate diversity of connected patterns. Our tests showed that graph cuts generally provides increased patterns diversity compared to other MPS simulation methods, as well as improved preservation of the long-range connectivity, which are two essential characteristics when modeling flow and transport in heterogeneous media. This is explained by the graph cuts continuously generating new patterns that are not present in the training image. This increases the patterns variability and improves the chances of obtaining structures that honor the connectivity present in the training image, even if patterns slightly different than those of the training image have to be generated through the iterative cut process. At the same time, the iterative generation of patterns through cuts ensures that verbatim copy does not occur.

Despite the method being iterative, the use of patches and the efficient implementation of the max-flow/min-cut algorithm allow for a substantial improvement in computational cost compared to Direct Sampling (10-150 times). This speedup is especially pronounced when simulating continuous variables. Perhaps more surprising is the fact that Graph Cuts is about 2 times more computationally efficient than IQ, even though the scan of the training image is done in the same manner. It is found that the difference in CPU time is due to smaller number of scans of the training image needed because graph cuts can accommodate arbitrary shape cuts and only requires a single cut for complex overlap. IQ on the other hand requires several smaller cuts for each overlap, involving additional scanning of the training image, which is the most time consuming part of the algorithm.

The proposed method can be further improved. For example, the search for candidate patches by convolution is time consuming, especially for 3D cases. This step could be further accelerated by using strategies such as Fourier-based decomposition of the training image [*Tahmasebi et al.*, 2014], a parallel search method, or by convolving only a part of the training image. The incorporation of auxiliary variables is also a natural continuation of this research, with ongoing work aimed at a multivariate implementation of Conditional Graph Cuts.

Acknowledgements

The China Scholarship Council is acknowledged for funding this work. A computer code is available on the website of the second author (http://www.minds.ch/gm/downloads.htm, then use the link for the graph-cuts based simulation).

5. References

Allard, D., R. Froidevaux, and P. Biver (2006), Conditional simulation of multi-type non stationary Markov object models respecting specified proportions, *Mathematical Geology*, *38*(8), 959-986.

Arpat, G. B., and J. Caers (2007), Conditional simulation with patterns, *Mathematical Geology*, *39*(2), 177-203.

Boykov, Y., and V. Kolmogorov (2004), An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *26*(9), 1124-1137.

Caers, J. (2011), *Modeling uncertainty in the earth sciences*, John Wiley & Sons.

Chatterjee, S., and R. Dimitrakopoulos (2012), Multi-scale stochastic simulation with a wavelet-based approach, *Computers and Geosciences*, *45*, 177-189.

Chatterjee, S., R. Dimitrakopoulos, and H. Mustapha (2012), Dimensional reduction of pattern-based simulation using wavelet analysis, *Mathematical Geosciences*, *44*(3), 343-374.

Cornaton, F., and P. Perrochet (2006), Groundwater age, life expectancy and transit time distributions in advective–dispersive systems: 1. Generalized reservoir theory, *Advances in Water Resources*, *29*(9), 1267-1291.

Daly, C. (2004), Higher order models using entropy, Markov random fields and sequential simulation, paper presented at Geostatistics Banff 2004, Kluwer Academic Publisher, Banff, Alberta, 215-224.

De Marsily, G., F. Delay, J. Gonçalvès, P. Renard, V. Teles, and S. Violette (2005), Dealing with spatial heterogeneity, *Hydrogeology Journal*, *13*(1), 161-183.

Deutsch, C., and A. Journel (1992), *GSLIB: Geostatistical Software Library*, 340 pp., Oxford Univ. Press, New York.

Deutsch, C., and T. Tran (2002), FLUVSIM: a program for object-based stochastic modeling of fluvial depositional systems, *Comp. & Geosci.*, *28*(4), 525-535.

Deutsch, C. V., and L. Wang (1996), Hierarchical object-based stochastic modeling of fluvial reservoirs, *Mathematical Geology*, *28*(7), 857-880.

Dubuisson, M., and A. Jain (1994), A Modified Hausdorff Distance for Object Matching, in *International Conference on Pattern Recognition*, edited, pp. 566-568, Jerusalem, Isarel.

Efros, A., and T. K. Leung (1999), Texture synthesis by non-parametric sampling, paper presented at Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, IEEE, 1033-1038.

Efros, A. A., and W. T. Freeman (2001), Image quilting for texture synthesis and transfer, paper presented at Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM, 341-346.

Emery, X., and C. Lantuéjoul (2014), Can a Training Image Be a Substitute for a Random Field Model?, *Math. Geosci.*, *46*(2), 133-147.

Eskandari, K., and S. Srinivasan (2010), Reservoir Modelling of Complex Geological Systems--A Multiple-Point Perspective, *Journal of Canadian Petroleum Technology*, *49*(08), 59-69.

Ford, L. R., and D. R. Fulkerson (1956), Maximal flow through a network, *Canadian journal of Mathematics*, *8*(3), 399-404.

Goldberg, A. V., and R. E. Tarjan (1988), A new approach to the maximum-flow problem, *Journal of the ACM (JACM)*, *35*(4), 921-940.

Goovaerts, P. (1998), Geostatistical tools for characterizing the spatial variability of microbiological and physico-chemical soil properties, *Biology and Fertility of soils*, *27*(4), 315-334.

Guardiano, F. B., and R. M. Srivastava (1993), Multivariate geostatistics: beyond bivariate moments, in *Geostatistics Troia'92*, edited, pp. 133-144, Springer.

Hermans, T., A. Vandenbohede, L. Lebbe, R. Martin, A. Kemna, J. Beaujean, and F. Nguyen (2012), Imaging artificial salt water infiltration using electrical resistivity tomography constrained by geostatistical data, *Journal of Hydrology*, *438*, 168-180.

Hermans, T., F. Nguyen, and J. Caers (2015), Uncertainty in training image-based inversion of hydraulic head data constrained to ERT data: Workflow and case study, *Water resources research*, *51*(7), 5332-5352.

Honarkhah, M., and J. Caers (2010), Stochastic simulation of patterns using distance-based pattern modeling, *Math. Geosci.*, *42*(5), 487-517.

Hu, L., and T. Chugunova (2008), Multiple-Point Geostatistics for Modeling Subsurface Heterogeneity: a Comprehensive Review, *Water Resour. Res.*, *44*(W11413).

Huang, T., X. Li, T. Zhang, and D. T. Lu (2013a), GPU-accelerated Direct Sampling method for multiple-point statistical simulation, *Computers & Geosciences*, *57*, 13-23.

Huang, T., D.-T. Lu, X. Li, and L. Wang (2013b), GPU-based SNESIM implementation for multiple-point statistical simulation, *Computers & Geosciences*, *54*(0), 75-87.

Huysmans, M., P. Orban, E. Cochet, M. Possemiers, B. Ronchi, K. Lauriks, O. Batelaan, and A. Dassargues (2013), Using Multiple-Point Geostatistics for Tracer Test Modeling in a Clay-Drape Environment with Spatially Variable Conductivity and Sorption Coefficient, *Math. Geosci.*, 1-19.

Isaaks, E. H., and R. M. Srivastava (1989), *Applied geostatistics*, Oxford University Press New York.

Jha, S. K., A. Comunian, G. Mariethoz, and B. F. J. Kelly (2014), Parameterization of training images for aquifer 3-D facies modeling integrating geological interpretations and statistical inference, *Water Resour. Res.*, *50*(10), 7731-7749.

Journel, A. G. (1993), Geostatistics: roadblocks and challenges, in *Geostatistics Troia'92*, edited, pp. 213-224, Springer.

Journel, A. G. (2005), Beyond covariance: the advent of multiple-point geostatistics, in *Geostatistics Banff 2004*, edited, pp. 225-233, Springer.

Jung, A., D. H. Fenwick, and J. Caers (2013), Training image-based scenario modeling of fractured reservoirs for flow uncertainty quantification, *Computational Geosciences*, *17*(6), 1015-1031.

Klise, K. A., G. S. Weissmann, S. A. McKenna, E. M. Nichols, J. D. Frechette, T. F. Wawrzyniec, and V. C. Tidwell (2009), Exploring solute transport and streamline connectivity using lidar□based outcrop images and geostatistical representations of heterogeneity, *Water resources research*, *45*(5).

Koltermann, C., and S. Gorelick (1996), Heterogeneity in sedimentary deposits: A review of structure-imitating, process-imitating, and descriptive approaches, *Water Resour. Res.*, *32*(9), 2617-2658.

Krishnan, S., and A. Journel (2003), Spatial connectivity: from variograms to multiple-point measures, *Mathematical Geology*, *35*(8), 915-925.

Kwatra, V., A. Schödl, I. Essa, G. Turk, and A. Bobick (2003), Graphcut textures: image and video synthesis using graph cuts, paper presented at ACM Transactions on Graphics (ToG), ACM, 277-286.

Laloy, E., N. Linde, D. Jacques, and G. Mariethoz (2016), Merging parallel tempering with sequential geostatistical resampling for improved posterior exploration of high-dimensional subsurface categorical fields, *Adv. Water Resour.*, *90*, 57-69.

Lasram, A., S. Lefebvre, and C. Damez (2012), Scented sliders for procedural textures, paper presented at EUROGRAPHICS short papers,

Lee, J., and P. K. Kitanidis (2014), Large-scale hydraulic tomography and joint inversion of head and tracer data using the Principal Component Geostatistical Approach (PCGA), *Water Resour. Res.*, *50*(7), 5410-5427.

Li, X., and G. Mariethoz (2015), Stochastic modelling of patterns using graph cuts, paper presented at Petroleum Geostatistics 2015, 278-282.

Lochbühler, T., G. Pirot, J. Straubhaar, and N. Linde (2014), Conditioning of Multiple-Point Statistics Facies Simulations to Tomographic Images, *Math. Geosci.*, *46*(5), 625-645.

Mahmud, K., G. Mariethoz, J. Caers, P. Tahmasebi, and A. Baker (2014), Simulation of Earth textures by conditional image quilting, *Water resources research*, *50*(4), 3088-3107.

Mahmud, K., G. Mariethoz, A. Baker, and A. Sharma (2015), Integrating multiple scales of hydraulic conductivity measurements in training image-based stochastic models, *Water Resour. Res.*, *51*(1), 465-480.

Mariethoz, G. (2010), A general parallelization strategy for random path based geostatistical simulation methods, *Computers & Geosciences*, *36*(7), 953-958.

Mariethoz, G., P. Renard, and J. Straubhaar (2010), The Direct Sampling method to perform multiple□point geostatistical simulations, *Water resources research*, *46*(11).

Mariethoz, G., and J. Caers (2014), *Multiple-point Geostatistics: Stochastic Modeling with Training Images*, John Wiley & Sons.

Mariethoz, G., and S. Lefebvre (2014), Bridges between multiple-point geostatistics and texture synthesis: Review and guidelines for future research, *Computers and Geosciences*, *66*, 66-80.

Meerschman, E., G. Pirot, G. Mariethoz, J. Straubhaar, M. Van Meirvenne, and P. Renard (2013), A practical guide to performing multiple-point statistical simulations with the Direct Sampling algorithm, *Computers and Geosciences*, *52*, 307-324.

Michael, H., A. Boucher, T. Sun, J. Caers, and S. Gorelick (2010), Combining geologic-process models and geostatistics for conditional simulation of 3-D subsurface heterogeneity, *Water Resour. Res.*, *46*(W05527).

Mustapha, H., and R. Dimitrakopoulos (2011), HOSIM: A high-order stochastic simulation algorithm for generating three-dimensional complex geological patterns, *Computers & Geosciences*, *37*(9), 1242-1253.

Paola, C., K. Straub, D. Mohrig, and L. Reinhardt (2009), The "unreasonable effectiveness" of stratigraphic and geomorphic experiments, *Earth-Science Reviews*, *97*(1), 1-43.

Pardo-Igúzquiza, E., and P. Dowd (2003), CONNEC3D: a computer program for connectivity analysis of 3D random set models, *Comp. & Geosci.*, *29*, 775-785.

Parra, Á., and J. M. Ortiz (2011), Adapting a texture synthesis algorithm for conditional multiple point geostatistical simulation, *Stochastic Environmental Research and Risk Assessment*, *25*(8), 1101-1111.

Pérez, C., G. Mariethoz, and J. M. Ortiz (2014), Verifying the high-order consistency of training images with data for multiple-point geostatistics, *Computers and Geosciences*, *70*, 190-205.

Renard, P., and D. Allard (2013), Connectivity metrics for subsurface flow and transport, *Adv. Water Resour.*, *51*, 168-196.

Rezaee, H., G. Mariethoz, M. Koneshloo, and O. Asghari (2013), Multiple-point geostatistical simulation using the bunch-pasting direct sampling method, *Computers and Geosciences*, *54*, 293-308.

Saibaba, A. K., and P. K. Kitanidis (2015), Fast computation of uncertainty quantification measures in the geostatistical approach to solve inverse problems, *Adv. Water Resour.*, *82*, 124-138.

Scheidt, C., and J. Caers (2009), Representing spatial uncertainty using distances and kernels, *Math. Geosci.*, *41*(4), 397-419.

Skorstad, A., R. Hauge, and L. Holden (1999), Well conditioning in a fluvial reservoir model, *Mathematical Geology*, *31*(7), 857-872.

Straubhaar, J., P. Renard, G. Mariethoz, R. Froidevaux, and O. Besson (2011), An improved parallel multiple-point algorithm using a list approach, *Mathematical Geosciences*, *43*(3), 305-328.

Straubhaar, J., A. Walgenwitz, and P. Renard (2013), Parallel multiple-point statistics algorithm based on list and tree structures, *Mathematical Geosciences*, *45*(2), 131-147.

Strebelle, S. (2002), Conditional simulation of complex geological structures using multiple-point statistics, *Mathematical Geology*, *34*(1), 1-21.

Strebelle, S. (2003), New multiple-point statistics simulation implementation to reduce memory and cpu-time demand, paper presented at Conference of the international association for mathematical geology. Portsmouth, UK, September,

Suzuki, S., and J. Caers (2008), A distance-based prior model parameterization for constraining solutions of spatial inverse problems, *Math. Geosci.*, *40*(4), 445-469.

Tahmasebi, P., A. Hezarkhani, and M. Sahimi (2012a), Multiple-point geostatistical modeling based on the cross-correlation functions, *Computational Geosciences*, *16*(3), 779-797.

Tahmasebi, P., M. Sahimi, G. Mariethoz, and A. Hezarkhani (2012b), Accelerating geostatistical simulations using graphics processing units (GPU), *Computers and Geosciences*, *46*, 51-59.

Tahmasebi, P., M. Sahimi, and J. Caers (2014), MS-CCSIM: accelerating pattern-based geostatistical simulation of categorical variables using a multi-scale search in Fourier space, *Computers & Geosciences*, *67*, 75-88.

Tahmasebi, P., and M. Sahimi (2016a), Enhancing multiple-point geostatistical modeling: 1. Graph theory and pattern adjustment, *Water Resour. Res.*

Tahmasebi, P., and M. Sahimi (2016b), Enhancing multiple-point geostatistical modeling: 2. Iterative simulation and multiple distance function, *Water Resour. Res.*

Tan, X., P. Tahmasebi, and J. Caers (2014), Comparing Training-Image Based Algorithms Using an Analysis of Distance, *Math. Geosci.*, *46*(2), 149-169.

Walsh, S. D., M. O. Saar, P. Bailey, and D. J. Lilja (2009), Accelerating geoscience and engineering system simulations on graphics hardware, *Computers & Geosciences*, *35*(12), 2353-2364.

Wei, L., and M. Levoy (2000), Fast Texture Synthesis using Tree-structured Vector Quantization, paper presented at SIGGRAPH '00: 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley, New Orleans, Zahner, T., T. Lochbühler, G. Mariethoz, and N. Linde (2016), Image Synthesis with Graph Cuts: A Fast Model Proposal Mechanism in Probabilistic Inversion, *Geophysical Journal International*, *204*, 1179-1190.

Zhang, T., P. Switzer, and A. Journel (2006), Filter-based classification of training image patterns for spatial simulation, *Mathematical Geology*, *38*(1), 63-80.

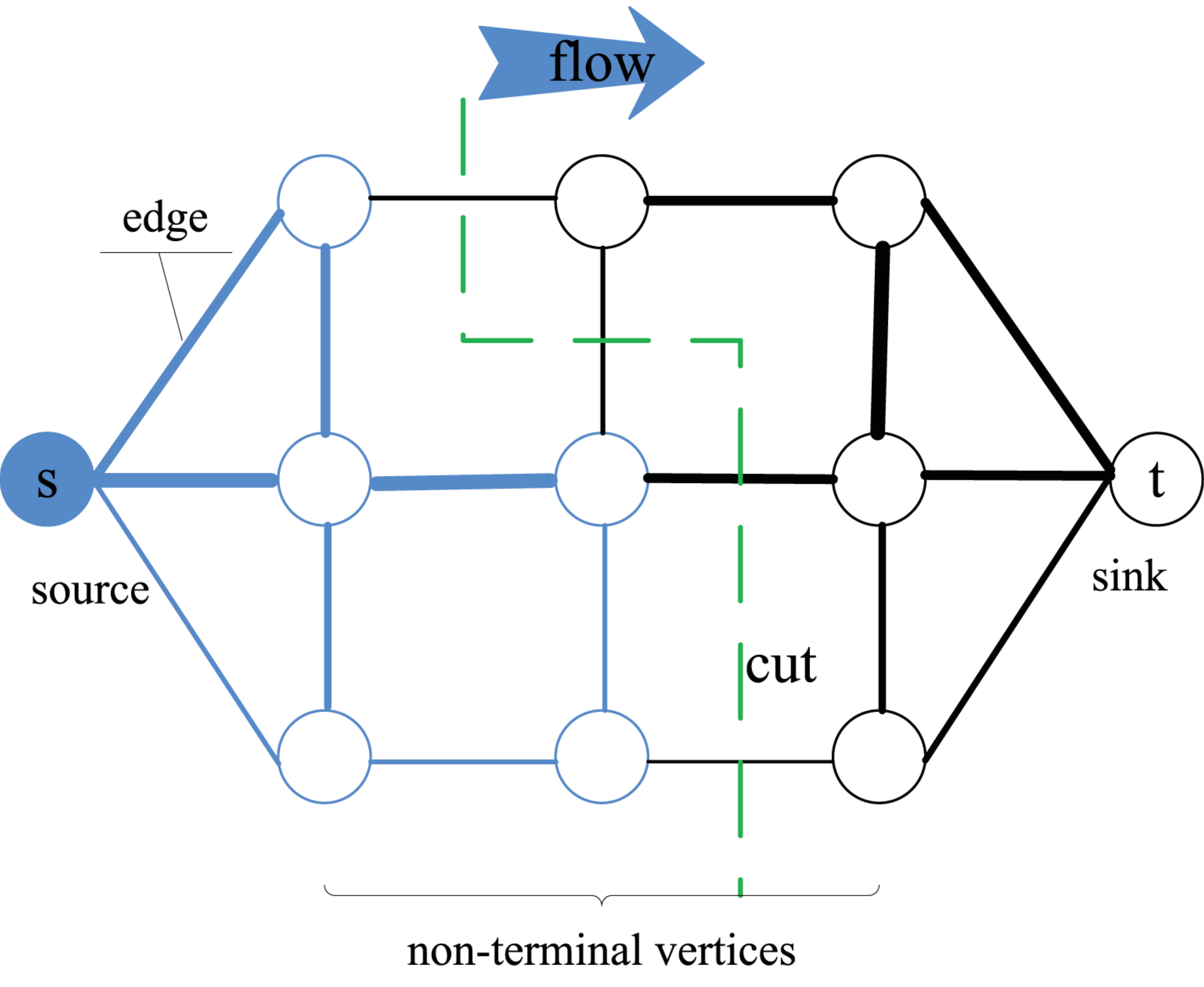

flow
edge
s
source
t
sink
cut
non-terminal vertices

Figure 2. Figure

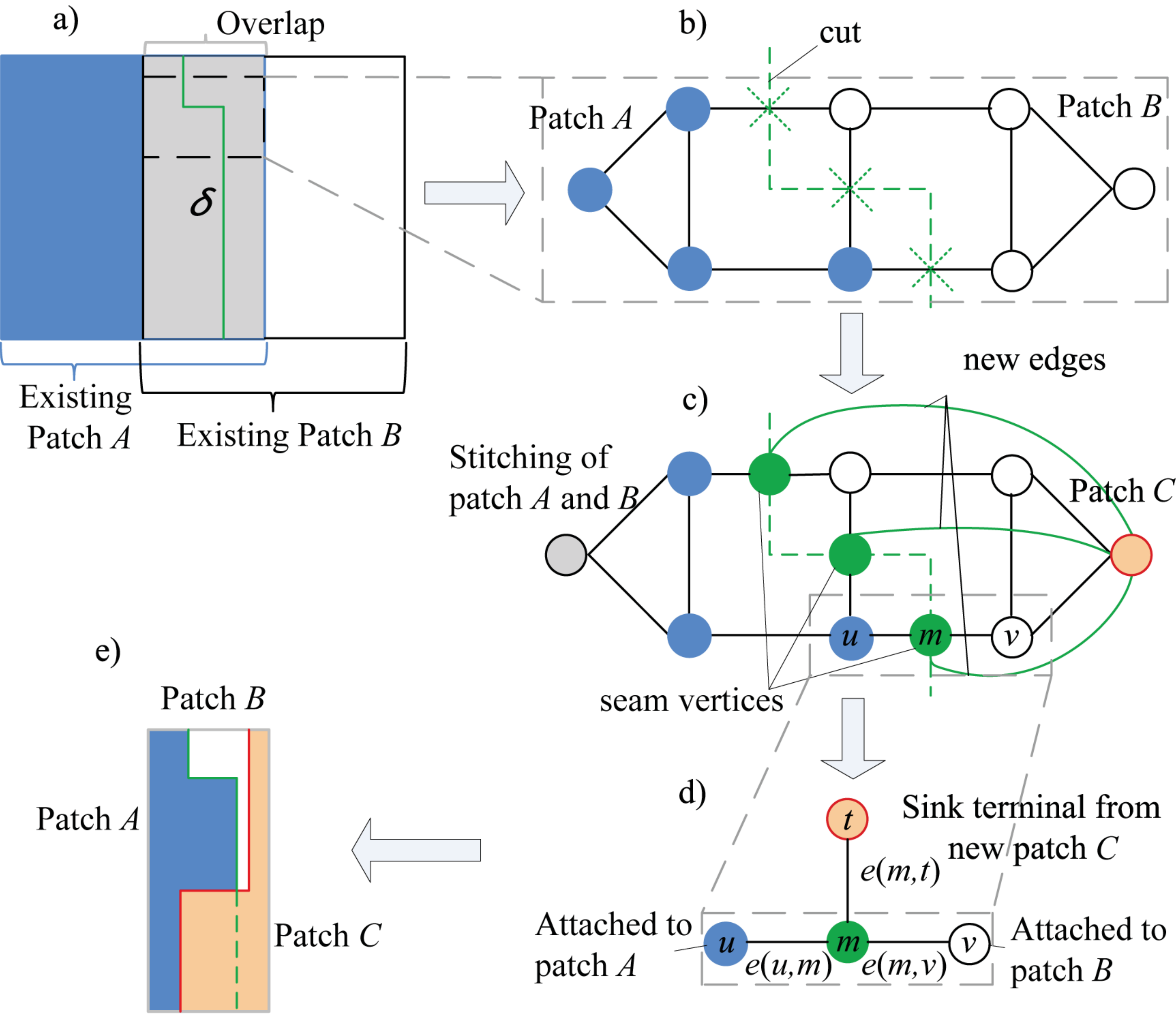
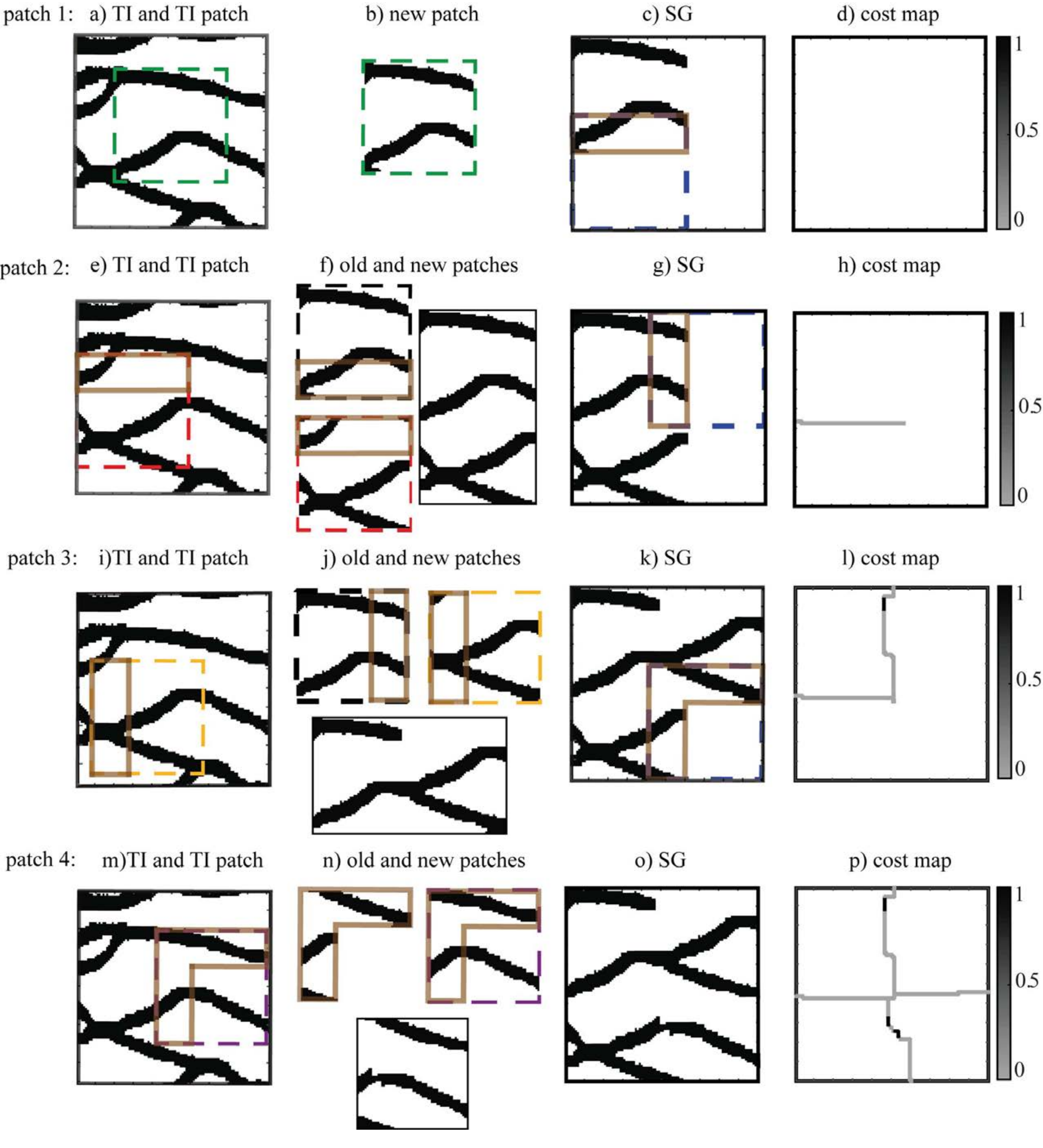

a)
Overlap
δ
Existing Patch A
Existing Patch B
b)
cut
Patch A
Patch B
c)
new edges
Stitching of patch A and B
Patch C
u
m
v
seam vertices
d)
t
Sink terminal from new patch C
e(m,t)
Attached to patch A
u
m
v
e(u,m)
e(m,v)
Attached to patch B
e)
Patch B
Patch A
Patch C

Figure 3. Figure



patch 1:
a) TI and TI patch
b) new patch
c) SG
d) cost map
1
0.5
0
patch 2:
e) TI and TI patch
f) old and new patches
g) SG
h) cost map
1
0.5
0
patch 3:
i)TI and TI patch
j) old and new patches
k) SG
l) cost map
1
0.5
0
patch 4:
m)TI and TI patch
n) old and new patches
o) SG
p) cost map
1
0.5
0

Figure 4. Figure

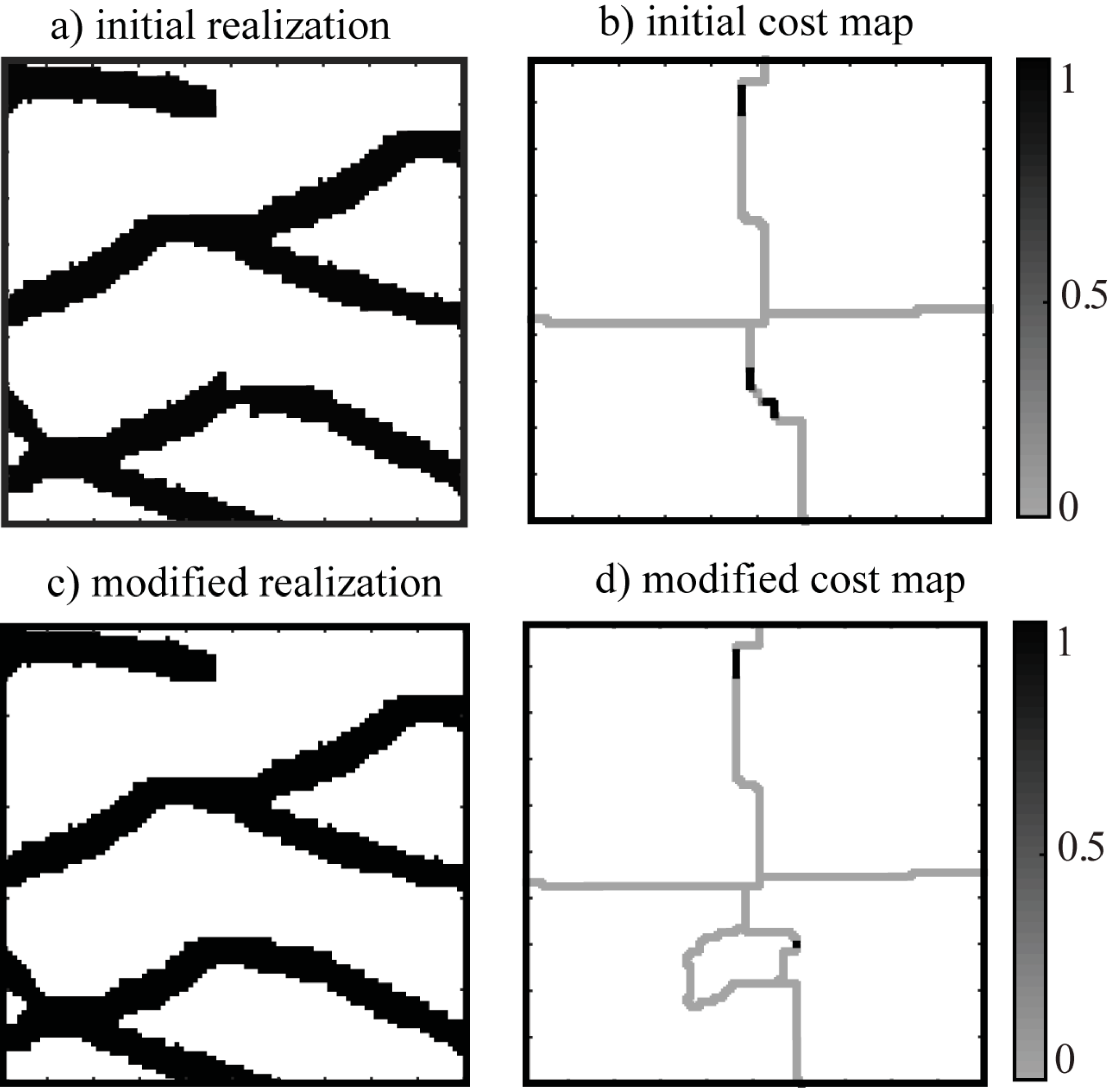

a) initial realization
b) initial cost map
1
0.5
0
c) modified realization
d) modified cost map
1
0.5
0

Figure 5. Figure

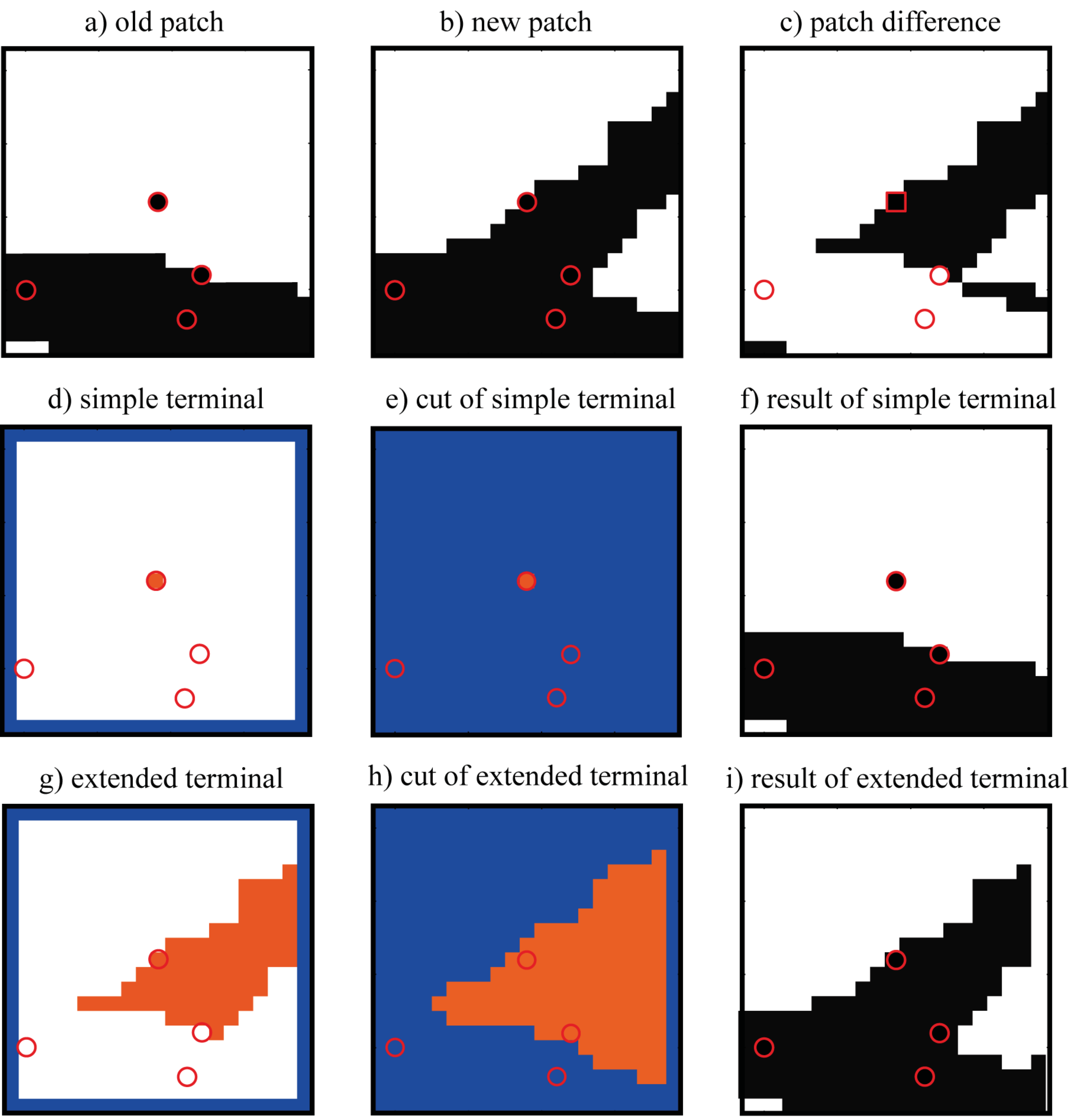
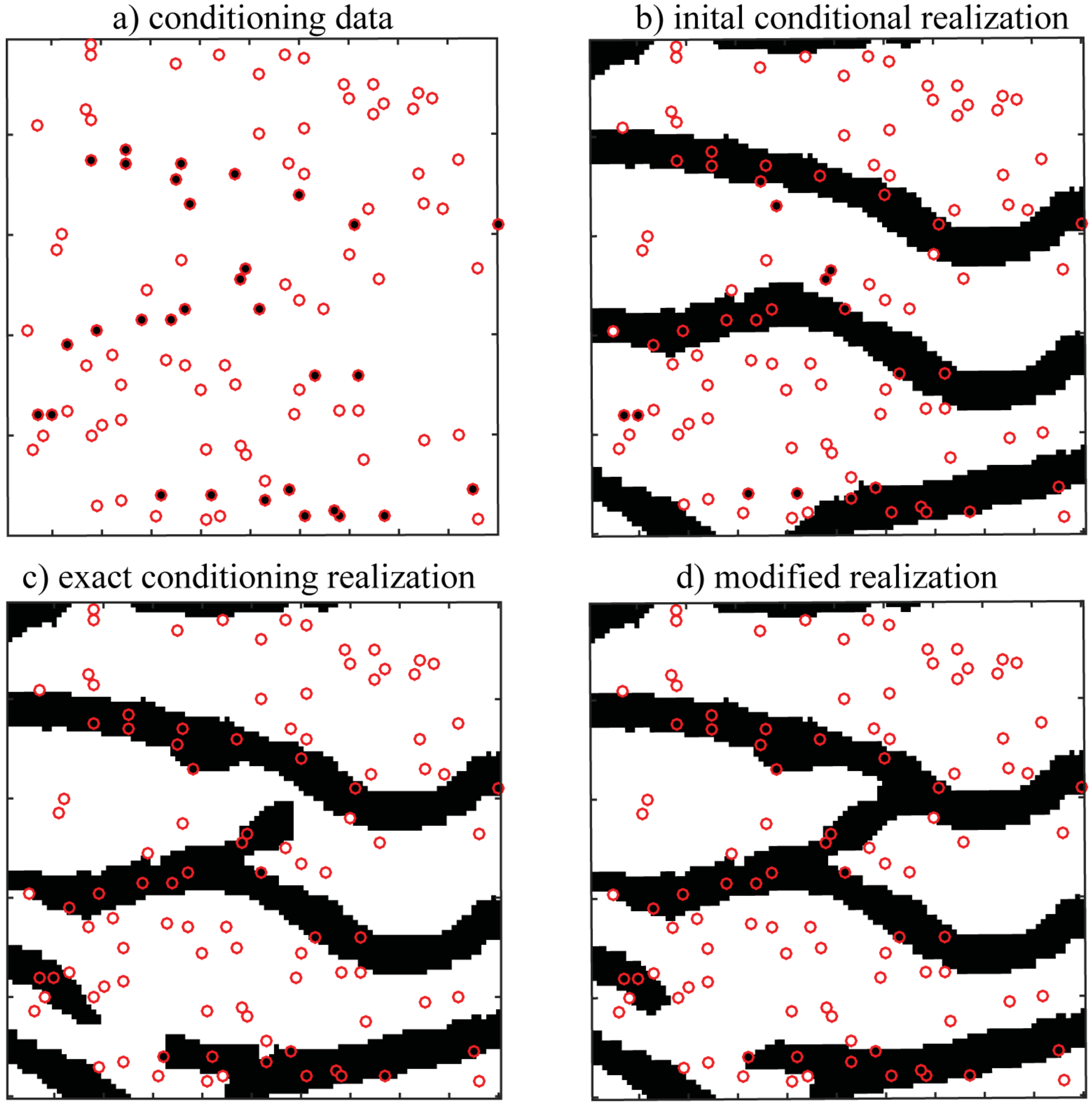

a) old patch
b) new patch
c) patch difference
d) simple terminal
e) cut of simple terminal
f) result of simple terminal
g) extended terminal
h) cut of extended terminal
i) result of extended terminal

Figure 6. Figure


a) conditioning data
b) inital conditional realization
c) exact conditioning realization
d) modified realization

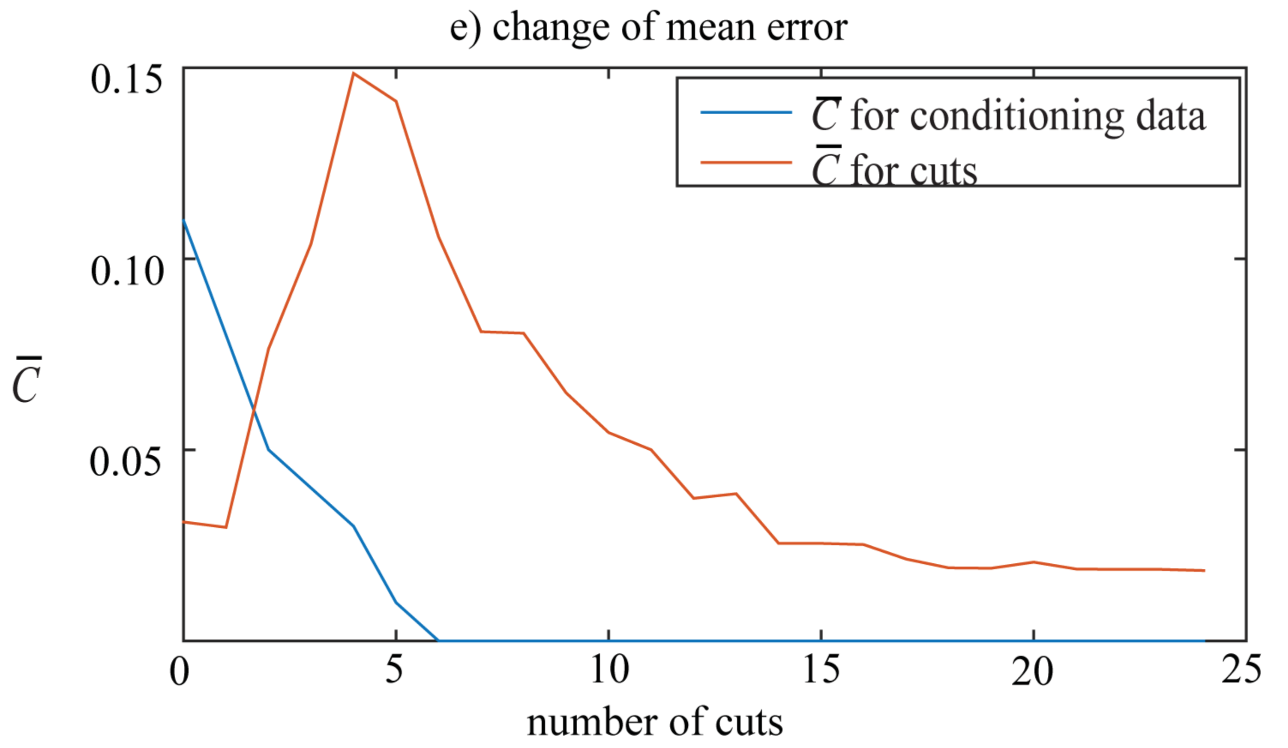

e) change of mean error
$\overline{C}$ for conditioning data
$\overline{C}$ for cuts
$\overline{C}$
0.15
0.10
0.05
0
5
10
15
20
25
number of cuts

Figure 7. Figure

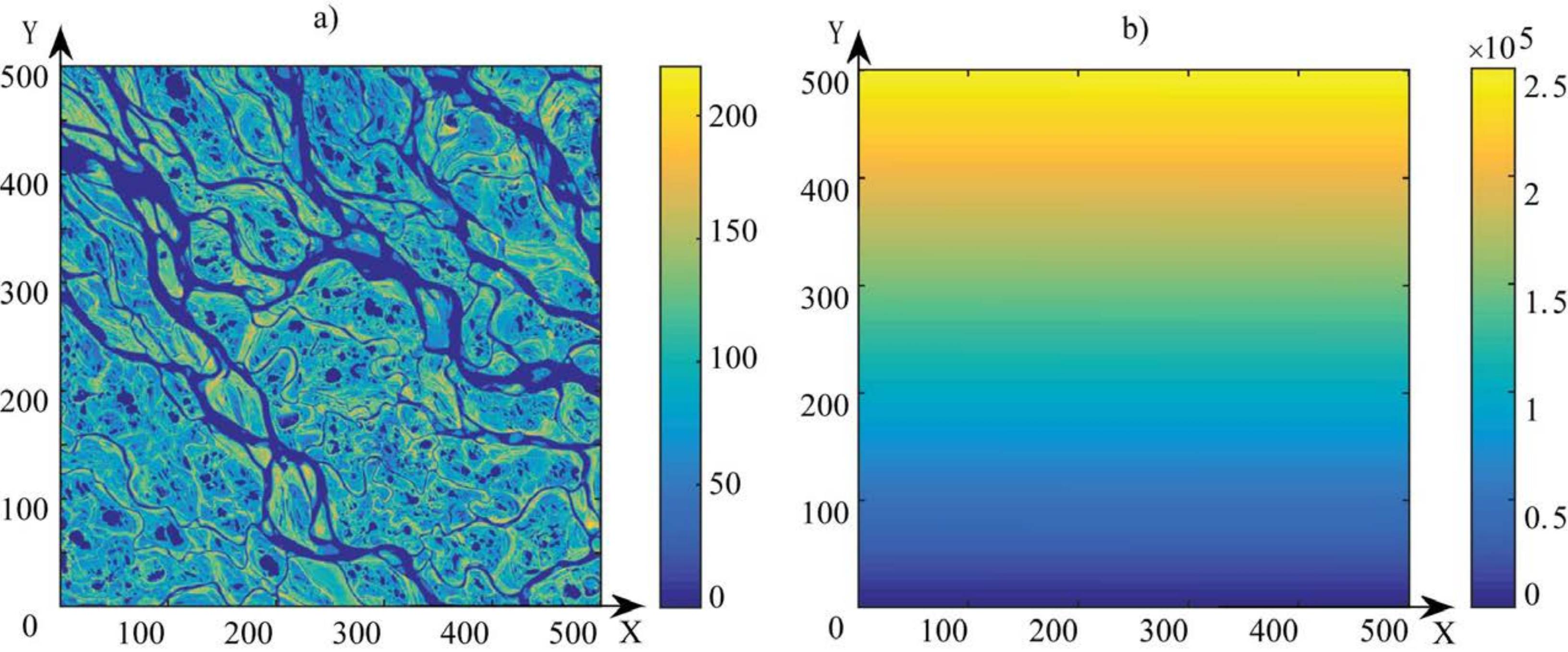

a)
b)
Y
X
0
100
200
300
400
500
×10^5
0.5
1
1.5
2
2.5
50
150

Figure 8. Figure

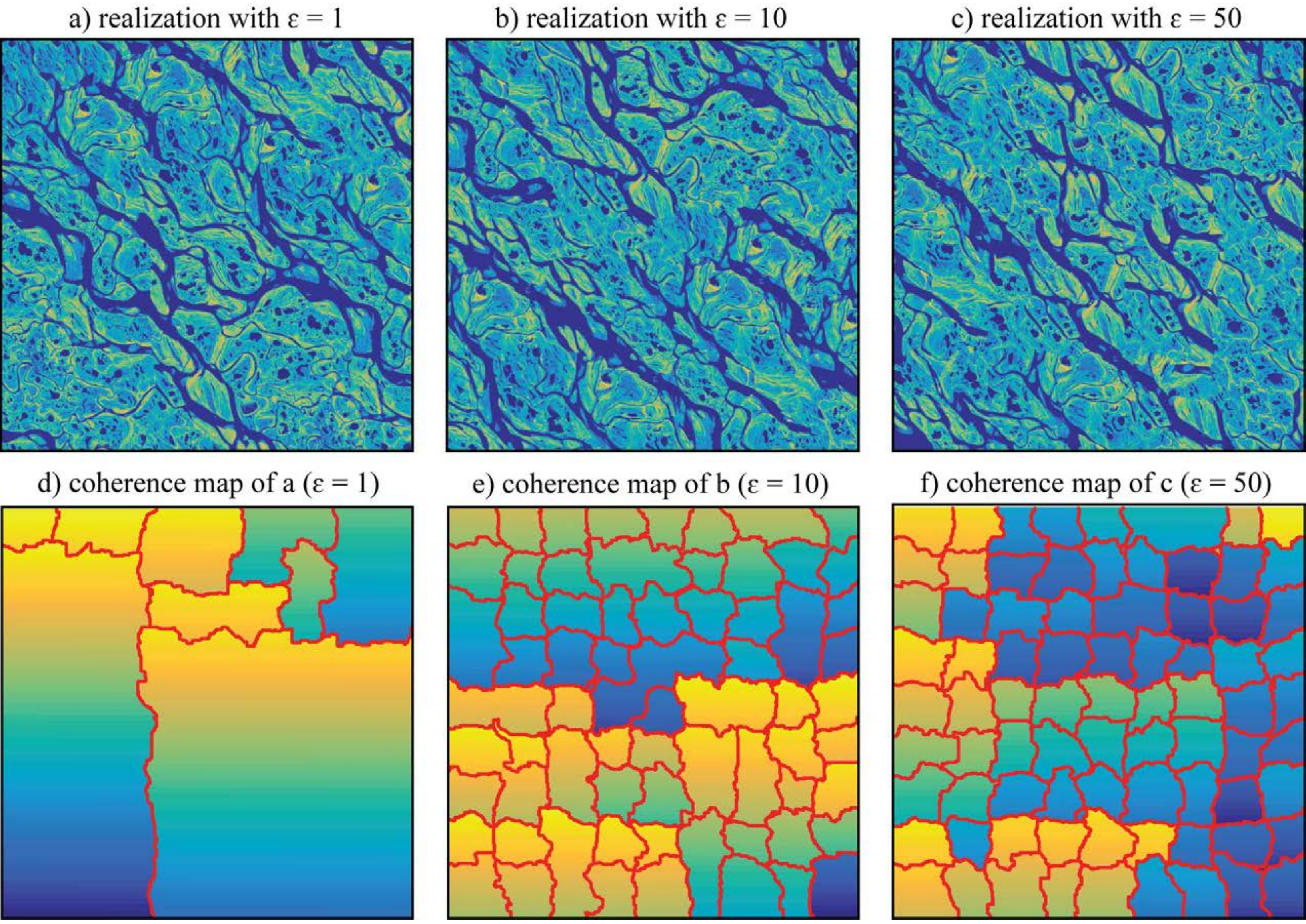

a) realization with ε = 1
b) realization with ε = 10
c) realization with ε = 50
d) coherence map of a (ε = 1)
e) coherence map of b (ε = 10)
f) coherence map of c (ε = 50)

Figure 9. Figure

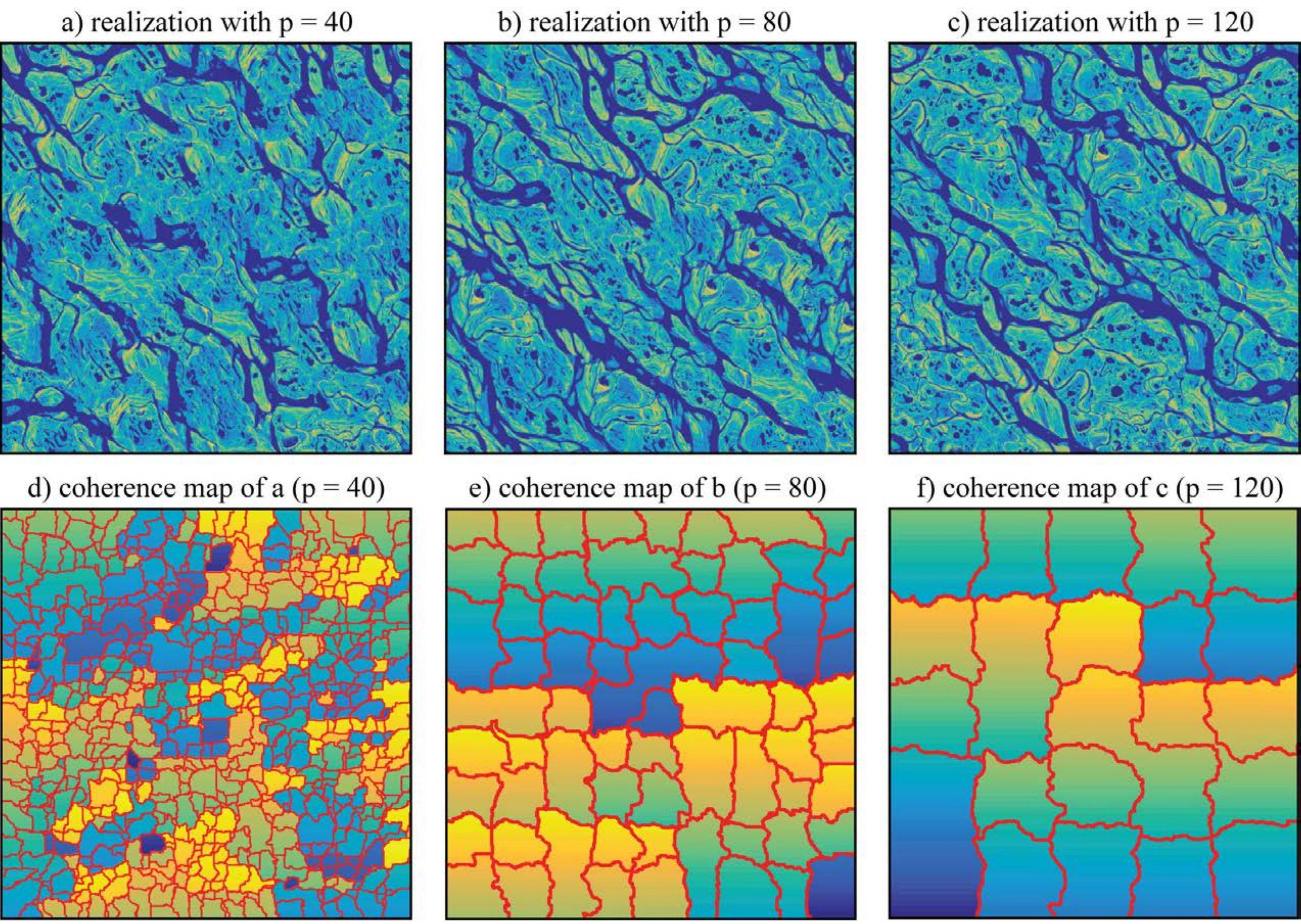

a) realization with p = 40
b) realization with p = 80
c) realization with p = 120
d) coherence map of a (p = 40)
e) coherence map of b (p = 80)
f) coherence map of c (p = 120)

Figure 10. Figure

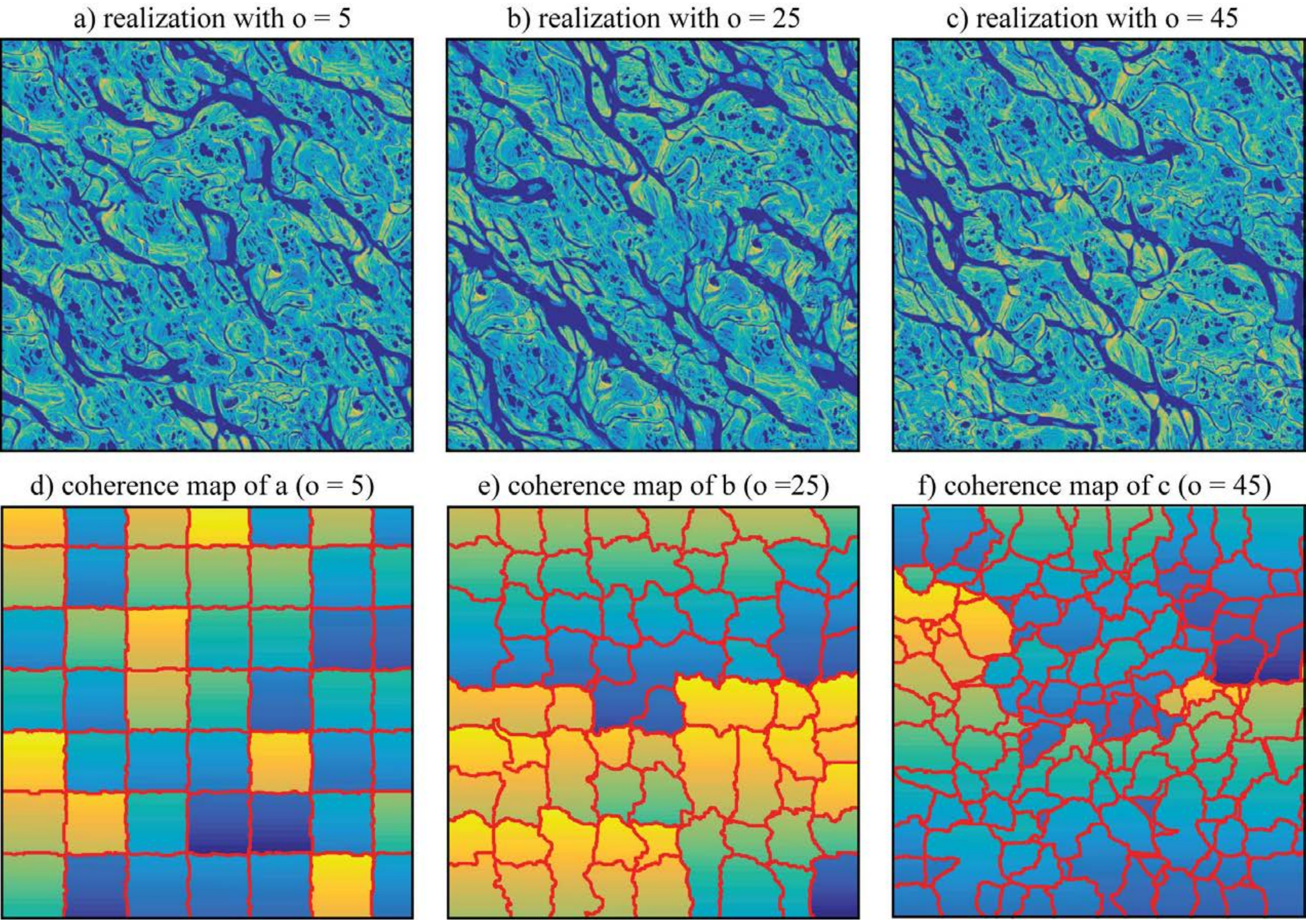

a) realization with o = 5
b) realization with o = 25
c) realization with o = 45
d) coherence map of a (o = 5)
e) coherence map of b (o =25)
f) coherence map of c (o = 45)

Figure 11. Figure

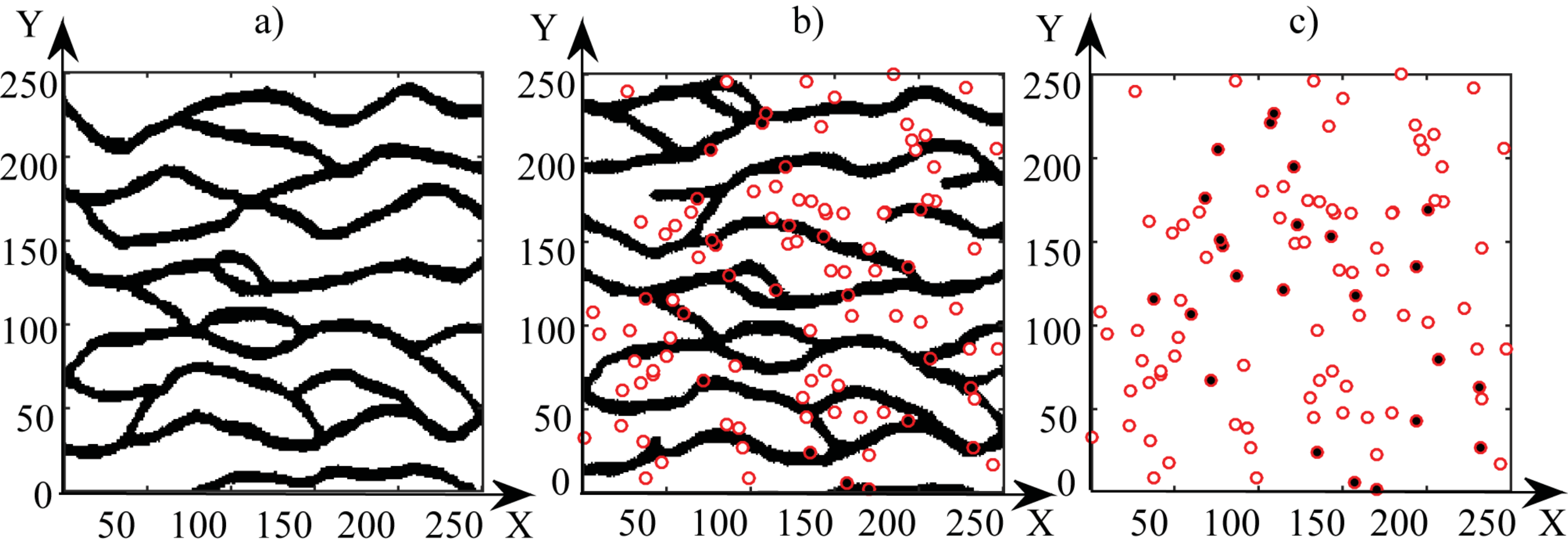

a)
b)
c)
Y
X
0
50
100
150
200
250

Figure 12. Figure

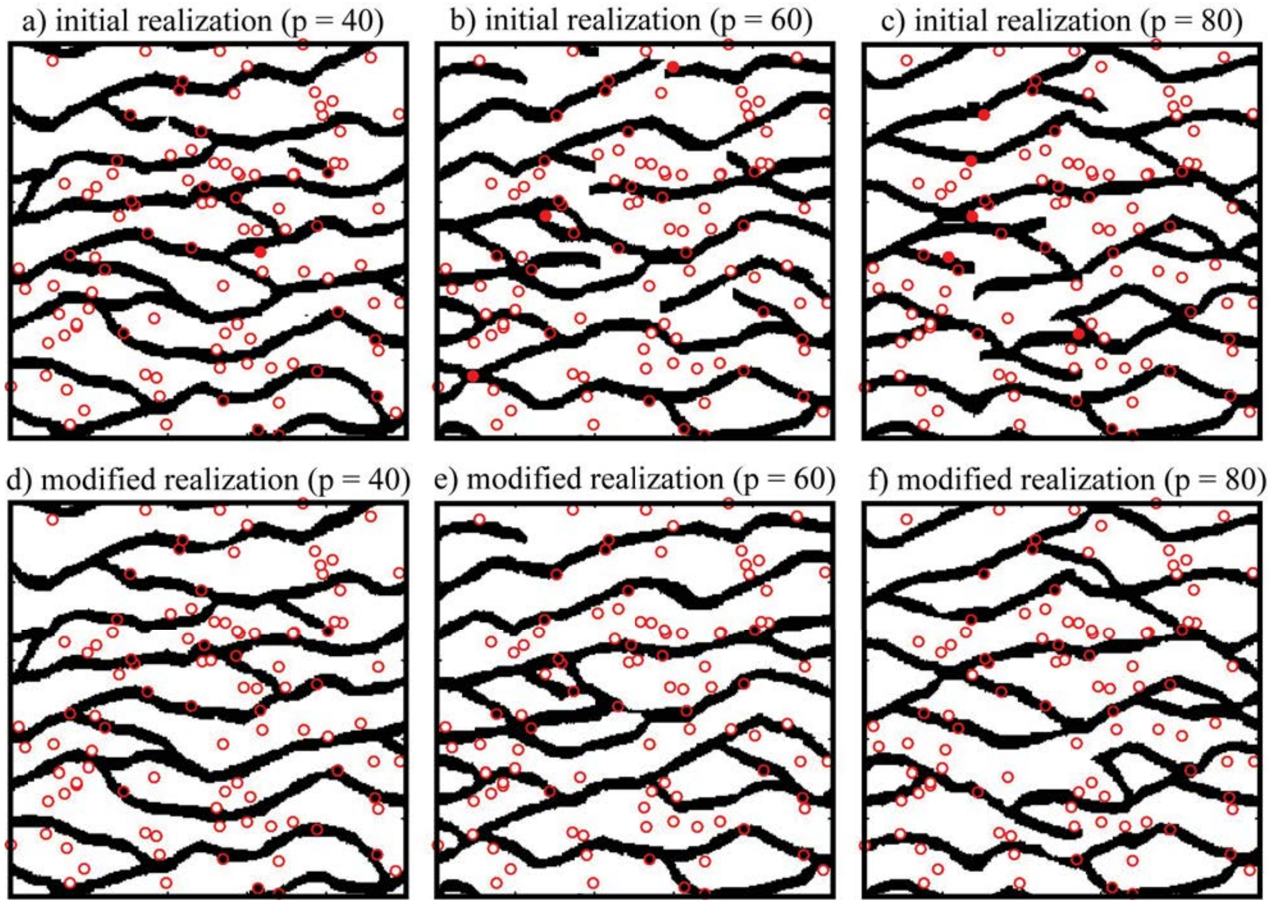
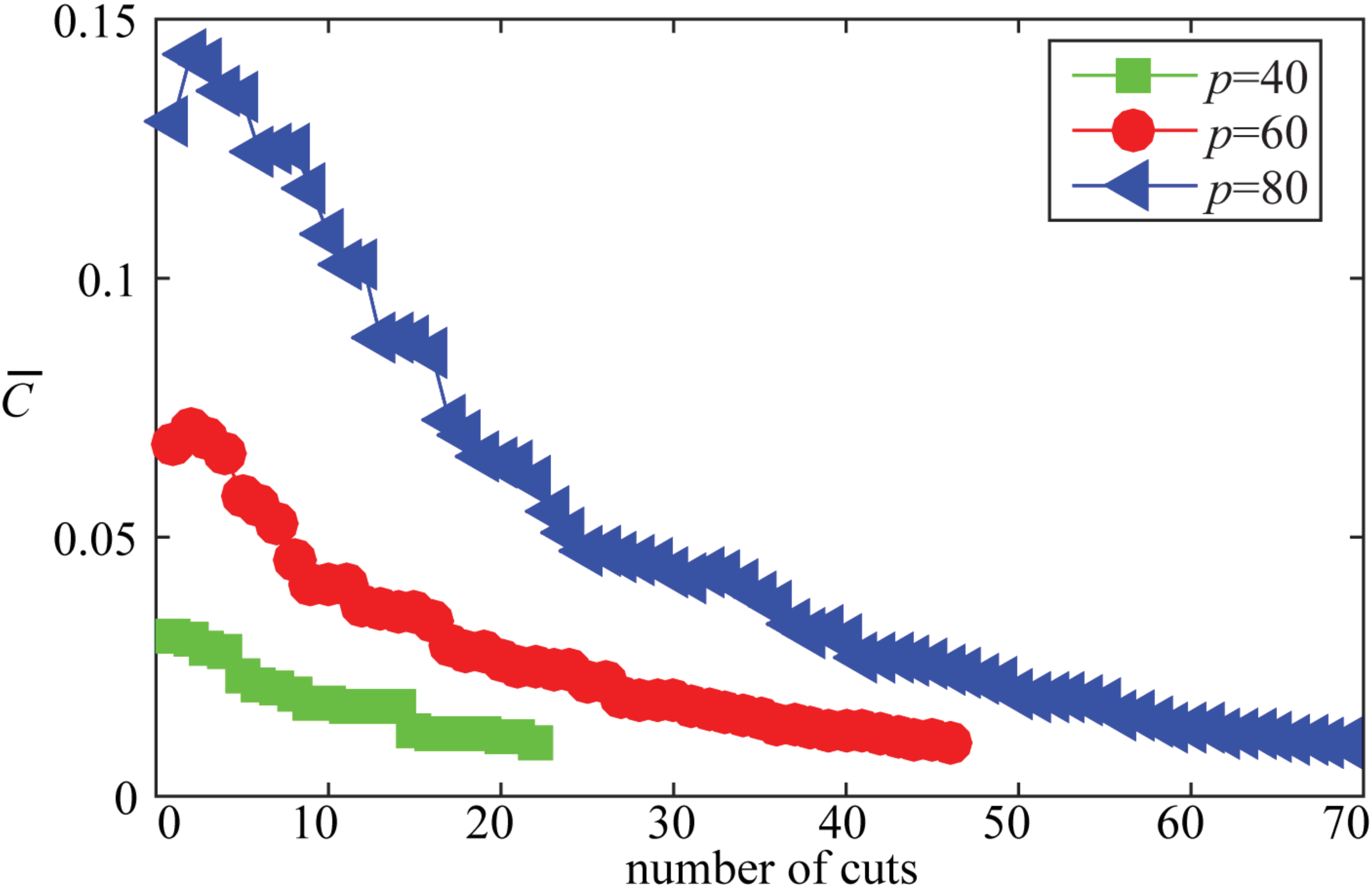

a) initial realization (p = 40)
b) initial realization (p = 60)
c) initial realization (p = 80)
d) modified realization (p = 40)
e) modified realization (p = 60)
f) modified realization (p = 80)

Figure 13. Figure



0.15
0.1
0.05
0
$\overline{C}$
0
10
20
30
40
50
60
70
number of cuts
p=40
p=60
p=80

Figure 14. Figure

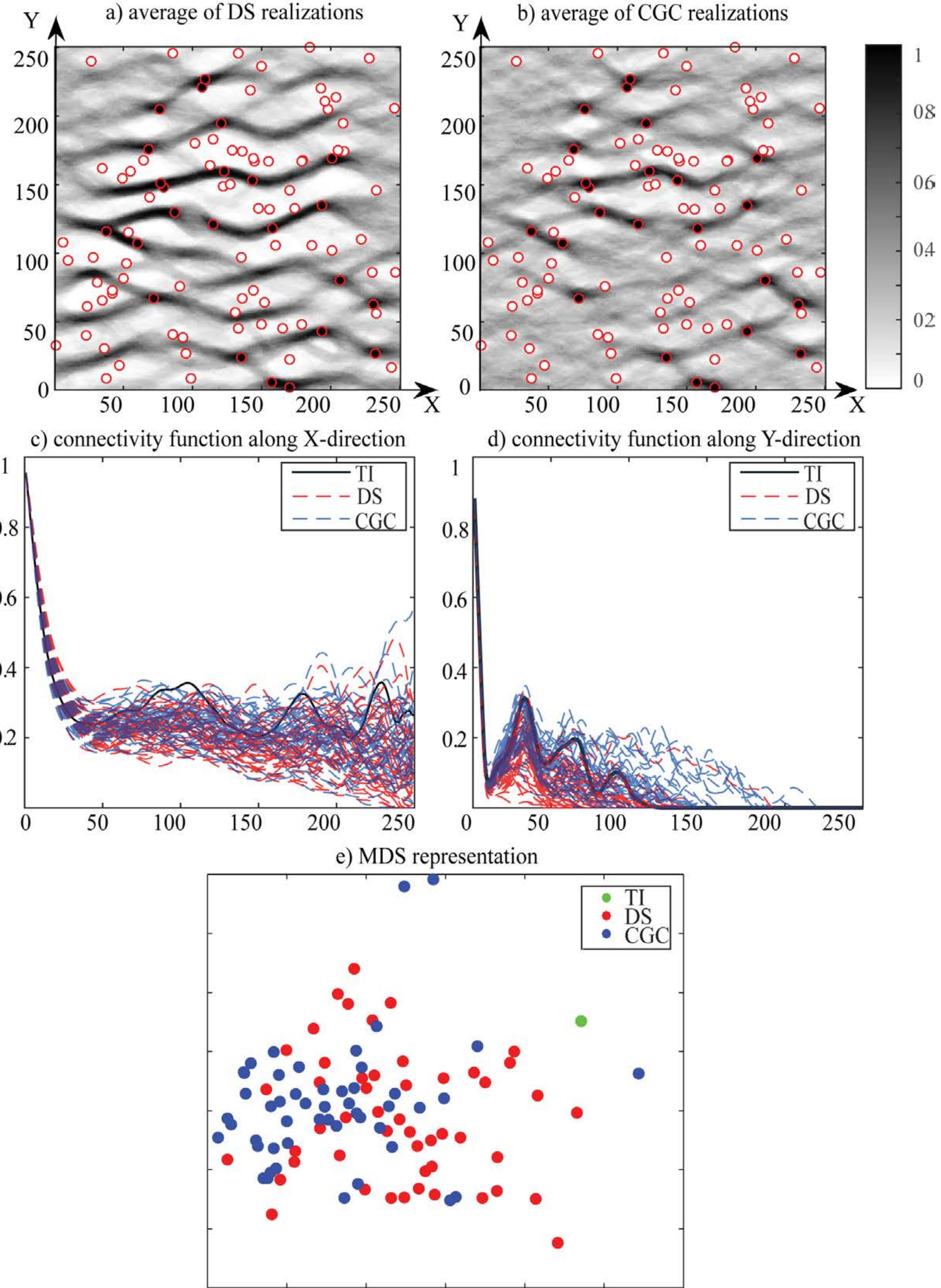

a) average of DS realizations
b) average of CGC realizations
Y
X
0
50
100
150
200
250
1
0.8
0.6
0.4
0.2
0
c) connectivity function along X-direction
d) connectivity function along Y-direction
TI
DS
CGC
e) MDS representation

Figure 15. Figure

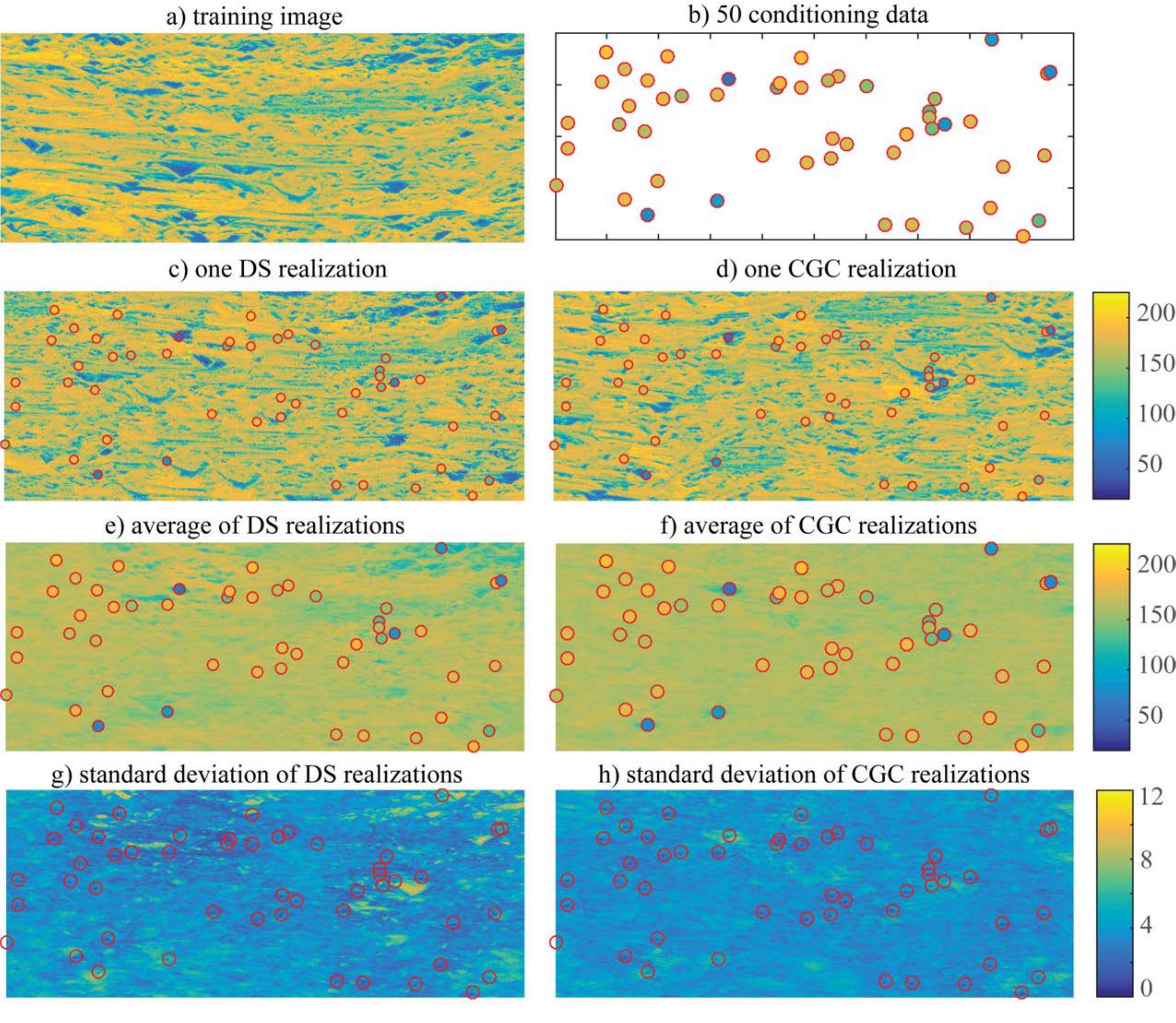

a) training image
b) 50 conditioning data
c) one DS realization
d) one CGC realization
200
150
100
50
e) average of DS realizations
f) average of CGC realizations
200
150
100
50
g) standard deviation of DS realizations
h) standard deviation of CGC realizations
12
8
4
0

Figure 16. Figure

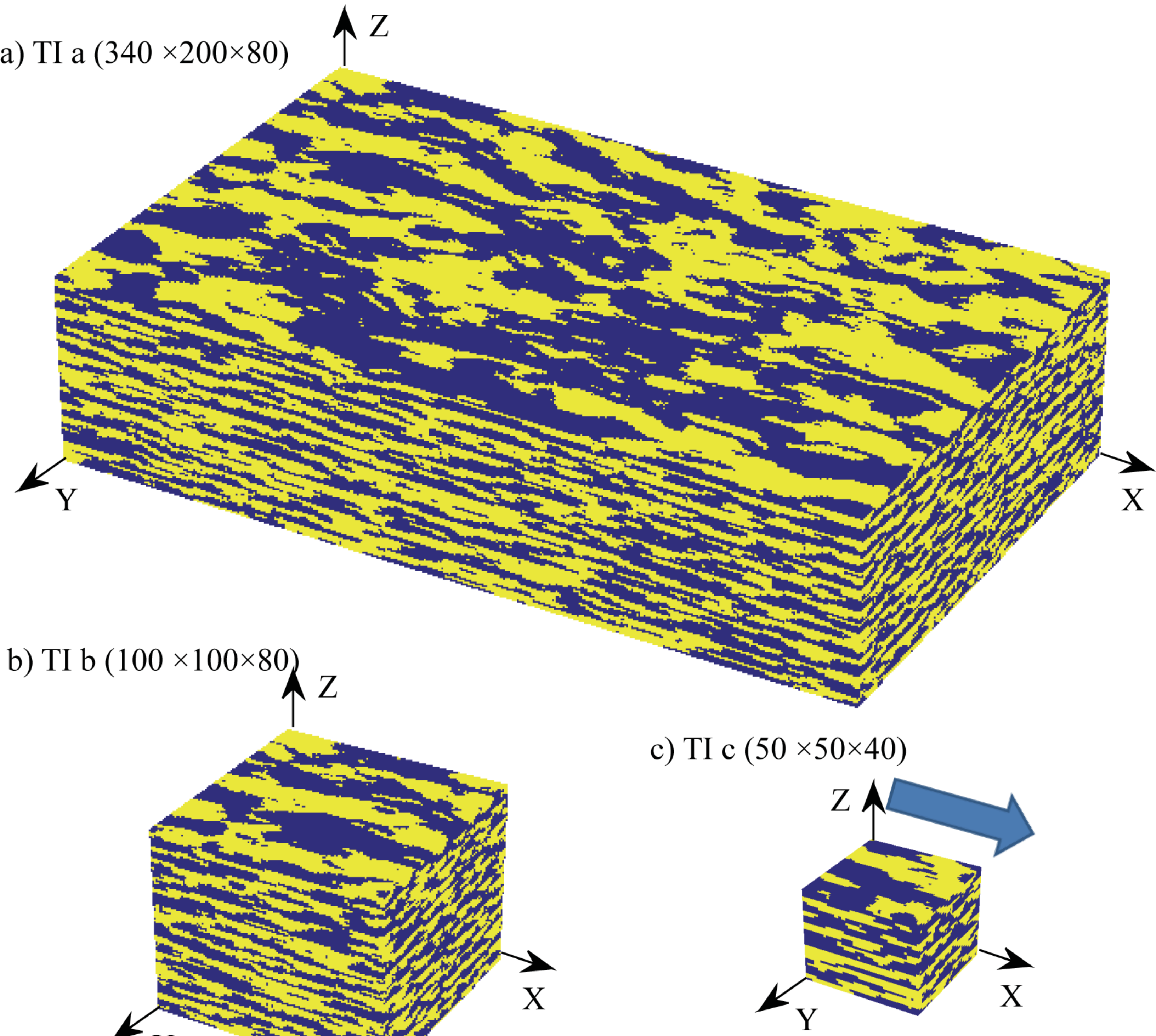
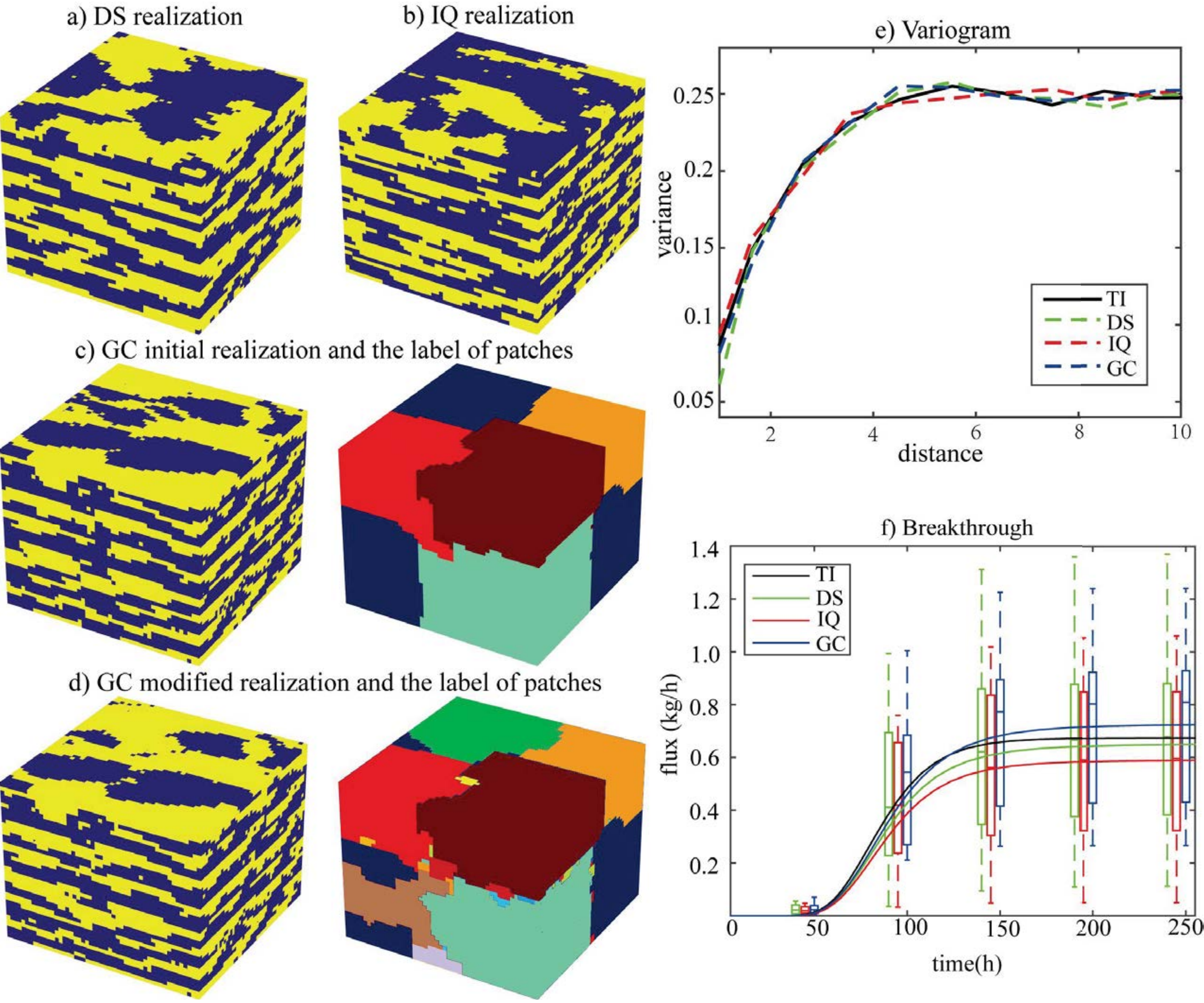

a) TI a (340 ×200×80)
Z
Y
X
b) TI b (100 ×100×80)
Z
X
Y
c) TI c (50 ×50×40)
Z
X
Y

Figure 17. Figure


a) DS realization
b) IQ realization
c) GC initial realization and the label of patches
d) GC modified realization and the label of patches
e) Variogram
variance
distance
0.05
0.1
0.15
0.2
0.25
2
4
6
8
10
TI
DS
IQ
GC
f) Breakthrough
flux (kg/h)
time(h)
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250
TI
DS
IQ
GC