Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Nuno de Souza Moura

LSHSIM: A Locality Sensitive Hashing Based Method for Multiple-Point Geostatistics

Tese de Doutorado

Thesis presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio, as partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática.

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
September 2017

Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Nuno de Souza Moura

LSHSIM: A Locality Sensitive Hashing Based Method for Multiple-Point Geostatistics

Thesis presented to the Programa de Pós-Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática. Approved by the undersigned Examination Committee.

Prof. Eduardo Sany Laber
Advisor
Departamento de Informática — PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. Sinésio Pesco
Departamento de Matemática — PUC-Rio

Prof. Artur Alves Pessoa
UFF

Prof. Alexandre Anozé Emerick
Petroleo Brasileiro – Rio de Janeiro – Matriz

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico da PUC-Rio

Rio de Janeiro, September 21th, 2017

Pedro Nuno de Souza Moura

Bachelor's in Information Systems at the Federal University of the State of Rio de Janeiro (2008). Masters' in Informatics at the Pontifical Catholic University of Rio de Janeiro (2011), with emphasis in Combinatorial Optimization.

Bibliographic data

Moura, Pedro Nuno de Souza

LSHSIM: A Locality Sensitive Hashing Based Method for Multiple-Point Geostatistics / Pedro Nuno de Souza Moura ; advisor: Eduardo Sany Laber. — 2017.
93 f. : il. ; 30 cm

Tese (Doutorado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2017.
Inclui bibliografia

1. Informática – Teses. 2. Geoestatística Multiponto; *Locality Sensitive Hashing*; *Run-Length Encoding*; Modelagem de Padrões; Imagem de Treinamento. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

First and foremost, I would like to thank God for giving me strength to face the challenges and the difficulties of a phd course.

I dedicate this conquest to my grandmother Carmen for her endless love and for always being by my side in my moments of study.

I would also like to dedicate to my parents, Mário and Fernanda, for teaching me the importance of studying and for staying there whenever I needed. Besides, I want to thanks my grandmother Maria de Lourdes, who gave me so much strength even though she was far away from Brazil.

To my brother Fernando, for all support and aid during this process.

To my beloved Nati, for all support during this process. Thanks for showing me the biggest example of complicity I've ever seen. I love you!

To my advisor Prof. Eduardo Laber, for not only teaching me how to be a researcher, but also for all knowledge passed. Honestly, I cannot think of a better advisor.

To Prof. Hélio Lopes, for introducing me to the field of Geostatistics, for the valuable contributions and for creating the SIMPAD project in which this research was born.

To Prof. Marcus Poggi, for all support and guidance during my Master's degree and the beginning of the PhD course. In addition, I would also like to thank you for the opportunity of working in Gapso (now Accenture).

To Prof. Amâncio, Prof. Luiz Pedro Jutuca and Profa. Simone from UNIRIO for encouraging me to follow an academic trajectory since the beginning. To Prof. Alexandre Andreatta for teaching me so many things in the undergraduate course and being a professor to whom I look up to.

To Prof. Tanaka, Geiza, Adriana, Beto, Morganna, Mariana, Flávia, Renata, Kate, Leila, Leo Azevedo, Leo Rocha, Márcio, Sean, Sidney, Jefferson, Vânia, Bruno, Heloísa, Alessandra, Douglas, Leandro, Ivana, Neide and all workers from UNIRIO, for the day by day company.

To all my colleagues from Galgos laboratory, for sharing moments of not only work, but also joy and happiness. To João, Chico, Lucas, Daniel Mesejo and Gabriel for participating and contributing to this research in so many moments. Your aid and hard work were essential for the implementation of this research.

To my students Thiago Albuquerque, Daniel Villaça and Renard from UNIRIO who became great friends and with whom I always great moments.

To my great friends from the gym, André and Demétrius, for the so many moments of joy and for always pushing me to seek to be a better person.

To my friends Felipe e Gabriel, for always being there through the years in my life. Your support was essential for the accomplishment of this work.

To my friends Igor, Marcelo e Jorge for the amazing conversations through the days that helped me a lot in this process. Also to Victor and Mateus for so many moments of happiness, although we live far apart.

To my friends Luanna and Anne for always being willing to listen to me and help me in the most varied situations.

To Fathers Lee and Alexandre Pacioli for the spiritual guidance in this process.

To CNPq for the conceded scholarship in the beginning of the PhD course and to CAPES for later paying the taxes of the course.

Abstract

Moura, Pedro Nuno de Souza; Laber, Eduardo Sany (advisor). **LSHSIM: A Locality Sensitive Hashing Based Method for Multiple-Point Geostatistics**. Rio de Janeiro, 2017. 93p. Tese de Doutorado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Reservoir modeling is a very important task that permits the representation of a geological region of interest. Given the uncertainty involved in the process, one wants to generate a considerable number of possible scenarios so as to find those which best represent this region. Then, there is a strong demand for quickly generating each simulation. Since its inception, many methodologies have been proposed for this purpose and, in the last two decades, multiple-point geostatistics (MPS) has been the dominant one. This methodology is strongly based on the concept of training image (TI) and the use of its characteristics, which are called patterns. In this work, we propose a new MPS method that combines the application of a technique called Locality Sensitive Hashing (LSH), which permits to accelerate the search for patterns similar to a target one, with a Run-Length Encoding (RLE) compression technique that speeds up the calculation of the Hamming similarity. We have performed experiments with both categorical and continuous images which showed that LSHSIM is computationally efficient and produce good quality realizations, while achieving a reasonable space of uncertainty. In particular, for categorical data, the results suggest that LSHSIM is faster than MS-CCSIM, one of the state-of-the-art methods.

Keywords

Multiple-Point Geostatistics; Locality Sensitive Hashing; Run-Length Encoding; Pattern Modeling; Training Image

Resumo

Moura, Pedro Nuno de Souza; Laber, Eduardo Sany. **LSHSIM: Um Método de Geoestatística Multiponto Baseado em *Locality Sensitivity Hashing***. Rio de Janeiro, 2017. 93p. Tese de Doutorado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A modelagem de reservatórios consiste em uma tarefa de muita relevância na medida em que permite a representação de uma dada região geológica de interesse. Dada a incerteza envolvida no processo, deseja-se gerar uma grande quantidade de cenários possíveis para se determinar aquele que melhor representa essa região. Há, então, uma forte demanda de se gerar rapidamente cada simulação. Desde a sua origem, diversas metodologias foram propostas para esse propósito e, nas últimas duas décadas, *Multiple-Point Geostatistics* (MPS) passou a ser a dominante. Essa metodologia é fortemente baseada no conceito de imagem de treinamento (TI) e no uso de suas características, que são denominadas de padrões. No presente trabalho, é proposto um novo método de MPS que combina a aplicação de dois conceitos-chave: a técnica denominada *Locality Sensitive Hashing* (LSH), que permite a aceleração da busca por padrões similares a um dado objetivo; e a técnica de compressão *Run-Length Encoding* (RLE), utilizada para acelerar o cálculo da similaridade de Hamming. Foram realizados experimentos com imagens de treinamento tanto categóricas quanto contínuas que evidenciaram que o LSHSIM é computacionalmente eficiente e produz realizações de boa qualidade, enquanto gera um espaço de incerteza de tamanho razoável. Em particular, para dados categóricos, os resultados sugerem que o LSHSIM é mais rápido do que o MS-CCSIM, que corresponde a um dos métodos componentes do estado-da-arte.

Palavras-chave

Geoestatística Multiponto; *Locality Sensitive Hashing*; *Run-Length Encoding*; Modelagem de Padrões; Imagem de Treinamento

Contents

List of Figures

List of Tables

Pain is temporary. It may last a minute, or an hour, or a day, or a year, but eventually it will subside and something else will take its place. If I quit, however, it lasts forever.

Lance Armstrong

1
Introduction

1.1
Motivation

Geostatistics was conceived as a discipline developed by researchers when facing with real problems and searching for a consistent set of numerical tools that would help them address these problems (Pyrcz & Deutsch, 2014). The reasons that led to this are: an increasing number of data to deal with, a need to address problems with consistent and reproducible methods, among others. On top of everything, a major reason was the fact that more reliable and profitable decisions would be made with improved numerical models (Pyrcz & Deutsch, 2014). A central point, then, was the uncertainty involved in this decision-making process.

One of the tools provided by geostatistics for modeling spatial phenomena is the stochastic conditional simulation. The stochastic simulation aims to create multiple, realistic representations, termed *realizations*, of a studied spatial phenomenon, that are constrained (conditioned) to any available data (Arpat & Caers, 2007).

Stochastic simulation permits to generate multiple realizations to jointly represent the model uncertainty, where each realization reproduces the specified input statistics and well data. Figure 1.1 exhibits two three-dimensional realizations of size 1 km $\times$ 1 km $\times$ 20 m generated by an Object-based model (Caers, 2011), in which the channels and associated facies (types of rock) fills are shown. In this example, the channels correspond to the sinuous objects and each facie corresponds to a different colour in model.

Once these realizations defining the facies architecture are generated, some reservoir properties such as porosity and permeability can be simulated. The simulation work flow reproduces the representative porosity and permeability distributions along with their spatial continuity and associated uncertainty (Pyrcz & Deutsch, 2014). Figure 1.2 illustrates two three-dimensional porosity realizations of size 1 km $\times$ 1 km $\times$ 20 m for the facies simulations of Figure 1.1, honoring the well data and input distributions. In each realization, colours close to red indicate high porosity, while values close to blue denote low

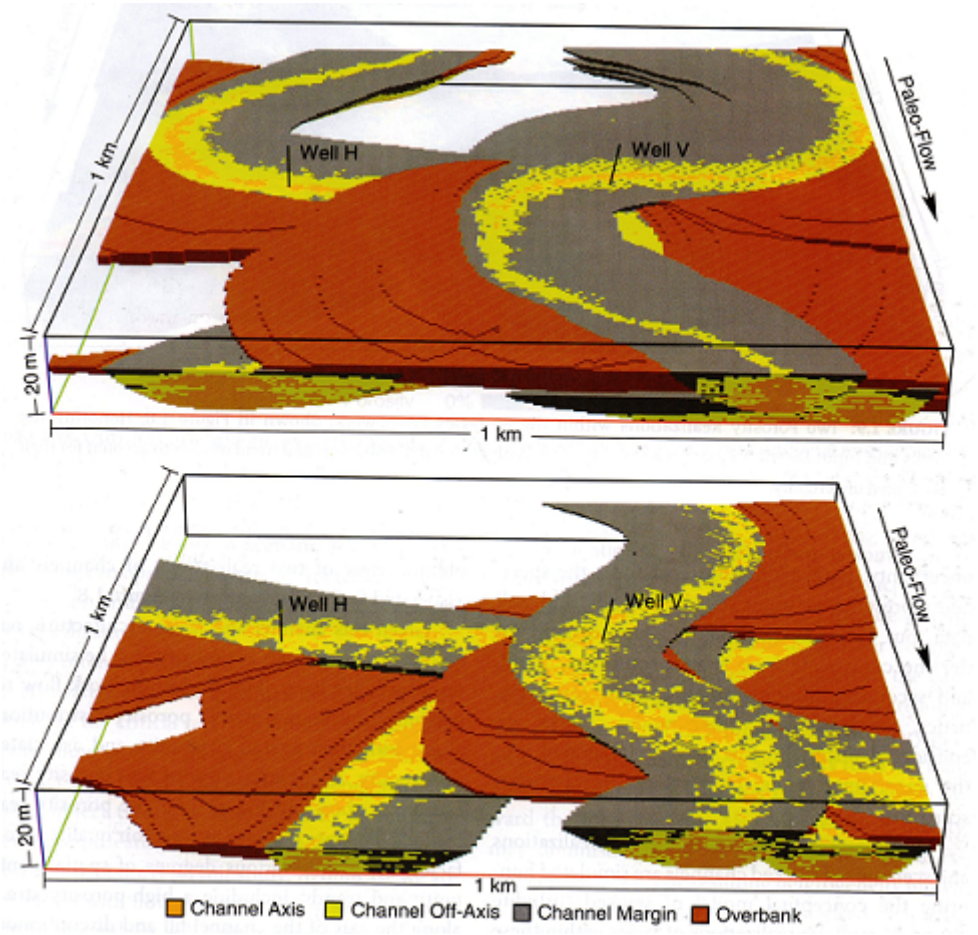

Figure 1.1: Two 3D realizations of Object-based channels with facies, conditional to the available well data and input statistics (Pyrcz & Deutsch, 2014).

porosity. This gives a perspective of the reservoir quality across facies along with the spatial continuity.

The permeability property may also be simulated. This property has a relationship with porosity within each facies and can be conditioned to any permeability information available along the wells (Pyrcz & Deutsch, 2014). Figure 1.3 illustrates a permeability realization cosimulated with the first porosity realization of Figure 1.2, where colours close to red indicate high permeability values and colours close to blue indicate low permeability values.

Production data, i.e., the real measurements of extracted oil from a given reservoir, brings important information about the spatial distribution of that reservoir variables. However, it rarely suffices to characterize heterogeneous reservoirs and a large amount of uncertainty still remains after the history matching of geostatistical models (Caers, 2002). In other words, the production data give information until some instant T in time, after which there is uncertainty. In this way, one desires to forecast how the production would

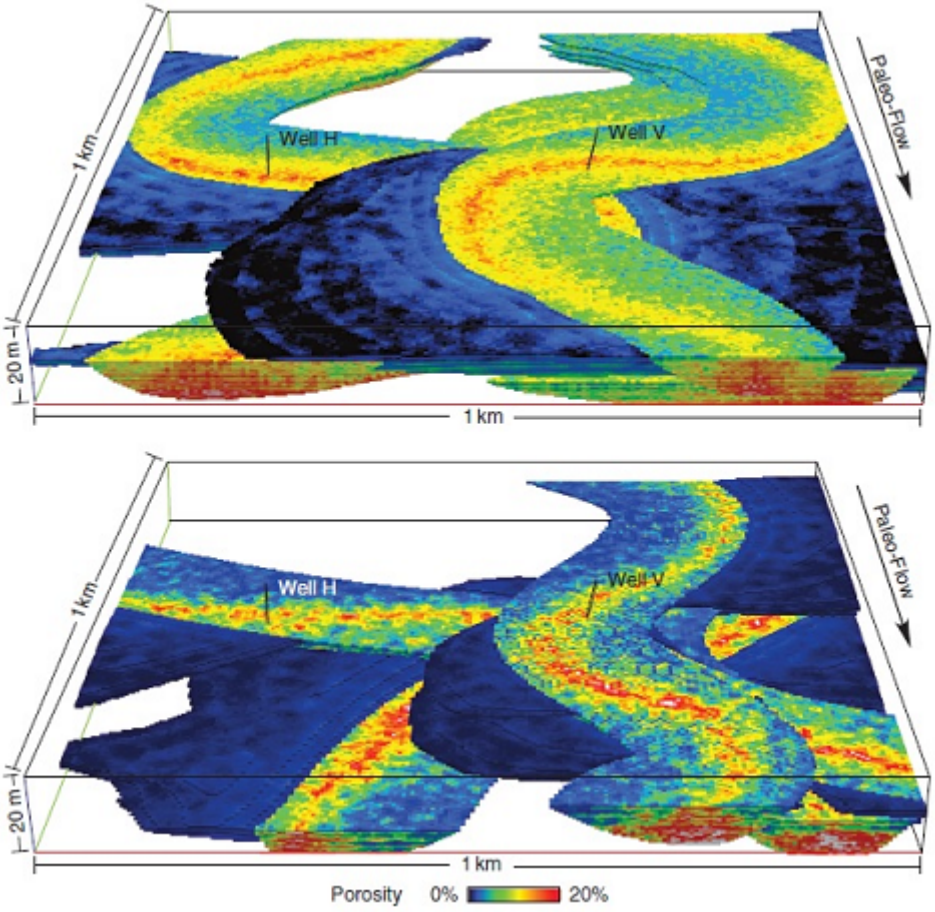

Figure 1.2: Two 3D porosity realizations for the facies simulation of Figure 1.1, honoring the wells and input distributions. (Pyrcz & Deutsch, 2014).

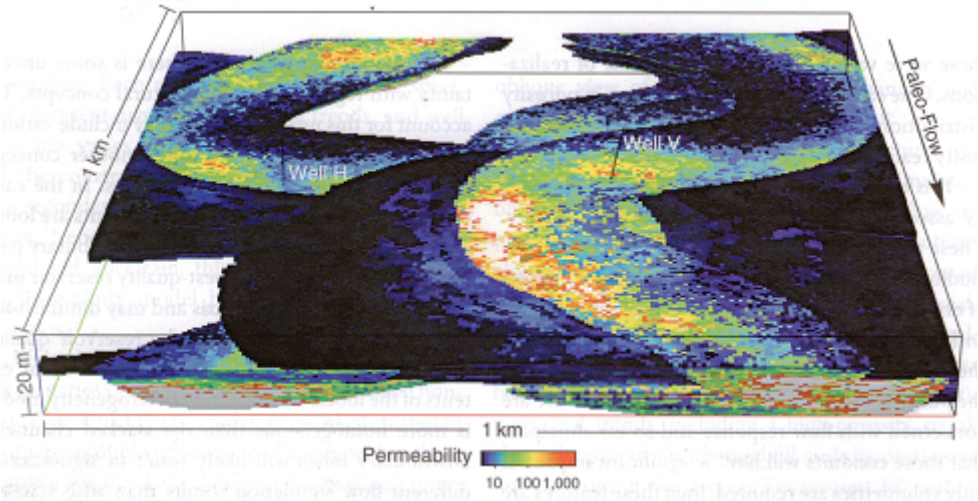

Figure 1.3: A 3D permeability realization cosimulated with the first porosity realization of Figure 1.2 (Pyrcz & Deutsch, 2014).

be after this point in time.

After simulating porosity and permeability, each realization is submitted to a flow simulator, so as to obtain the dynamic response, which is its production curve (or fractional flow) estimated from a time instant 0 and up to some point in time much later than T. These production curves are compared to the real production curve in the interval $[0, T]$ and the associated realizations are modified in an iterative process, until their production curves fit to the real production curve in that interval. For example, Figure 1.4 illustrates this fact, where a finite difference simulator was used to converge the fractional flow obtained for some simulated scenarios to the target reference production (Caers, 2002).

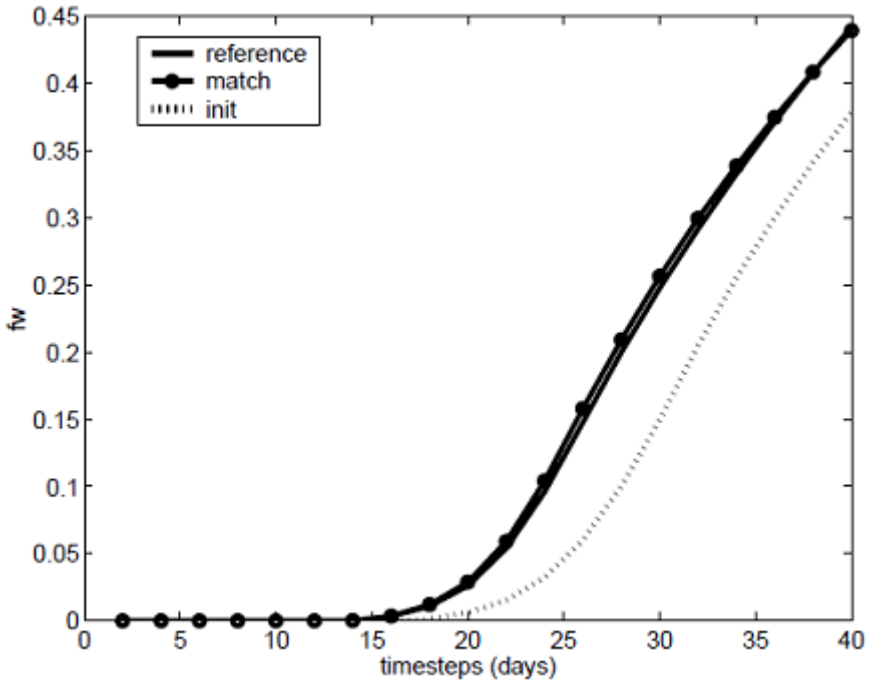

Figure 1.4: History match to the flow data fw (Caers, 2002).

At this point, from all generated realizations, one can make production forecast and thus some specific realizations are selected as representatives, which typically correspond to percentile models: the $P10$ model, such that 10% of the model responses are lower than this model; the $P50$ model, which corresponds to the median model, such that 50% of the models have smaller response and 50% have higher response; and the $P90$, such that only 10% of the model responses are higher than this one. In sum, these selected representatives are used to forecast the produced oil and support the decision-making process. Therefore, the final product is an uncertainty model for reservoir response, given the exploitation method (Pyrcz & Deutsch, 2014).

One important entity in this process is the investor (or sponsor), which is the one who is paying and investing to extract oil from a given geological region. He has great expectation in performing a good forecast of the production and minimizing the risk associated with it, so as to obtain a high return on investment (ROI). Therefore, the financial impact plays a major role in this kind of study.

1.2 Problem Statement

In the last few decades, multiple-point geostatistics (MPS) became very popular. It provides a variety of techniques to model and simulate facies scenarios of reservoirs for a given geological region. MPS was born out of a need to address the issue of lack of physical reality as well as the lack of control in the simulated fields in traditional modeling (Mariethoz & Caers, 2014). Besides, in contrast to traditional parametric techniques based on variogram, which make use of two points statistics, MPS is non-parametric and based on higher-order statistics to describe complex structures. It generates less artificial simulations, having more realistic geological characteristics (Mariethoz & Caers, 2014).

The source of these statistics and a fundamental concept in this area is the *training image* (TI), which typically represents a specific geological region of interest. The goal of MPS is to mimic physical reality and the vehicle to achieve this is the TI (Mariethoz & Caers, 2014). The TI is usually a hand draw made by a specialist, such as a geologist, or obtained by the application of another type of technique, such as a Boolean model realization (Caers, 2011). Figure 1.5 exhibits an example of training image, which is the Strebelle TI (Strebelle, 2002), where the white stripes represent sand channels (good reservoir quality) and black pixels represent the background with poor reservoir quality.

The aim is to generate simulations following the geometry of facies associations seen in the TI while honoring specific constraints related to reservoir data (Chilès & Delfiner, 2012). In fact, these constraints correspond to real measurements (a.k.a. *hard data*) made in the regions of interest using some suitable equipment. When the hard data are not used/available, the process is called unconditional; otherwise, it is called conditional and every realization must honor these data.

The first methods in literature were based on simulating each pixel (or node) of the realization, while more recent ones follow an approach that is called pattern-based, because it is highly focused on the extraction and reproduction of a contiguous group of pixels from the TI.

In the geostastitical field, Zhang et al. (2006) and Arpat & Caers (2007)

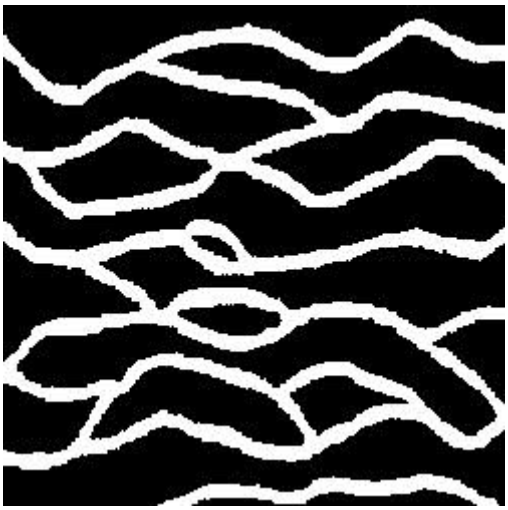

Figure 1.5: The Strebelle training image.

were the first to propose working with patterns. Generally, in a pattern-based MPS approach, a realization (scenario) is built through the execution of a loop where the two following steps are performed several times: (i) a location of a certain size/shape of the realization under construction is selected; (ii) this location is replaced with a similar pattern of the same size/shape from the TI. The locations selected from the realization are known in the MPS literature as *data events*. The similarity between patterns and data events is defined according to some similarity/distance measure (e.g. Euclidean distance) (Arpat & Caers, 2007).

The Figure 1.6 illustrates this process by considering a TI containing black and white facies. From left to right, it shows a realization, a data event defined at a given location and its comparison with patterns of the TI. One of these patterns is chosen and pasted at that location. Note that blue values in realization correspond to regions that are not yet filled.

In a simulation, the size of the patterns/data events considered is known in the MPS literature as *template size*. For example, the template size adopted in Figure 1.6 was 5×5. In addition, Figure 1.7 shows two realizations of 250×250 pixels for the TI in Figure 1.5 where the template size varied. In (A) the template size was set to 10×10 and in (B) it was defined to 50×50. It can be noted that the realization (A) using the smaller template size was too attached to small features of the TI and did not express well its characteristics, while the simulation (B) using the bigger template size was able to capture the image's spatial continuity.

In typical applications where MPS is employed, a large number of realizations has to be generated, so that the time taken to perform each one

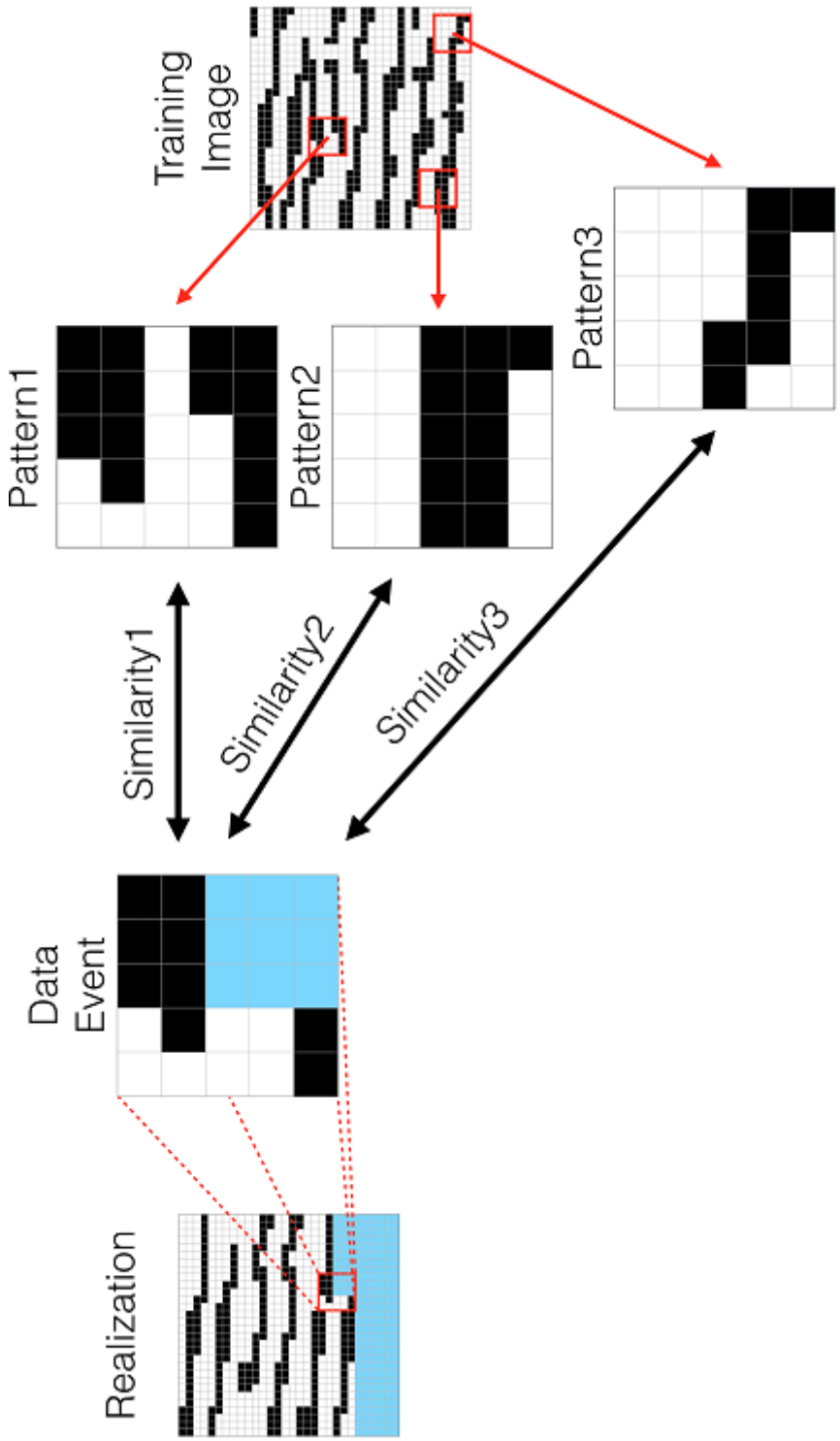

Figure 1.6: General structure of a pattern-based approach.

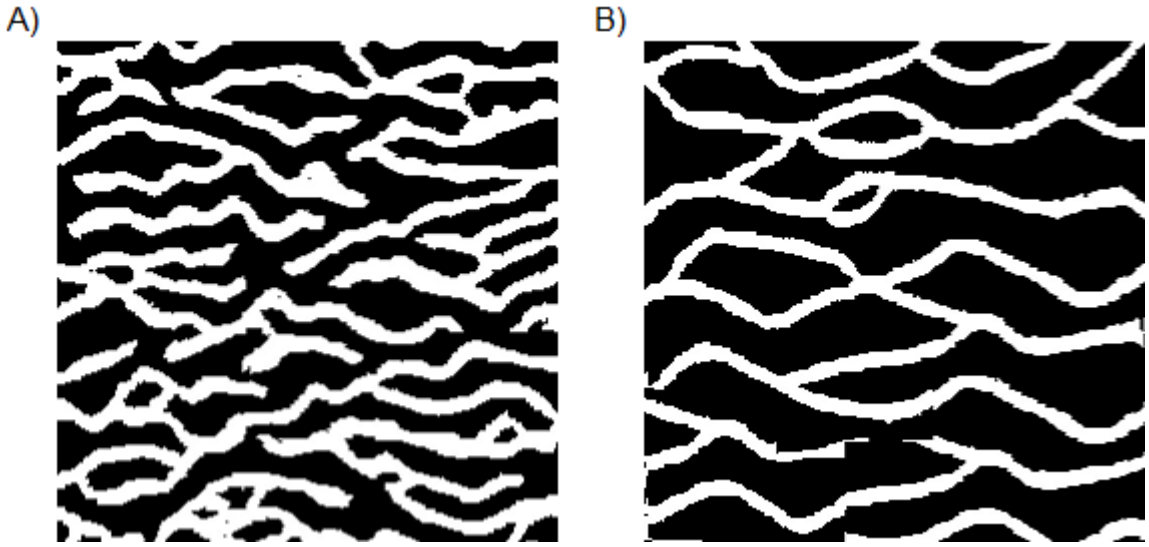

Figure 1.7: Two realizations of the Strebelle TI of 250 × 250: (A) setting template size to 10 × 10 and (B) setting template size set to 50 × 50.

should be as low as possible. Nevertheless, the quality of these realizations should also be good, in the sense that they should reproduce well the TI's spatial continuity. In addition, these simulations should have variability, so that a reasonable space of uncertainty is covered. These three components constitute the objectives that a MPS method should pursue.

This is a very sensitive problem and a central point when performing a geostatistical study. Therefore, this motivates the study and application of new algorithmic techniques to speed up the process, while also addressing the quality and variability requirements.

1.3 Objective

The objective of this work is to study and propose a pattern-based MPS method that generates a great number of realizations as fast as possible, with good quality and achieving a good variability.

For that, we first addressed the problem of searching for the most similar patterns when performing a realization. We have carefully studied one of the state-of-the-art methods regarding both computational time and quality, the MS-CCSIM method (Tahmasebi et al., 2014), and verified that it calculates convolutions via the Fast Fourier Transform (FFT) algorithm (Cooley & Tukey, 1965) to search for the most similar patterns to a given data event. In this sense, we proposed the use of compression techniques to accelerate the computation of convolutions. We have performed an in-depth investigation

of the potential of Run-Length Encoding (RLE) and Lempel-Ziv (LZ) based methods for efficiently calculating convolutions of patterns of a fixed size and a given image. Our first contribution consists in developing new methods and variants of existing ones for this purpose and providing (extensive) empirical evaluations of the proposed methodologies. Our fastest method outperforms a highly optimized implementation based on FFT for small patterns.

Our second and main contribution in this thesis is to propose LSHSIM, a new method that generates realizations faster than MS-CCSIM. The key innovations introduced by LSHSIM are the application of the *Locality Sensitive Hashing* (LSH) technique to filter patterns similar to a given data event and the use of the *Run-Length Encoding* (RLE) technique, to speed up the calculation of similarity between patterns when the image is categorical. Our experimental study suggests that our method produces realizations with similar quality in almost one order of magnitude faster than MS-CCSIM. Besides, LSHSIM also guarantees a good variability among the generated realizations.

1.4 Organization

This thesis is organized as follows: in Chapter 2, we present previous related works, following their evolution since the inception of MPS, and discuss in more details some methods which are part of the state of the art, specially the MS-CCSIM method. In Chapter 3, we give some background that is necessary for the understanding of our contributions, such as the concept of convolution, some compression techniques and the LSH technique. Later, in Chapter 4, we address our study with respect to the use of compression techniques to speed up the calculation of convolutions as a way of searching for the most similar patterns in a pattern-based MPS approach.

In Chapter 5, we introduce LSHSIM, describing each of its components and discussing our computational experiments, where we compare LSHSIM with MS-CCSIM regarding computational time, realization's quality and variability. Besides, we also study the impact of conditioning in LSHSIM and present a sensitivity analysis of the method concerning its parameters. Finally, in Chapter 6 we present our final considerations and suggested future works.

2
Related Work

In this chapter, we first present the related work regarding MPS. Then, we present references with respect to compression techniques for calculating convolutions.

2.1
Multiple-Point Geostatistics

The seminal work of Guardiano & Srivastava (1993) was the first one to propose the use of MPS in the ENESIM method, while trying to simulate a sandstone formed by alternating fine and coarser sediments. The simulations obtained were more accurate than the ones generated by previous techniques (Chilès & Delfiner, 2012). Despite its importance for the area, the method was computationally slow, because it had to scan the whole TI for each new pixel to be simulated.

In the SNESIM algorithm, Strebelle (2002) proposed the use of a search tree, containing all possible training data events of a given size, to speed up the computation. The tree was built scanning once the TI before performing the simulations. What is more, it required an intensive use of computer memory and it was restricted to categorical TIs.

The Direct Simulation method of Mariethoz et al. (2010) used a different strategy: to simulate a pixel, it performed a random path through a fraction of the TI and selected the first similar training event whose distance to the conditioning data event was smaller than some defined threshold. In this way, it avoided the drawback of SNESIM, since it did not require the storage of the training data events in a tree data structure.

Following a pixel by pixel simulation, these methods had a difficulty of reproducing the spatial continuity of the TI. This motivated the adoption of pattern-based approaches, which lowered the computational time and improved the quality of realizations, but introduced a new difficulty, the high dimension of the patterns. The FILTERSIM method (Zhang et al., 2006) proposed a clusterization based on image features, so as to cope with this issue. In its turn, SIMPAT (Arpat & Caers, 2007) indexed in a list all possible patterns in a TI.

Honarkhah & Caers (2010) extended both works and proposed the DISPAT method, in which they apply the Multidimensional Scaling (MDS) technique (Borg & Groenen, 2005), so as to cope with the high dimensionality of the patterns. Then, in this reduced dimension provided by MDS, the clusterization is an easier task and they use the K-Means algorithm for this purpose, after applying a kernel transformation to the data. Therefore, their results are better than the previous SIMPAT method, regarding both quality and CPU time.

In the CCSIM method, Tahmasebi et al. (2012) proposed the use of the cross-correlation distance (convolution) in association with a raster path, introducing thus the concept of overlap, which is the region shared by a data event with previous simulated patterns of a realization. As an example, for the data event shown in Figure 1.6, its non-blue values correspond to an overlap area of size 2. They also claimed that the adopted distance captures better the similarity between patterns and its calculation is performed in the spatial domain, i.e., applying a naive convolution directly from the formula. In this way, they were able to generate better simulations than previous methods. Concerning the conditioning, the method performs sequential subdivisions in the template size, so as to find a pattern honoring the hard data.

The work of Gardet et al. (2016) applied a K-Means technique to cluster patterns and thus accelerate its search. It also proposes the use of a wavelet decomposition to reduce the time required to compute distances, defining a similarity measure over the decomposed patterns. They compared their method with CCSIM and reported a wider variability, but a worse pattern reproduction.

Recently, Abdollahifard (2016) proposed the FPSIM method which explores two points: (i) a new path strategy that prioritizes data-events placed in the contour between the filled and empty regions of a realization; (ii) a search scheme that is based on the gradient vector of the central pixel of data-events. This search first compares this gradient vector with the gradient of each TI's pixel, in order to obtain a set of candidate patterns, and then performs a search in this set using the Euclidean distance. The authors claim to reduce the search space up to hundreds of times.

The search phase of LSHSIM, which will be explained in Chapter 5, resembles that of FPSIM in the sense that it first filters patterns that are likely to be similar to a given data event and then it looks for a good candidate in the filtered set. Besides, the reduction on the search space can be controlled by a parameter α. As an example, for the experiments with 2D categorical TIs, to be presented in Chapter 5, we use $\alpha = 0.5\%$, which reduces the original

space of patterns by a factor of at least 200 and, hence, is comparable to the reduction of hundreds of times reported in (Abdollahifard, 2016).

It is also worth mentioning that the evolution of methods belonging to MPS has followed the same path as in the computer vision area, where it is named as texture synthesis, such as described by the review of Mariethoz & Lefebvre (2014). Some concepts brought to the MPS area were previously proposed in the former. The main difference consists of the concept of hard data, which does not exist in texture synthesis methods.

Lastly, the book of Mariethoz & Caers (2014) was the first one solely dedicated to MPS, providing a survey of the area and explaining in details the principal techniques used by methods published until 2014. Besides, it also provides a library of training images (TrainingImagesLibrary, 2016), which we have adopted in the experiments of our research.

In the next few subsections, we describe and discuss the relation of our method with some of the state-of-the-art methods.

2.1.1 Review of the MS-CCSIM method

The MS-CCSIM (Tahmasebi et al., 2014) is an extension of the CCSIM method for categorical variables that introduces two new ideas that accelerate the search for a pattern and the convolution's calculation: (i) the use of a multi-scale approach, in which the TI is represented in increasingly different resolutions and so the search space of a query is reduced; and (ii) the calculation of the cross-correlation function in the frequency domain using the fast Fourier transform (FFT) (Cooley & Tukey, 1965). Figure 2.1 illustrates the multi-scale strategy, in which the FFT is only applied in the red dashed region of each resolution, starting from the coarsest one, searching for the most similar pattern to a given data event. In this sense, the method reduces its search space, because it never applies the FFT to the whole original TI, applying to coarser resolutions and later exploring only a fraction of the TI.

In addition to that, MS-CCSIM adopts a raster path, which starts from a corner of the realization and fills it line by line, horizontally or vertically, such as depicted by Figure 2.2. This strategy brings some problems when dealing with hard data. For this reason, the method employs the idea of a co-template, such as proposed by Parra & Ortiz (2011). It is a way of "looking ahead", trying to verify if there is some hard data lying ahead of the path. It selects then training patterns whose co-patterns satisfy these constraints.

Another important issue brought by this method was the approach to the patchiness problem, which typically brings discontinuities to generated

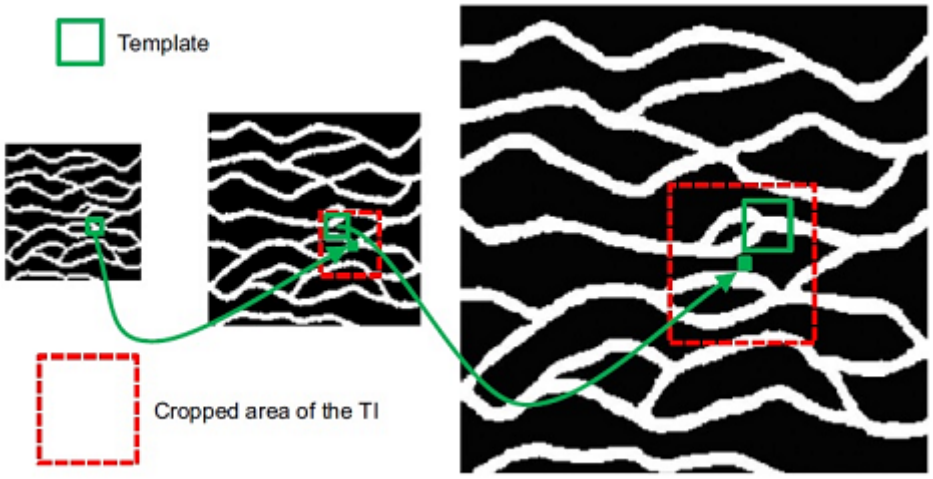

Figure 2.1: Multi-scale approach (Tahmasebi et al., 2014).

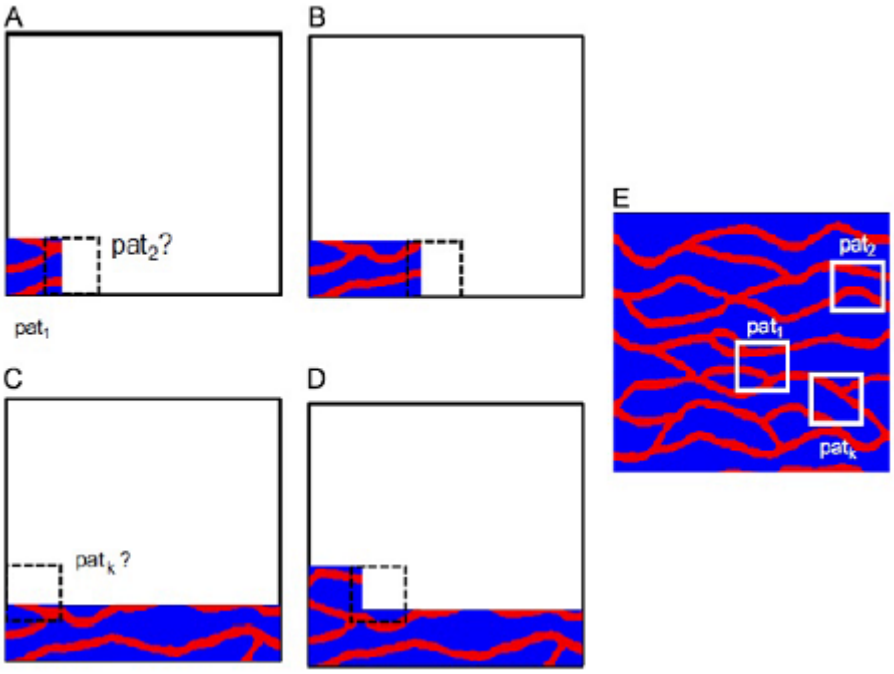

Figure 2.2: Raster path strategy (Tahmasebi et al., 2014).

realizations. Aiming to deal with this question, it applies the technique of minimum error boundary cut (MEBC), originally proposed in the Image Quilting method by Efros & Freeman (2001), which was tailored for the texture synthesis area. However, this approach has some limitations and this fact was later discussed and addressed by Tahmasebi & Sahimi (2016*a*), who applied a graph network formulation to this problem. Tahmasebi & Sahimi (2016*b*) described some of the advantanges and disadvantages of raster path algorithms, as well as other strategies for dealing with hard data, other than co-template. Mahmud et al. (2014) also worked on this issue, proposing an extension of the Image Quilting method to conditioning and to 3D images, while having other similar characteristics to the CCSIM method.

2.1.2 Review of the GOSIM method

GOSIM proposed by Yang et al. (2016) is an optimization-based method which uses the EM procedure to build a realization. It starts with a realization $R(0)$ obtained from a coarse representation of the Training Image and then it obtains a sequence of partial realizations. To obtain the realization $R(i+1)$ from the realization $R(i)$ it executes a E step and a M step. The E step consists of solving an approximate nearest neighbor problem (ANN) for each data event D in the realization $R(i)$. In other words, for each data event D, a similar pattern (hopefully the most similar one) in the training image shall be found.

To find these similar patterns, GOSIM runs the PatchMatch algorithm proposed in Barnes et al. (2009). PatchMatch is a randomized procedure that starts with a random guess of the most similar pattern in the TI for each data event D in the realization $R(i)$. Then, the algorithm refines the guess $G(D)$ for each D in $R(i)$ by taking account the current guess of the neighbors of D in $R(i)$ and by performing an exponential search on the neighborhood of $G(D)$. The procedure is expected to converge after a few iterations. By an iteration we mean the process of refining the current guess of the most similar pattern for each data event D in $R(i)$.

The PatchMatch does not apply, at least directly, to our proposal because the approximate nearest neighbor (ANN) problem LSHSIM has to solve is different than the one solved by GOSIM. In the E step of GOSIM, the set of patterns for which we want to find the most similar patterns in the TI is known beforehand – it is the set of patterns in the current realization $R(i)$. On the the other hand, in LSHSIM, the set of patterns for which we need to find the most similar patterns in the TI is not known beforehand because they

are dynamically created during the raster path simulation. In fact, the i-th pattern for which we need to solve the ANN depends on the solution of the ANN for the patterns that are adjacent to the i-th pattern in the simulation. This subtle but important difference prevents a direct use of the PatchMatch in LSHSIM.

We shall remark, however, that the LSH technique could be used, instead of PatchMatch, in the E step of GOSIM. Whether or not it would speed up the process might be a topic for future research.

2.1.3 Review of the fast template matching in transform domain based method

In (Abdollahifard & Nasiri, 2017), Abdollahifard and Nasiri proposed an approach to speed up template matching for MPS methods. The key idea consists of calculating the similarity between data events and TI's patterns using a low dimension approximation of them. To accomplish this goal, each pattern P of the TI is mapped into a new pattern P' via an orthonormal transformation (e.g. discrete cosine transform - DCT). Then, only the m most significant coefficients from P' are stored (the others are considered to be 0). The value of m shall be chosen to provide a significant reduction in the dimension of the TI patterns without losing much information. This approach, used in the preprocessing phase, has been widely used in the data compression community.

The search for a pattern to replace a data event D in the simulation grid has two steps: (i) D is mapped into a low dimension representation D_{low} using the same transformation employed for the TI patterns and (ii) an exhaustive search in the set of low dimension TI patterns is performed to find the most similar pattern to D_{low} with respect to Euclidean distance. The number of operations required in the search phase is proportional to Nm, where N is the number of patterns in the TI and m is the chosen number of coefficients.

The approaches of Abdollahifard & Nasiri (2017) and LSHSIM are quite different. The former does an exhaustive search in the set of low dimension patterns while the latter does an optimized search (RLE based) in a subset of the original patterns that are likely to be close to the data event. In common, both use ideas that were originated in the data compression community.

Moreover, LSHSIM has a theoretical guarantee of finding, with high probability, a pattern that is near to the given data event. As far as we know there are no theoretical guarantees available for the search proposed by Abdollahifard & Nasiri (2017). However, a theoretical guarantee could be achieved if a random projection (Leskovec et al., 2014) is used rather than a

DCT.

In fact, we understand that (Abdollahifard & Nasiri, 2017) is much more related with CCSIM (Tahmasebi et al., 2012) since both rely on efficiently calculating convolutions. If a fast Fourier transformation (FFT) is employed, as proposed in MS-CCSIM (Tahmasebi et al., 2014), CCSIM executes $\log k$ operations per pattern/data event comparisons, where k is the number of pixels of the data event. Thus, (Abdollahifard & Nasiri, 2017) is advantageous with respect to CCSIM, in terms of speed, if $m < \log k$ can be chosen. In terms of reproduction quality, CCSIM has the potential advantage of not losing information.

2.2 Compression Techniques for Calculating Convolutions

We have investigated the use of compression techniques to speed-up the calculation of convolution in the search for the most similar patterns.

Since convolutions arise in applications from different domains, a significant amount of research has been dedicated to develop efficient methods for its computation. A naive method directly obtained from definition calculates the convolution between a pattern P and a sequence S in $\mathcal{O}(|S||P|)$ time. By using the celebrated fast Fourier transform (FFT) this complexity can be improved to $\mathcal{O}(|S| \log |P|)$ time (Cooley & Tukey, 1965). Some methods were proposed to work on compressed data (Freschi & Bogliolo, 2010; Tanaka et al., 2013).

Motivated by applications in string matching, in (Freschi & Bogliolo, 2010), it is shown how to calculate the convolution between a pattern P and a sequence S compressed with the LZ78 (Ziv & Lempel, 1978). The method requires $\mathcal{O}(|S| + |P|N_S)$ time, where N_S is the number of factors of the LZ78 decomposition of S. Our first LZ based method, to be presented in Chapter 4 can be seen as an extension of this one for 2D structures. In contrast to ours, no empirical evaluation is presented in this paper.

In (Tanaka et al., 2013), it is discussed how to calculate the convolution between a pattern P and a sequence S represented by a straight line program (SLP) of length g. The relevance of SLP's is because the output of different compressing schemes can be seen as, or quickly transformed into, a SLP (Rytter, 2003). Two methods are presented by Tanaka et al. (2013). The most efficient one runs in $\mathcal{O}(\min\{|S| - \alpha, |P|g)\} \log |P|)$ time, where α is a measure of redundancy of the SLP's with respect to substrings of length m. Both methods output a data structure from which the i-th dot product (coordinate) of the convolution vector can be retrieved in $\mathcal{O}(\log |S|)$ time. Because the decomposition given by LZ78 can be seen as a SLP, the authors claim that

this method is better in terms of time complexity than the one proposed in (Freschi & Bogliolo, 2010). Again, no empirical evaluation is presented.

In (Simard et al., 1998), it is proposed a method for quickly approximating convolution's computation. The particular case of this method, for binary patterns/images, is exactly the calculation of a convolution on a run-length encoding representation. This paper presents some experiments comparing their approach with the naive method. In this thesis, we have carried a deeper investigation of the potential of run-length encoding and we also proposed and empirically explored non-trivial variations that yield to reasonable gains.

Lastly, we should also mention some researches on speeding up convolution evaluation for some specific domains (Werman, 2003; Hassanieh et al., 2012).

3
Background

We discuss in this chapter some fundamental concepts that are required to understand our work. We first define the concept of convolution and some compression techniques used in our work for accelerating convolutions' calculation. Moreover, we describe some graph theory concepts that we employed in this research. Then, we define what is a distance measure and explain some distances that are used by our method LSHSIM. Finally, we discuss how to address the problem of finding similar patterns using LSH, explaining the LSH scheme for both the Hamming similarity and the Euclidean distance. Those acquainted with these concepts may skip this chapter.

3.1
Convolution

The convolution of two sequences $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{m-1})$, where $n > m$, of real numbers can be defined as a sequence $C = (c_0, \ldots, c_{n-m})$ of real numbers, where

$$c_k = \sum_{i=0}^{m-1} a_{i+k} b_i, \quad \text{for } k = 0, \ldots, n-m. \tag{3-1}$$

The computation of convolutions arises in many applications from different areas as digital signal processing, image processing and string processing, among others (Tanaka et al., 2013).

We have explored compression techniques for the fast computation of convolutions, a topic that has been explored recently in the data compression community motivated by applications in string matching (Tanaka et al., 2013; Freschi & Bogliolo, 2010). This will be presented in Chapter 4.

3.2
Compression

The next subsections present two compression techniques used in our research.

3.2.1 Run-Length Encoding

The Run-Length Encoding (RLE) is a simple technique used for compressing sequences that have many repetitions among consecutive symbols. For this purpose, the method transforms each consecutive subsequence of the same symbol into a pair, where the first value represents the number of repetitions and the second one denotes the symbol. For example, the sequence 1110000011 is compressed by the RLE method to $(3, 1), (5, 0), (2, 1)$.

3.2.2 Lempel-Ziv

The Lempel-Ziv (LZ or LZ78) (Ziv & Lempel, 1978) is a method that scans the input sequence and breaks it into factors, progressively building a dictionary. It explores the internal repetitions of a sequence to compress it, so that each entry in its dictionary is a factor f, which is the concatenation of a previously encountered factor f' and a symbol v. The dictionary is previously initialized with all the symbols of the alphabet. Let $\mathbf{c}$ be the compressed sequence, which starts empty, and $\mathbf{u}$ be the uncompressed input sequence. The method searches for the longest factor (subsequence) $\mathbf{s}$ in the dictionary that matches $\mathbf{u}$. It then adds to $\mathbf{c}$ a pointer to the entry $\mathbf{s}$ in the dictionary, removes $\mathbf{s}$ from $\mathbf{u}$ and adds the concatenation of $\mathbf{s}$ and the next input symbol v to the dictionary. After that, it repeats the same steps to the remaining input $\mathbf{u}$.

For instance, the application of the LZ method to the sequence 010010001000 of alphabet $\Sigma = \{0, 1\}$ is as follows. The dictionary $\mathcal{D}$ is initialized with the symbols 0 and 1. The method starts scanning the input, finds the subsequence 0 already in $\mathcal{D}$ and adds its concatenation with the next symbol 1, i.e., the subsequence 01, to the dictionary $\mathcal{D}$, removing 0 from the input. The algorithm then finds 1 which is in $\mathcal{D}$ and adds 10 to it, removing 1 from the input. After that, LZ finds 0 which is already in $\mathcal{D}$, removes it from the input and adds 00 to $\mathcal{D}$. Finally, LZ finds 01 which is in $\mathcal{D}$, removes it from the input and adds 010 to the dictionary. LZ performs these steps until the end of the input sequence. The obtained segmentation in factors of the original sequence is: $0 - 1 - 0 - 01 - 00 - 010 - 00$, where the symbol '-' is used to separate factors. Table 3.1 describes the evolution of the dictionary $\mathcal{D}$ and the obtained factors as the input sequence is parsed step by step.

Input	**Dictionary $\mathcal{D}$**	**Factors**
010010001000	$\{0, 1\}$	
10010001000	$\{0, 1, 01\}$	0
0010001000	$\{0, 1, 01, 10\}$	$0 - 1$
010001000	$\{0, 1, 01, 10, 00\}$	$0 - 1 - 0$
0001000	$\{0, 1, 01, 10, 00, 010\}$	$0 - 1 - 0 - 01$
01000	$\{0, 1, 01, 10, 00, 010, 000\}$	$0 - 1 - 0 - 01 - 00$
00	$\{0, 1, 01, 10, 00, 010, 000, 0100\}$	$0 - 1 - 0 - 01 - 00 - 010$
	$\{0, 1, 01, 10, 00, 010, 000, 0100\}$	$0 - 1 - 0 - 01 - 00 - 010 - 00$

Table 3.1: Example of application of the LZ method to a given input sequence.

3.3 Some Concepts of Graph Theory

For a multigraph $G = (V, E)$, that is, a graph with repetitions of edges allowed, an Eulerian path (or trail) is a path that visits every edge of G exactly once. For example, the graph of Figure 3.1 has an Eulerian path, which corresponds to: $v_4, v_5, v_2, v_4, v_1, v_2, v_3, v_6, v_5, v_6$.

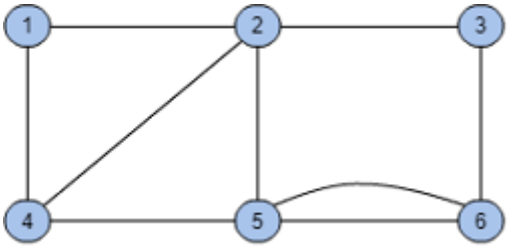

Figure 3.1: A multigraph with an Eulerian path.

The next definitions assume a given weighted graph $G = (V, E)$, where there exists a weight function $w : E \to \mathbb{R}^+$.

A minimum spanning tree (MST) T of G is a connected and acyclic subgraph of G containing all its vertices and such that the $\sum_{e \in T} w(e)$ is minimum. Figure 3.2 illustrates a graph (A) and its minimum spanning tree (B), where the edges which are part of the MST are highlighted in red.

A Hamiltonian path $H = v_1, v_2, \ldots, v_n$ in G is a simple path that passes through all vertices $v \in V$ exactly once. A Hamiltonian path with low cost H is one such that $\sum_{e \in H} w(e)$ is minimum among all possible Hamiltonian paths. Figure 3.3 (A) exhibits a Hamiltonian path for the example graph of Figure 3.2 (A), while Figure 3.3 (B) shows a Hamiltonian path with low cost. For both (A) and (B), the edges that are part of the Hamiltonian path are highlighted in red. An important known result is that if the edges of G satisfy the triangle inequality, i.e., for every $u, v, x \in V$ it is true that $w(uv) \leq w(ux) + w(xv)$,

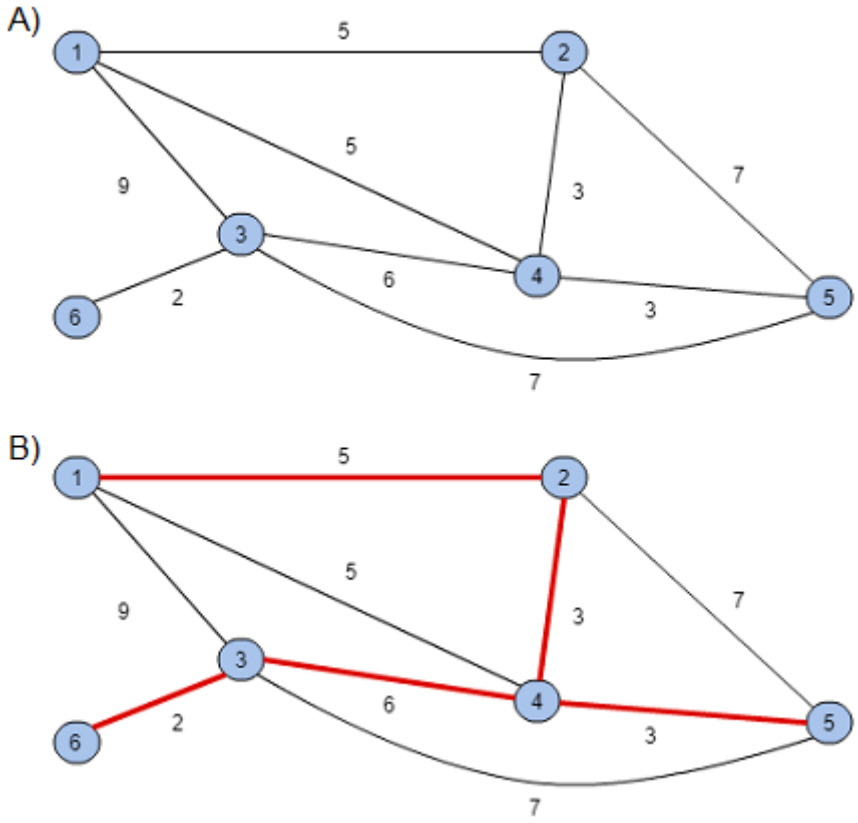

Figure 3.2: A weighted graph (A) and its minimum spanning tree (B).

there exists a 1.5-approximation in polynomial time algorithm, which is known as Christofides algorithm (Vazirani, 2001). We explore this fact in Chapter 4.

A matching in G is a set of edges M without common vertices. A matching M is perfect if it matches all vertices of G. Typically, for weighted graphs, we are interested in a minimum weighted perfect matching M, i.e., a perfect matching such that $\sum_{e \in M} w(e)$ is minimum among all possible perfect matchings of G. In this sense, Figure 3.4 illustrates a minimum weighted perfect matching for the example graph of Figure 3.2 (A), where the selected edges are highlighted in red.

3.4 Distance Measures

A distance measure is a function $d(x, y)$ that takes two points in a space as arguments and produces a real number, satisfying the following axioms (Leskovec et al., 2014):

1. $d(x, y) \geq 0$ (no negative distances)
2. $d(x, y) = 0$ iff $x = y$ (distances are positive, except for the distance from a point to itself)

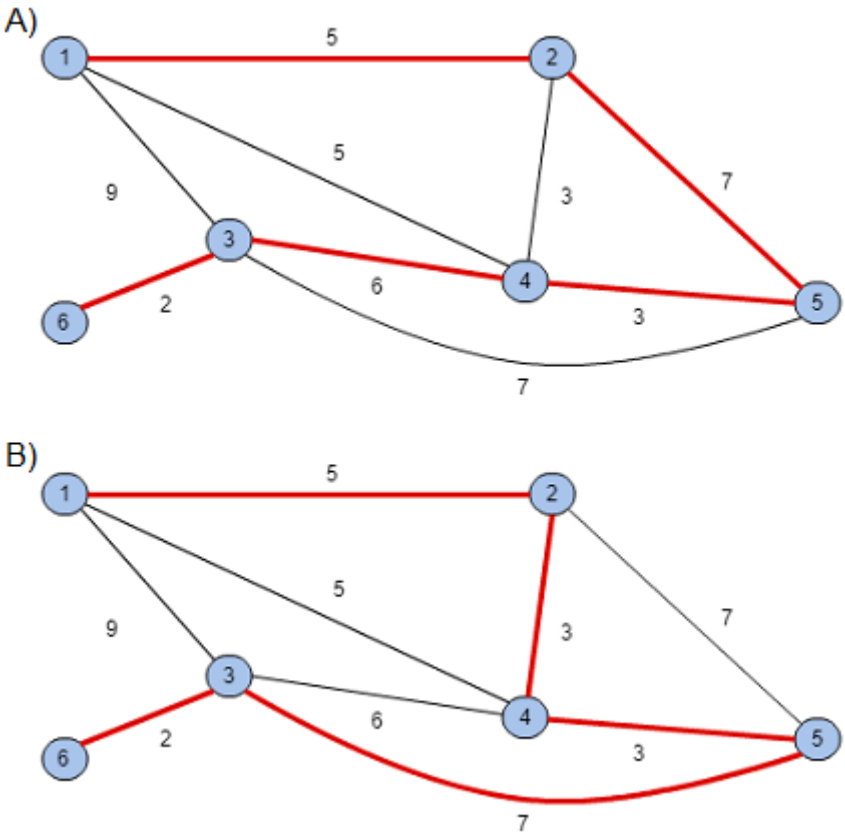

Figure 3.3: A Hamiltonian path (A) and a Hamiltonian path with low cost (B).

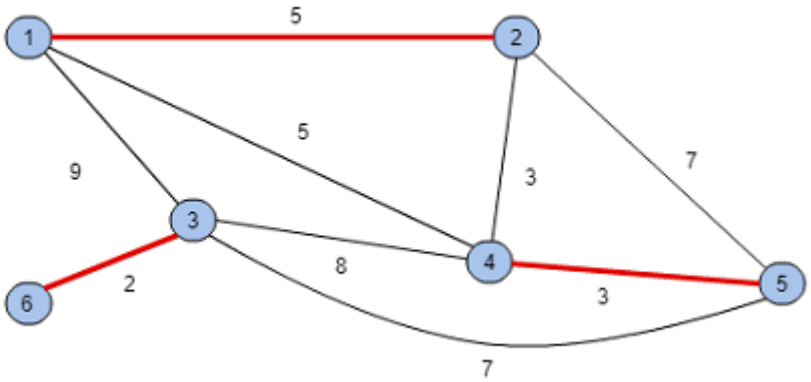

Figure 3.4: A minimum weighted perfect matching.

3. $d(x, y) = d(y, x)$ (symmetry)

4. $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

In the next subsections, we review some distance measures that are used throughout this work.

3.4.1 L_p-Norm Distance

The L_p-norm, for $p \geq 1$, of a vector $x = (x_1, \ldots, x_n)$ is given by:

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Then, the L_p-norm distance, also called Minkowski distance, between two vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ is defined as:

$$||x - y||_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

For $p = 1$, the 1-norm distance corresponds to the Manhattan distance, also called taxicab distance. For two vectors x, y in an n-dimensional real vector space, such that $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, the Manhattan distance is given by:

$$d(x, y) = ||x - y||_1 = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n|$$

$$= \sum_{i=1}^{n} |x_i - y_i|$$

For $p = 2$, we have the Euclidean distance, which is the distance measure between two vectors in an n-dimensional Euclidean space. Given $x, y \in \mathbb{R}^n$, such that $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, the Euclidean distance is defined as:

$$d(x, y) = ||x - y||_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.4.2 Hamming Distance

The Hamming distance was first proposed by Hamming (1950) in the information theory context. It is a natural way to measure distance/similarity among categorical data, specially for binary strings. Given two vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, the Hamming distance is defined as the ratio between the number of coordinates in which x and y are different and n. Conversely, the Hamming similarity corresponds to the ratio between the number of coordinates where x and y match and n. For example, if $x = (a, b, b)$ and $y = (a, c, b)$, then $HammingDistance(x, y) = 1/3$ and $HammingSimilarity(x, y) = 2/3$.

3.5 Locality Sensitive Hashing

As mentioned in Chapter 2, one of the challenges to implement the pattern-based approach is the high dimensionality of data. To address this issue, we propose the application of the so called *Locality Sensitive Hashing* (LSH).

In order to explain the technique, we first recall that a hash table is a data structure that implements an associative array: given an object x, a hash function $\boldsymbol{h}(\cdot)$ is used to determine the position in the structure/array where we can find information about x (see Cormen et al. (2009)).

The LSH was first proposed by Indyk & Motwani (1998) and Gionis et al. (1999). We first explain the definition of LSH for similarity measures. Given a set of elements S and a set of buckets B, a family $\mathcal{H}$ of functions $h : S \to B$, together with a probability distribution $\mathcal{D}$ over the functions in $\mathcal{H}$, is a LSH for a similarity measure $s(\cdot, \cdot)$ if, for any $x, y \in S$, we have

$$Pr_{\mathcal{H}}[h(x) = h(y)] = s(x, y),$$

where the probability is taken according to the distribution $\mathcal{D}$. This way similar elements have a large probability to be assigned to the same bucket while non-similar ones have a small probability.

We now present the definition of LSH for distance measures. Given S and B as before, and c, R, p_1, p_2 real numbers, a family $\mathcal{H}$ of functions $h : S \to B$ is called (R, cR, p_1, p_2)-sensitive for a distance measure $d(\cdot, \cdot)$ if, for any $x, y \in S$, we have

- if $d(x, y) \leq R$ then $Pr_{\mathcal{H}}[h(x) = h(y)] \geq p_1$
- if $d(x, y) \geq cR$ then $Pr_{\mathcal{H}}[h(x) = h(y)] \leq p_2$

where the probabilities p_1, p_2 are considered with respect to the random choice of a function h from the family $\mathcal{H}$. A family is useful when $p_1 > p_2$.

In words, the functions $h \in \mathcal{H}$ associate each element of S with buckets $b \in B$, depending on their distance. If x and y are close, they will be put in the same bucket b with probability at least p_1. On the other hand, if they are far apart, they will be given the same bucket with probability at most p_2. Figure 3.5 illustrates this concept, where the close points are hashed to the same bucket while the distant one is assigned to a different bucket.

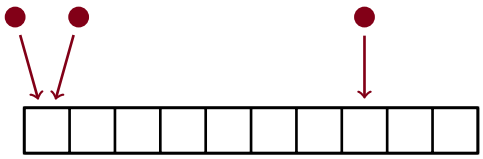

Figure 3.5: Illustration of the LSH concept for distance measures.

One of the main applications of LSH is as a tool to address the *Approximate Nearest Neighbor* (ANN) problem (Gionis et al., 1999). This problem admits the following formulation:

Input. A set of points S, a query point q and a value $\epsilon > 0$.

Output. A point $p \in S$ such that $s(q, p) \geq (1 - \epsilon)s(q, S)$, where $s(q, S)$ is the similarity of q to its most similar point in S.

The ANN problem naturally arises in the context of pattern based simulation since a key operation in this kind of simulation consists of finding patterns that are (very) similar to a given data event.

To address the ANN problem, via the LSH approach, we have two phases:

– *Preprocessing Phase.* In this phase K hash functions are randomly selected from $\mathcal{H}$ using the probability distribution $\mathcal{D}$. Let $h_1, h_2 \ldots h_K$ be the chosen functions and $\boldsymbol{h} = h_1 h_2 \ldots h_K$ be the function obtained by the concatenation of these functions. Then, $\boldsymbol{h}$ is used to build a hash table that maps each $x \in S$ into a bucket $\boldsymbol{h}(x) \in B$. This procedure is repeated L times so that we end up with L hash tables, each of them storing all the elements in S.

– *Search Phase.* Given a point q, we find its bucket/position in each one of the L hash tables using the hash function $\boldsymbol{h}$. Let C_q be the set of points that are mapped to the same bucket of q in at least one of the L hash tables. Then, we can either return an arbitrarily chosen point in C_q or return the most similar element to q among those in C_q. The latter possibility increases the chance of returning patterns that are more

similar to q but it is more expensive in terms of computational time. Another possibility in the search phase is to return the most similar point after inspecting some fraction of the points in C_q. This way we trade-off between the quality of the returned point and the computational time.

By choosing the values of K and L properly it is possible to guarantee a high probability of returning a point that is among the most similar to the query q with respect to the similarity $s(\cdot, \cdot)$.

3.5.1 LSH for Hamming Similarity

Let S be a set of points in a n-dimensional space. In addition, for $i = 1, \ldots, n$, let $h_i : S \mapsto R$ be a function that maps each $x \in S$ into x_i, which is the i-th coordinate of x. A well known result in the theory of LSH states that the family $\mathcal{H} = \{h_1, \ldots, h_n)$, together with a uniform distribution $\mathcal{D}$ over $\mathcal{H}$, is a LSH scheme for the Hamming similarity. This scheme is used by LSHSIM for categorical images, so as to filter patterns that are similar to a given data event.

3.5.2 LSH for Euclidean Distance

For continuous data, since the Hamming similarity does not apply, we employ the Euclidean distance, which has been used in pattern-based methods since the work of Arpat & Caers (2007).

In the LSH scheme for the Euclidean distance, each hash function h in the family $\mathcal{H}$ is associated with a random line in the n-dimensional space. Given a constant a, this line is divided into segments of length a, which correspond to the buckets. Each $x \in S$ is then projected onto the line and hashed to the bucket concerning the segment in which it lies. Figure 3.6 illustrates this concept, where two points at a given distance d, making an angle θ with the random line, are projected onto different segments, thus hashed to different buckets.

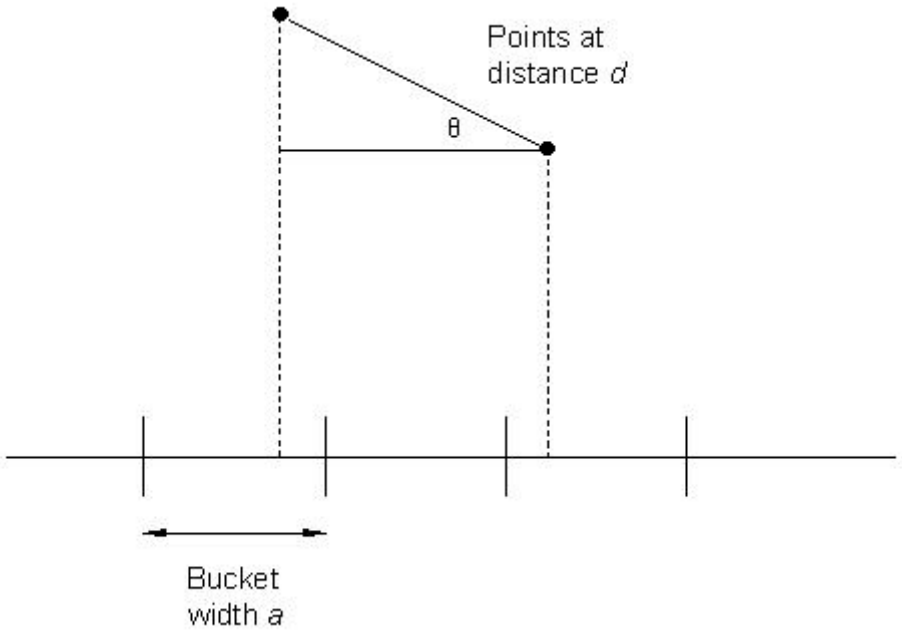

Figure 3.6: Illustration of the LSH scheme for Euclidean distance (adapted from Leskovec et al. (2014)).

4
Compression Techniques for Computing Convolutions

In this chapter, we present our study regarding the use of compression techniques for efficiently computing convolutions as a way of searching for the most similar patterns in a pattern-based MPS approach.

In a pattern-based method, a fundamental step is the search in the TI for the most similar patterns to a given data event, so that one of these patterns is chosen and pasted in realization. This process was illustrated by Figure 1.6. This motivated us to define the following problem:

Problem 4.0.1 *Given an image I and a list of patterns $P_1, \ldots, P_n$, all of them with the same dimension, the problem consists of computing the convolution between I and each of these patterns, as fast as possible, with the constraint that pattern P_i only becomes available after the convolution between I and P_{i-1} has been computed.*

Our application is mapped on the above problem as follows: I corresponds to the training image, the patterns $P_1, \ldots, P_n$ correspond to the data events and the constraint on P_i is because P_i is extracted from the realization after pasting the chosen pattern obtained by computing the convolution between P_{i-1} and I.

In our setting, n is usually very large so that it is worth to preprocess the TI with the goal of speeding up the convolution time. Another important aspect of our domain is that, for categorical TIs, each pixel may assume just a few values (e.g. 2 or 3 values, called facies) so that these images are expected to attain high compression rates and as a consequence they are suitable for methods that make use of compression techniques for calculating convolutions.

We have performed an investigation of the potential of Run-Length Encoding (RLE) and Lempel-Ziv (LZ) based methods for solving Problem 4.0.1. Besides, we have also provided an experimental study of these methodologies.

4.1
Methods

In this section, we describe our methods for dealing with 2D TIs, but they are also extensible to 3D TIs. First, we need to introduce some notation.

Let $I = \{I(i,j)|0 \leq i < w \text{ and } 0 \leq j < h\}$ be a 2D image of dimension $w \times h$, where $I(i,j)$ is the value of the pixel located at position (i,j) of I. Moreover, let $P = \{P(i,j)|0 \leq i < m \text{ and } 0 \leq j < m\}$ be a square pattern [1] of dimension $m \times m$, where $P(i,j)$ is the value of the pixel located at position (i,j) of P. Furthermore, let $B_{i,j}$ be the block (subimage) of I of dimension $m \times m$ whose left top pixel has indexes i and j, that is, $B_{i,j} = \{I(p,q)|i \leq p < i+m \text{ and } j \leq q < j+m\}$. Our goal is to compute the matrix $C = \{C(i,j)|0 \leq i < w-m+1 \text{ and } 0 \leq j < h-m+1\}$, where

$$C(i,j) = \sum_{p=0}^{m-1} \sum_{q=0}^{m-1} P(p,q) \cdot I(p+i, q+j). \tag{4-1}$$

In order to explain the methods it will be convenient to set a linear order among the blocks of image I, among the entries of matrix C and also among the pixels of pattern P. Let $NumBlocks = (w-m+1) \times (h-m+1)$ be the size of the vector C and let $w' = w - m$. Thus, $B_k = B_{\lfloor k/w' \rfloor, k \bmod w'}$ and $C(k) = C(\lfloor k/w' \rfloor, k \bmod w')$, for $k = 0, \ldots, NumBlocks - 1$ and $P(k) = P(\lfloor k/m \rfloor, k \bmod m)$, for $k = 0, \ldots, m^2 - 1$. Figure 4.1 depicts an example training image of size 4×4 and its blocks of size 3×3 denoted by its position in the linear order, each one depicted by the red dashed region.

Our methods are split into two phases: the preprocessing phase and the convolution phase. In the preprocessing phase, we produce a compressed image $I^{\mathcal{M}}$, which is the concatenation of the compressed blocks $\mathcal{B}_0^{\mathcal{M}}, \ldots, \mathcal{B}_{NumBlocks-1}^{\mathcal{M}}$, where $\mathcal{B}_k^{\mathcal{M}}$ is the compressed block obtained by compressing B_k through method $\mathcal{M}$. Whenever the context is clear we drop $\mathcal{M}$ from $\mathcal{B}_k^{\mathcal{M}}$. Note that the compressed image may be considerably larger than the original one but this is not necessarily a problem since our main focus is to minimize convolution time.

In the convolution phase, we compute the convolution C between each pattern P and the compressed image $I^{\mathcal{M}}$ by calculating the dot product between P and each compressed block $\mathcal{B}_k^{\mathcal{M}}$.

4.1.1 RLE based convolution

To motivate this approach let us consider the computation of the dot product between a sequence of run lengths $R = \{(3,1),(2,2),(4,1),(1,0)\}$, where the first value is the count and the second one is the data, and the pattern $P = (1,2,1,2,2,1,1,0,0,1)$, over the alphabet $\Sigma = \{0,1,2\}$. Let $S[i,j]$

[1] Our methods naturally extend to other shapes but for the sake of a clean presentation we focus on squares.

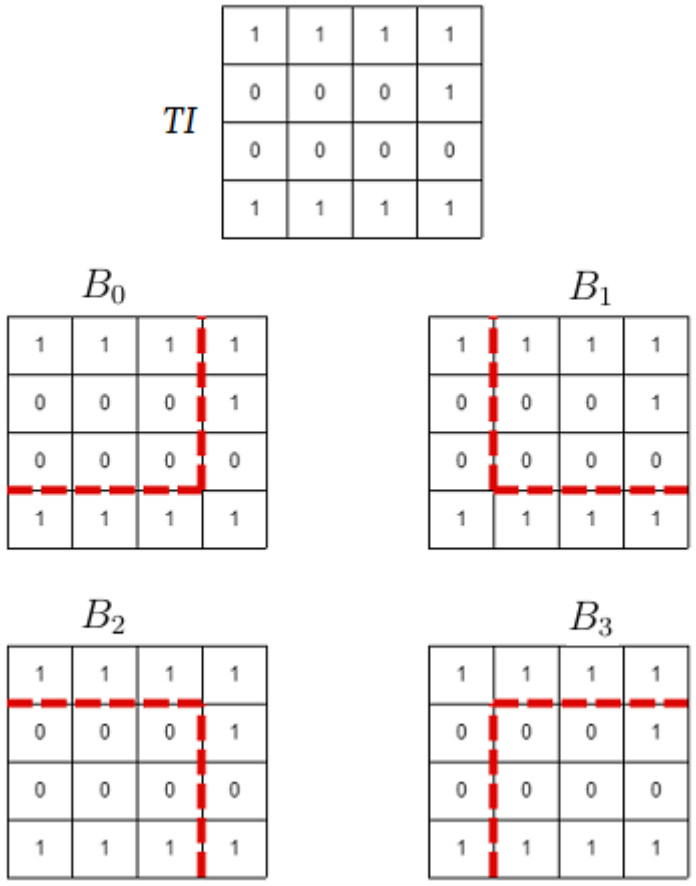

Figure 4.1: An example training image and its blocks of size 3×3 denoted by its position in the linear order.

be the sum of the elements of the subsequence of P that starts at ith position and ends at the jth position. Then,

$$R \cdot P = S[0,2] \times 1 + S[3,4] \times 2 + S[5,8] \times 1 + S[9,9] \times 0$$

Note that $S[i,j] = S[0,j] - S[0,i-1]$ and that all $S[0,i]$'s can be calculated through a single pass over P.

This example illustrates the fact that the dot product can be calculated in time proportional to $|R| + |P|$. The method described below relies on this simple idea.

Preprocessing Phase. For our simplest RLE based method, the block $\mathcal{B}_k$ is the sequence of run lengths obtained by scanning the block B_k line by line continuously. We use $\mathcal{B}_k(i).\texttt{count}$ and $\mathcal{B}_k(i).\texttt{value}$ to denote the counter and the value associated with the i-th run length of block $\mathcal{B}_k$, respectively. For example, Figure 4.2 exhibits the compressed blocks $\mathcal{B}_k$ using the RLE method following this continuous horizontal scan for the blocks B_k of size 3×3 of Figure 4.1, where, for each run length, the first value represents the counter

and the second one represents the value.

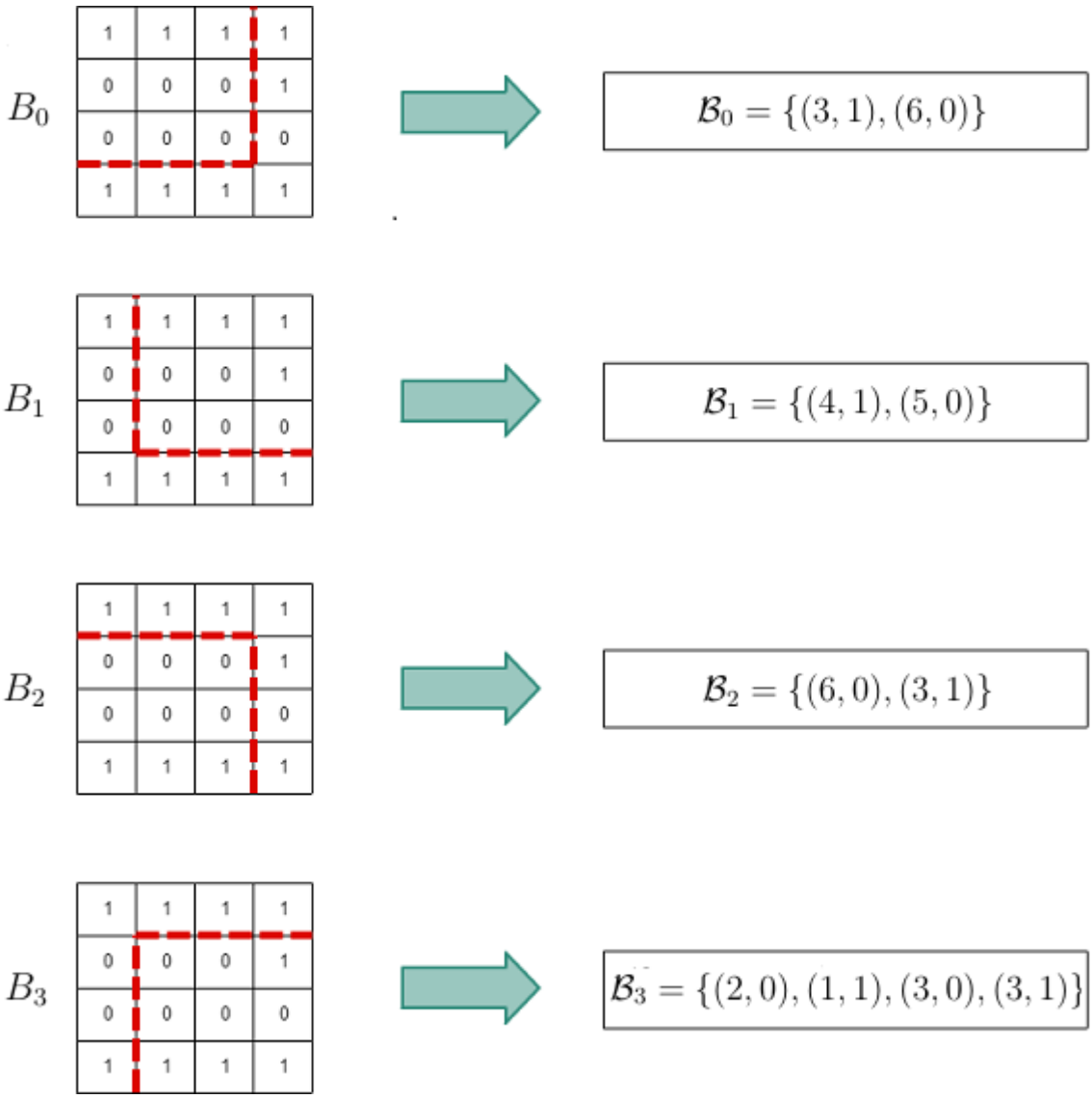

Figure 4.2: Compressed blocks $\mathcal{B}_k$ obtained with RLE method to blocks B_k of Figure 4.1.

Convolution Phase. To compute the convolution matrix C, we first preprocess pattern P in $\mathcal{O}(m^2)$ time so that each sum $S[i,j] = \sum_{\ell=i}^{j} P(\ell)$ can be computed in constant time. Then, for each compressed block $\mathcal{B}_k$ we execute the pseudocode in Algorithm 4.1.1 that runs in $\mathcal{O}(|\mathcal{B}_k|)$ time and performs few operations per run length processed.

Optimizing the scanning order. In the method described above, each compressed block $\mathcal{B}_k$ is obtained by scanning B_k line by line. However, this is not necessarily the best order to scan the blocks. Figure 4.3 shows two images, the left one is suitable for a horizontal scan of the blocks while the right one

Algorithm 4.1.1: Pseudocode for RLE Convolution Phase

Result: Convolution matrix entry $C(k)$

RLEConvPhase (P, $\mathcal{B}_k$)
  $\texttt{off} \leftarrow 0$
  $C(k) \leftarrow 0$
  for $i = 0, \ldots, m^2 - 1$ **do**
    Compute $S[0, i]$
  end for
  for $i = 0, \ldots, |\mathcal{B}_k| - 1$ **do**
    $C(k) \leftarrow C(k) + \mathcal{B}_k(i).\texttt{value} \times S[\texttt{off}, \texttt{off} + \mathcal{B}_k(i).\texttt{count} - 1]$
    $\texttt{off} \leftarrow \texttt{off} + \mathcal{B}_k(i).\texttt{count}$
  end for
return $C(k)$

benefits more from a diagonal scan. Given that the complexity of the proposed method is linear on the total number of run lengths of the compressed blocks, a natural question that arises is: *Which order of scanning minimizes the total number of run lengths of the compressed blocks?*

Figure 4.3: The left training image is better compressed using a horizontal scan while the right one is better compressed using a diagonal scan.

This question can be formulated as a graph theoretical problem. In fact, let $G = (V, E)$ be a weighted complete undirected graph, where $V = \{(i, j) | 0 \leq i < m \text{ and } 0 \leq j < m\}$. Let (i_1, j_1) and (i_2, j_2) be two vertices in V that satisfy either $i_1 < i_2$ or $(i_1 = i_2)$ and $(j_1 < j_2)$. The weight $w(e)$ of the edge e that connects (i_1, j_1) to (i_2, j_2) is given by:

$$|\{B_k | 0 \leq k < NumBlocks \text{ and } B_k(i_1, j_1) \neq B_k(i_2, j_2)\}|$$

In words, $w(e)$ is given by the number of blocks in I such that the pixels located in positions (i_1, j_1) and (i_2, j_2) in these blocks have different values. The following lemma presents important properties about the graph G.

Lemma 1 *Let H be a Hamiltonian path in G and let cost(H) be the sum of*

the weights of the edges of H. Then,

$$cost(H) + NumBlocks = \sum_{k=0}^{NumBlocks-1} |\mathcal{B}_k^H|,$$

where $|\mathcal{B}_k^H|$ is the number of run lengths of the block obtained when B_k is compressed with RLE following the scanning order H. Furthermore, the weights of graph G satisfy the triangle inequality.

Proof: Given a RLE compression $\mathcal{B}_k$ of a block B_k, the number of run lengths obtained by $\mathcal{B}_k$ corresponds to the number of times adjacent pixels in the scanning order performed in B_k have different values, plus 1. For instance, the sequence 0001100 scanned from left to right has two adjacent symbols with different values and thus three run lengths in its RLE compression: $\{(3,0),(2,1),(2,0)\}$. A Hamiltonian path H defines a scanning order for visiting all the pixels of the image's blocks, and the cost $w(e)$ of each edge $e \in H$ counts the number of times the vertices (pixels) of e have different values for all the blocks. Then, $cost(H)$ gives this number for all adjacent vertices (pixels) of the path. Hence, the number of run lengths for compressing all the image's blocks following H corresponds to $cost(H)$ plus $NumBlocks$ (one for each block).

We now prove that the graph G satisfies the triangle inequality. For this purpose, it suffices to show that every block B_k satisfies the inequality and then as a consequence the sum for all blocks will also satisfy. For each block B_k, given vertices (pixels) $v_i, v_j, v_p \in V$, it must hold that

$$\text{diff}(v_i, v_j) \leq \text{diff}(v_i, v_p) + \text{diff}(v_p, v_j)$$

where $\text{diff}(v_i, v_j)$ is equal to 1 if the vertices (pixels) v_i and v_j have the same value in B_k and 0 otherwise. In this sense, we analyse the two possibilities:

- $\text{diff}(v_i, v_j) = 0$. In this case, as the right hand side of the inequality cannot sum less than 0 by definition, the inequality holds;
- $\text{diff}(v_i, v_j) = 1$. In this case, since v_i and v_j have different values, either $\text{diff}(v_i, v_p) = 1$, or $\text{diff}(v_p, v_j) = 1$, or both, since v_p cannot have the same value as v_i and v_j simultaneously. This makes the right hand side sum at least 1 and the inequality also holds.

Therefore, each block B_k satisfies the triangle inequality. ■

The above lemma motivates the scan of each block following the order induced by a Hamiltonian path with low cost. The problem of finding the

minimum cost Hamiltonian path in graphs whose weights satisfy the triangle inequality admits a 1.5-approximation in polynomial time through Christofides algorithm (Vazirani, 2001). First, this algorithm constructs a minimum spanning tree T for G. Next, it calculates a minimum weighted perfect matching M^* for $G[O]$, the subgraph of G induced the set of vertices O with odd degree in T. Finally, it obtains a Hamiltonian path from an Eulerian path in the graph induced by the edges in $T \cup M^*$. Because the computation of the optimal matching is costly, we replaced it with a simple greedy heuristic that constructs a matching M for $G[O]$ by iteratively selecting an unmatched node, say u, and adding an edge uv to M, where v is the unmatched node which is closest to u in terms of Manhattan distance.

In addition, to reduce the computational time, we do not work with the graph G but with a subgraph of G, namely $G_d = (V, E_d)$, such that (i_1, j_1), $(i_2, j_2) \in V$ are connected by an edge in G_d if and only if

$$|i_1 - i_2| + |j_1 - j_2| \leq d$$

In words, two vertices are connected by an edge if and only if their Manhattan distance (see Chapter 3) is less than d.

The motivation for this selection is to keep just the edges that correspond to pixels that are close in the image I and, as a consequence, are more likely to have the same value. Although the graph G_d does not satisfy the triangle inequality, this is not a problem, since we can get the same approximation as before with respect to the optimal solution to G_d, but using edges that are not in G_d. Figure 4.4 exhibits the graph G_d obtained for the example binary TI and its 4 blocks of size 3×3 in Figure 4.1, using $d = 2$. For instance, the edge e_1 between vertices $(0, 0)$ and $(0, 1)$ has $w(e_1) = 0$ since their pixels have the same value in the 4 blocks. Conversely, the edge e_2 between vertices $(0, 0)$ and $(2, 0)$ has $w(e_2) = 4$ since their pixels have different values in all the 4 blocks. Lastly, a hamiltonian path with low cost is highlighted in red.

The graph G_d has m^2 vertices and $\mathcal{O}(m^2 d^2)$ edges. The Hamiltonian path can be computed in $\mathcal{O}((w-m)(h-m)d^2 + m^2 d^2(w+h) + m^4)$ time, where the term $((w-m)(h-m)d^2 + m^2 d^2(w+h))$ is due to the time required to calculate the weights of the edges and the term m^4 is due to the matching computation.

Finally, we shall note that to use this approach, in the convolution phase, we have to reorder the pattern under consideration according to the order induced by the Hamiltonian path. This can be done in $\mathcal{O}(m^2)$ time.

Multiple Scanning Orders. In the two methods presented so far the same scanning order is employed for all blocks. A natural idea is to consider different

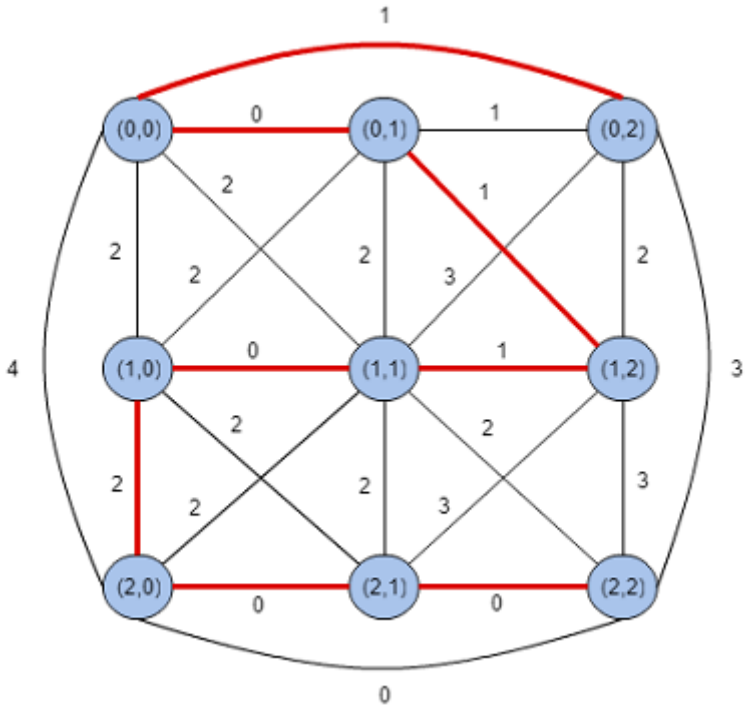

Figure 4.4: Graph G_2 obtained for the example of Figure 4.1 and a hamiltonian path with low cost highlighted in red.

scanning orders for different blocks. In fact, if we have k different orders available we could use the most suitable one for each block. However, this approach would have to be used with parsimony because at the convolution phase we would have to pay $\mathcal{O}(km^2)$ to reorder the current pattern according to each of these k orders.

We could define a combinatorial optimization problem that asks for the set of k scanning orders that yield to the most compressed image. Although we do not carry on a deep investigation on this problem, we perform a preliminary test on its potential by considering two natural orders: a continuous horizontal scan and a continuous vertical, which are illustrated by Figure 4.5 (A) and (B), respectively, for a pattern of size 4×4. The results are reported in Section 4.2.

4.1.2 Lempel-Ziv based convolution

The LZ based methods decompose each block B_k into a sequence of factors and then they calculate the dot product between the given pattern and the compressed block by adding the contributions of the dot products between the factors and the corresponding subsequences of the pattern. Let $P[i, j]$ be the subsequence of P that starts at position i and ends at position j. The

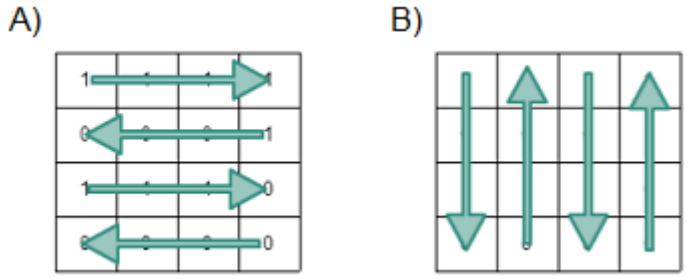

Figure 4.5: A continuous horizontal scan (A) and a continuous vertical scan (B).

key property behind the efficiency of these methods is that each factor f, by construction, is a concatenation of a factor f' and a value v so that dot product between a subsequence $P[i, j]$ and f can be calculated in $\mathcal{O}(1)$ time if the dot product between $P[i, j-1]$ and f' is known.

Our method employs a data structure $\mathcal{D}$ that consists of a set of m^2 dictionaries. Each entry of the i-th dictionary $\mathcal{D}(i)$ corresponds to a subsequence that begins at position i of some block B_k. Let e be an entry of $\mathcal{D}(i)$ and let $\mathbf{s}$ be the sequence associated with e. The entry e has four fields: e.`value`, which is the value of the last pixel of the sequence $\mathbf{s}$; e.`len`, the length of $\mathbf{s}$; e.`parent`, a pointer to the entry in $\mathcal{D}(i)$ associated with the prefix of $\mathbf{s}$ of length $|\mathbf{s}| - 1$; e.`DotProduct`, the dot product between $\mathbf{s}$ and the subsequence of the pattern under consideration that starts at position i and has length $|\mathbf{s}|$. All these fields but `DotProduct` are filled in the preprocessing phase.

Preprocessing Phase. At this phase, we compress block B_0 then block B_1, and so on, until block $B_{NumBlocks-1}$. We parse B_k as follows: we keep a pointer `off`, initially at position 0, for the beginning of the next subsequence of B_k to be parsed. Then, we look for the largest subsequence of B_k that starts at position `off` of B_k and matches an entry of $\mathcal{D}$(`off`). Let e be such an entry and let $\mathbf{s}$ be the matched subsequence. We add to the compressed block $\mathcal{B}_k$ a pointer to entry e and, if the length of $\mathbf{s}$ does not exceed $m^2 -$ `off`, we also add a new entry, say e', to $\mathcal{D}$(`off`). The fields of e' are filled as follows: e'.`parent` $= e$, e'.`len` $= e$.`len` $+ 1$ and e'.`value` is filled with the value of the pixel that succeeds $\mathbf{s}$ in B_k. Finally, we update the pointer `off` to `off` $+ |\mathbf{s}|$. Figure 4.6 exhibits the compressed blocks $\mathcal{B}_k$ using the LZ method for the blocks B_k of size 3×3 of Figure 4.1, where the symbol '-' is used to separate the factors.

Convolution Phase. We compute the convolution by scanning each compressed block $\mathcal{B}_k$ according to the pseudocode presented in Algorithm 4.1.2.

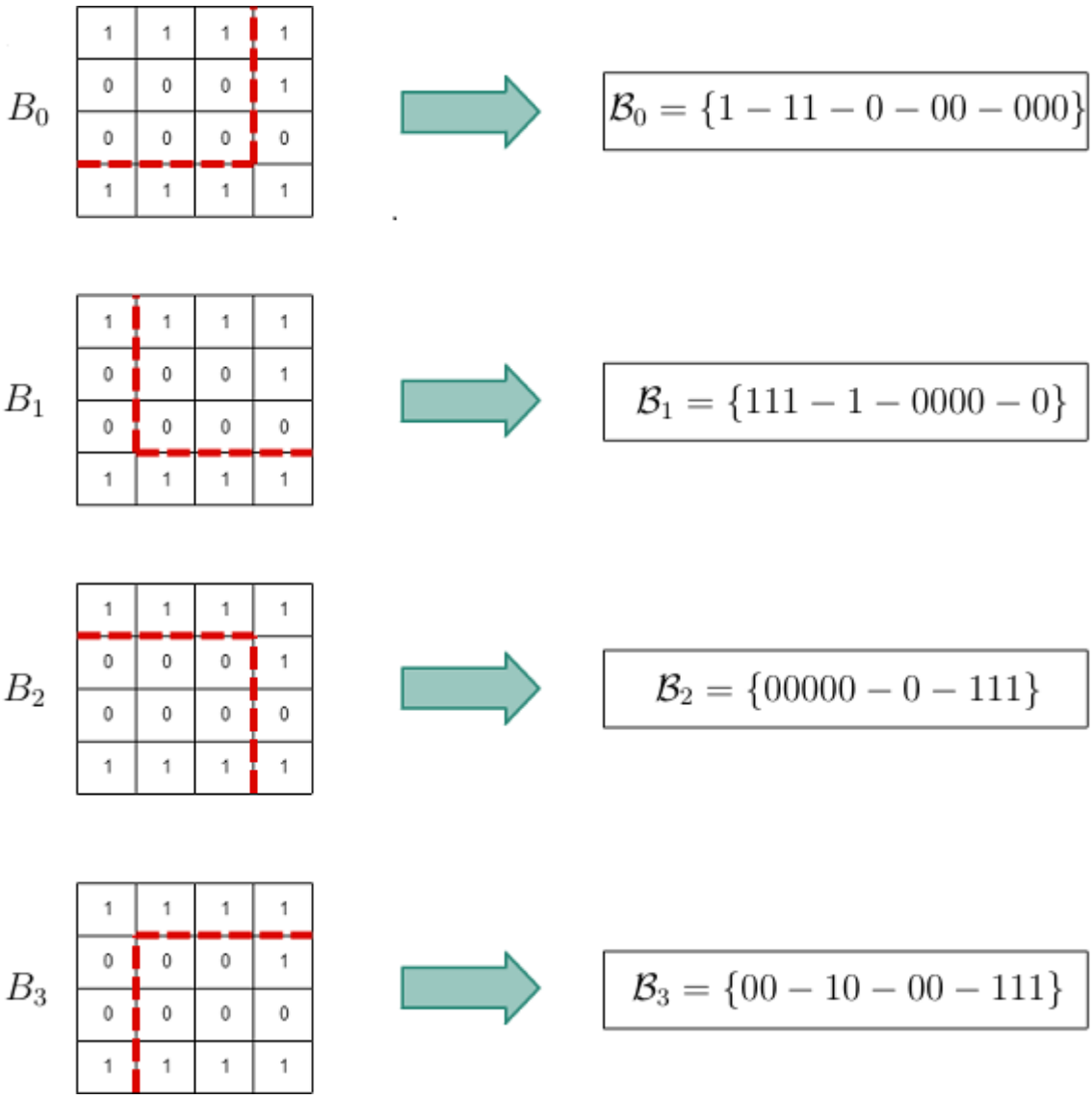

Figure 4.6: Compressed blocks $\mathcal{B}_k$ obtained with LZ method to blocks B_k of Figure 4.1.

Algorithm 4.1.2: Pseudocode for LZ Convolution Phase

Result: Convolution matrix entry $C(k)$

LZConvPhase (P, $\mathcal{B}_k$)
  $\texttt{off} \leftarrow 0$
  $C(k) \leftarrow 0$
  for $i = 0, \ldots, |\mathcal{B}_k| - 1$ **do**
    if $\mathcal{B}_k(i)$.`DotProduct` is not defined
      $\mathcal{B}_k(i)$.`DotProduct` $\leftarrow$ $\mathcal{B}_k(i)$.`Parent.DotProduct` $+ \mathcal{B}_k(i)$.`Value` $\times P(\texttt{off} + \mathcal{B}_k(i).\texttt{len} - 1)$
    end if
    $\texttt{off} \leftarrow \texttt{off} + \mathcal{B}_k(i)$.`len`
    $C(k) \leftarrow C(k) + \mathcal{B}_k(i)$.`DotProduct`
  end for
return $C(k)$

This method also runs in $\mathcal{O}(|\mathcal{B}_k|)$ time. However, it makes few more operations than RLE per factor and, the most important, in contrast with RLE, it does not access the memory sequentially because neither $\mathcal{B}_k(i)$ and $\mathcal{B}_k(i)$.`parent` nor $\mathcal{B}_k(i)$ and $\mathcal{B}_k(i+1)$ are expected to be consecutive in memory. With respect to the last statement recall that $\mathcal{B}_k(i)$ is a pointer to an entry in the data structure $\mathcal{D}$.

4.1.3 RLE+Lempel-Ziv

We propose a variation of the previous method in which the preprocessing phase of this method has two subphases. In the first one, we compress each block using RLE. In the second subphase, each compressed block is applied to the LZ parsing previously explained. The main difference is that each factor in the previous approach corresponds to a sequence of pixels and now each factor corresponds to a sequence of run lengths. Each run length can be thought as a pixel value.

The convolution phase is similar to the previous one except for the fact that now we have to take into account that each entry of the data structure corresponds to a sequence of run lengths rather than a sequence of values. This adds some extra operations per factor processed but the running time of the convolution phase is still linear on the number of factors of the compressed image.

4.2 Experimental Study

We carried on some experiments to evaluate the performance of our methods. We used 6 images available in TrainingImagesLibrary (2016) that are commonly used for studying simulation methods in geostatistics, which are shown in Figure 4.7. Their main features are presented in Table 4.1. All experiments were executed under the following settings of hardware/software: Intel Core i7-4500U CPU @ 1.80 GHz running Windows 8 64 bits, with 8 GB of memory. All codes were implemented in C++.

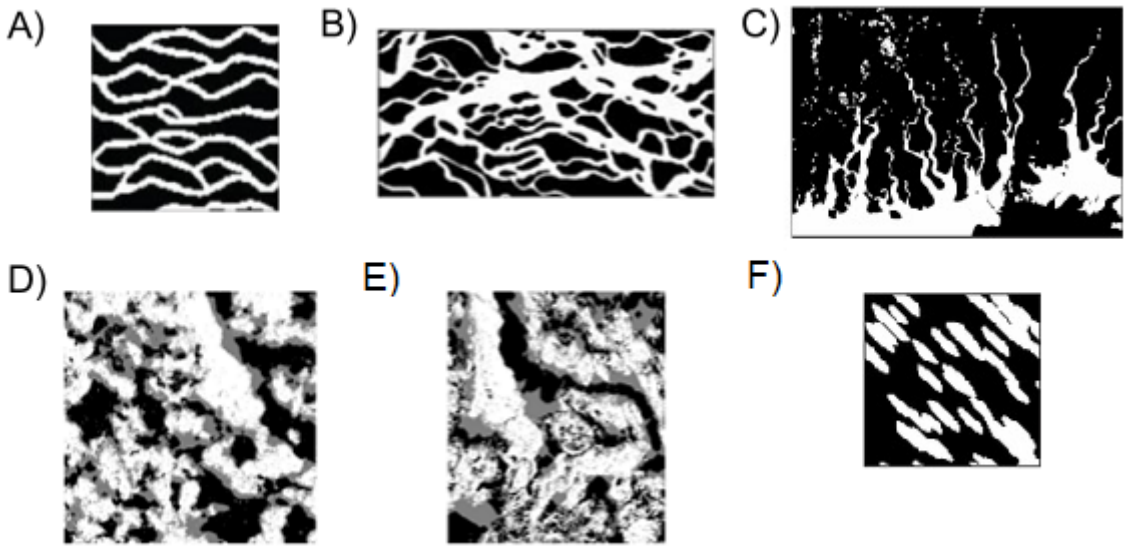

Figure 4.7: Training images used for compression experiments: available in (TrainingImagesLibrary, 2016).

Image	Dimensions	Type
(A) Strebelle	250×250	binary
(B) Bangladesh	768×243	binary
(C) Sundarban	1750×1200	binary
(D) C_Wlticat	400×400	ternary
(E) A_wlreferencecat	300×260	ternary
(F) Diagonal	100×100	binary

Table 4.1: Main features of the images used for the experimental study

We investigate the behaviour of the following methods: Naive, RLE-HS, HamPath, RLE-MO, LZ, RLE+LZ and Conv-FFT. Naive is the method derived directly from equation (4-1). RLE-HS is the first method described in Section 4.1.1 with a continuous horizontal scan. HamPath is the RLE based method that uses an optimized scanning order constructed by our variation of

Christofides' algorithm using the graph G_d, with $d = 3$. RLE-MO is also a RLE based method that uses for each block the best order between the horizontal and the vertical continuous one. LZ and RLE+LZ are, respectively, the first and the second methods described in Section 4.1.2.

Conv-FFT is our convolution's implementation based on the FFT. In its preprocessing phase it transforms the image from spatial domain to frequency domain. Its convolution phase works as follows: (i) it transforms a vector of dimension $w \times h + (m-1)w + m$, representing the pattern, from spatial domain to frequency domain; (ii) it computes the product between the image and the pattern in the frequency domain; (iii) it transforms the corresponding product from frequency domain to spatial domain. For computing the FFT and its inverse we used FFTW (Frigo & Johnson, 2005), a highly optimized library available in (FFTW, 2015), with parameter FFTW_MEASURE.

Table 4.2 shows the 'compression ratio' achieved by the preprocessing phase of our compression based methods for $m \in \{20, 40\}$. For LZ (RLE) based methods the ratio is given by the number of factors (run lengths) per block divided by the size of each block. For all images, but Diagonal, the LZ based methods clearly outperformed the RLE based ones. For binary images, RLE+LZ was significantly better than LZ while for ternary images LZ was slightly better than RLE+LZ. With respect to the RLE based methods, RLE-HS was slightly better than HamPath for some cases (up to 6%) while for others, as Sundarban and Diagonal images, HamPath managed to reduce in up to 40% the number of run lengths. RLE-MO obtained a reduction, ranging from 3% to 14%, on the number of run lengths with regard to HamPath for all images but Diagonal. For the latter HamPath provided a reduction of more than 35%.

Table 4.3 presents the convolution time for convolving 500 randomly chosen square patterns of dimension $m \times m$, with $m \in \{10, 20, 40\}$. Some observations are in order:

– Naive was the slowest method, among all tested. As an example, HamPath was 20 times faster than Naive, ranging from 3.4 times to 90 times. With regard to RLE based methods, HamPath was significantly faster than RLE-HS for Sundarban and Diagonal while for the other images both presented similar performance, with a slight advantage for RLE-HS. RLE-MO and HamPath presented similar results for all images but for Diagonal, where the latter was clearly faster. It is worth mentioning that the small advantage of RLE-MO in terms of compression ratio did not translate into time savings, probably due to the overhead of dealing

Image	Pattern size	RLE-HS	HamPath	RLE-MO	LZ	RLE+LZ
Strebelle	20×20	3.20	3.39	3.18	2.54	**1.81**
Strebelle	40×40	2.99	3.10	2.99	2.87	**2.09**
Bangladesh	20×20	4.76	4.86	4.50	2.65	**1.98**
Bangladesh	40×40	4.52	4.68	4.47	2.97	**2.38**
Sundarban	20×20	1.35	1.13	0.97	0.63	**0.45**
Sundarban	40×40	1.19	0.93	0.84	0.66	**0.36**
C_Wlticat	20×20	16.46	16.62	15.66	**6.31**	6.41
C_Wlticat	40×40	16.33	16.42	15.88	**6.51**	6.77
A_wlreferencecat	20×20	24.55	23.41	22.45	**9.36**	9.64
A_wlreferencecat	40×40	23.97	22.87	22.44	**9.50**	9.97
Diagonal	20×20	9.25	5.92	9.03	7.34	**5.15**
Diagonal	40×40	9.33	**5.35**	9.13	8.53	6.37

Table 4.2: Compression ratio of images in percentage values. For LZ (RLE) based methods the compression ratio is given by the number of factors (run lengths) per block over m^2

with more than one order. The results for RLE-MO and RLE-HS are omitted in Table 4.3 for the sake of a clean presentation.

– The RLE based approach outperformed the LZ based one by a factor of 10, in average. Since all compression based methods run in linear time on the number of factors/run lengths, one could have expected that the LZ based methods would be the most successful ones. The advantage of RLE approach, however, is because it accesses memory sequentially while this does not happen with the LZ approach. This difference makes the former much more efficient in terms of caching, which is translated into significant time saving.

– The results obtained by the RLE approach are competitive with those obtained by Conv-FFT. In general, for small patterns the former is superior than the latter. The threshold in which Conv-FFT starts to be advantageous depends on the dimension of the image and how compressible it is. As an example, for Sundarban, the most compressible image, HamPath is faster than Conv-FFT for patterns up to 100x100. On the other hand, for A_wlreferencecat, the least compressible one, Conv-FFT outperforms both HamPath and RLE-HS for square patterns larger than 10x10.

Although the minimization of preprocessing time was not the main focus of our research, it cannot be prohibitive. In the experiments reported in Table 4.3, the preprocessing phase of HamPath, the slowest among RLE based methods, was always faster (at least 15%) than its convolution phase for 500

Image	Pattern size	Naive	HamPath	LZ	RLE+LZ	Conv-FFT
Strebelle	10x10	6.17	**0.469**	0.984	0.688	3.16
Strebelle	20x20	22.7	**1.05**	8.17	7.19	3.08
Strebelle	40x40	76.5	**2.45**	42.8	46.9	3.5
Bangladesh	10x10	18.7	**2.03**	2.48	2.09	5.53
Bangladesh	20x20	71.4	**4.50**	32.5	32.2	10.9
Bangladesh	40x40	254	11.5	237	215	**5.75**
Sundarban	10x10	217	9.77	8.64	**8.27**	143
Sundarban	20x20	869	**15.7**	68	49	196
Sundarban	40x40	3458	**38**	702	389	261
C_Wlticat	10x10	19.4	**4.34**	14.1	18.3	13.9
C_Wlticat	20x20	65.8	11	117	130	**8.98**
C_Wlticat	40x40	241	29.6	572	616	**16.4**
A_wlreferencecat	10x10	8.3	**2.42**	9.05	13.6	3.28
A_wlreferencecat	20x20	32.3	6.59	74.8	89.2	**4.05**
A_wlreferencecat	40x40	111	17.6	359	404	**3.92**
Diagonal	10x10	1.13	**0.109**	0.234	0.219	0.531
Diagonal	20x20	3.17	**0.188**	1.45	1.73	0.609
Diagonal	40x40	6.89	**0.281**	7.41	10	0.672

Table 4.3: Time for calculating 500 convolutions in seconds.

patterns. When we have to convolve a large number of patterns ($>> 500$), as in the case of the application that motivated Problem 4.0.1, the time required for the preprocessing phase becomes almost negligible.

We conclude this chapter by mentioning that an heuristic that switches between FFT and RLE based methods, depending on the image and the dimension of the pattern, is the approach we recommend for efficiently solving Problem 4.0.1.

5
LSHSIM

We now present our proposed pattern-based MPS method which has two central pillars: the LSH technique, described in Section 3.5, and an adaptation of the RLE based search described in Section 4.1.1.

5.1
Method

Two points are often considered by pattern-based MPS methods available in the literature: (i) the choice of the similarity measure and (ii) how to efficiently find a pattern in the TI that is (very) similar to a given data event. To address (i), we use the Hamming similarity for categorical images and the Euclidean distance for continuous images. With regard to the second point, we propose the application of the LSH scheme to filter patterns that are likely to be similar to a given data event, followed by an exhaustive search. This search is used to find the most similar patterns among the filtered ones and is based on the RLE similarity calculation when the TI is categorical. Our techniques can be adapted to work together with different types of simulation paths as random or raster paths. Here, we explain how they are used with raster paths.

The pseudocode of our method for categorical TIs is presented in Algorithm 5.1.1. Further, we explain how to modify it for handling continuous TIs. In line 2, the set of hash tables for the LSH scheme is built. The details of how LSHSIM applies this scheme are given in Sections 3.5.1 and 5.1.1. In line 3 each pattern of the TI is compressed using the RLE method described in Section 4.1.1. We adapt the RLE method to calculate the Hamming similarity, which will be described in Section 5.1.2. We observe that these two lines, 2 and 3, that are computationally expensive, just need to be executed once in the usual case where multiple realizations are generated.

In line 4, a raster path is defined based on the template size, $\texttt{size}_{\texttt{T}}$, and the overlap size, $\texttt{size}_{\texttt{OL}}$. In this step, our method chooses a random corner of the realization as a starting point, as well as a random direction (between horizontal or vertical), to generate the path. For each location $\texttt{u}$ defined along the path, the corresponding data event $\texttt{dataEvent}_{\texttt{u}}$ is extracted from the realization $\texttt{R}$ and then the search phase of the LSH scheme is executed (lines 6

and 7) so that the set `cand`, which is supposed to contain patterns similar to $\texttt{dataEvent}_{\texttt{u}}$, is obtained. Note that the size of this set `cand` is limited to at most the value of α times the number of TI patterns, where α is a positive value smaller than 1. If this set is not empty, the Hamming similarity is calculated between the data event and each of the filtered patterns using the RLE approach (lines 8 - 9). Otherwise, the same approach is applied over the compressed TI, considering only a fraction α of all the patterns of the TI (lines 10 - 12), thus reducing the search space. In both cases, the subset `bestCand`, containing the `maxCandidates` most similar candidates, is obtained. Finally, in lines 13 and 14, a random pattern from this set is chosen, the MEBC method is applied to it and the result is pasted in realization at location `u`. The MEBC algorithm will be explained in Section 5.1.3. Figure 5.1 provides an overview of LSHSIM.

For continuous data, the Algorithm 5.1.1 requires some small changes: line 3 is not executed, because the RLE method does not apply for this case. Lines 8 and 9 perform a non-compressed search, calculating the Euclidean distance between the data event and each filtered pattern. Lines 10 - 12 perform a non-compressed search in the original TI, considering only a fraction α of all its patterns.

Algorithm 5.1.1: Pseudocode for LSHSIM

```
   Result: Realization R
 1 LSHSIM (ti, size_T, size_OL, maxCandidates, K, L, α)
 2   PreprocessLSH(ti, size_T, size_OL, K, L)
 3   compressedTI ← PreprocessRLE(ti, size_T, size_OL)
 4   path ← generateRasterPath(size_T, size_OL)
 5   for each location u ∈ path do
 6     dataEvent_u ← R(u)
 7     cand ← applyLSH(dataEvent_u, K, L, α)
 8     if cand ≠ ∅
 9       bestCand ← exhaustiveSearchCandidatesSet(dataEvent_u,
     cand, maxCandidates)
10     else
11       bestCand ← exhaustiveSearchTrainingImage(dataEvent_u,
     compressedTI, α, maxCandidates)
12     end if
13     chosenPat ← drawRandom(bestCand)
14     R(u) ← applyMEBC(chosenPat)
15   end for
16 return R
```

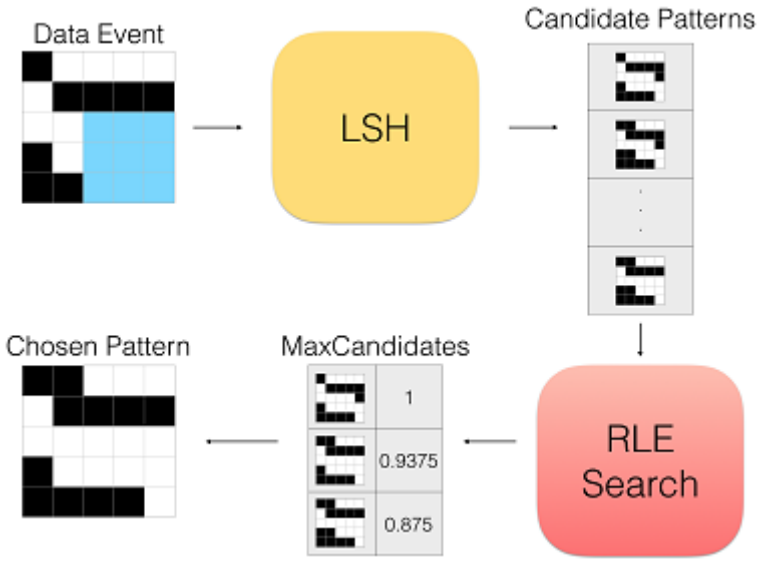

Figure 5.1: General structure of LSHSIM.

5.1.1 Filtering Patterns via LSH

In the preprocessing phase (line 2 of Algorithm 5.1.1), LSHSIM builds 3 sets of LSH tables as explained in Section 3.5. Each set corresponds to one of the 3 possible types of overlap regions described in Tahmasebi et al. (2012). Thus, for each pattern of a given size in the TI, the method extracts three regions and inserts each of them in the corresponding set of LSH tables. Figure 5.2 illustrates this phase using a TI with two facies (black and white), template size of 5×5 and overlap size of 2. The second image, from left to right, is an arbitrarily chosen pattern, say P, from the TI. On its right side, we have three images, in each of them the non-gray values represent a possible overlap area of P. These areas are inserted in the corresponding LSH table.

In the search phase (line 7 of Algorithm 5.1.1), it first verifies the type of overlap of the data event $\texttt{dataEvent}_\texttt{u}$ and then search in the corresponding set of LSH tables, such as illustrated in Figure 5.3. In order to speed up the search, the size of the returned set, `cand`, is limited by a fraction α of all possible patterns of the TI.

It can be proved that the probability of including a pattern P in the set `cand` (line 7) is given by

$$1 - (1 - Sim(P, \texttt{dataEvent}_\texttt{u})^K)^L$$

Thus, K and L shall be defined in order to guarantee that patterns similar (non-similar) to $\texttt{dataEvent}_\texttt{u}$ have a large (small) probability of being included

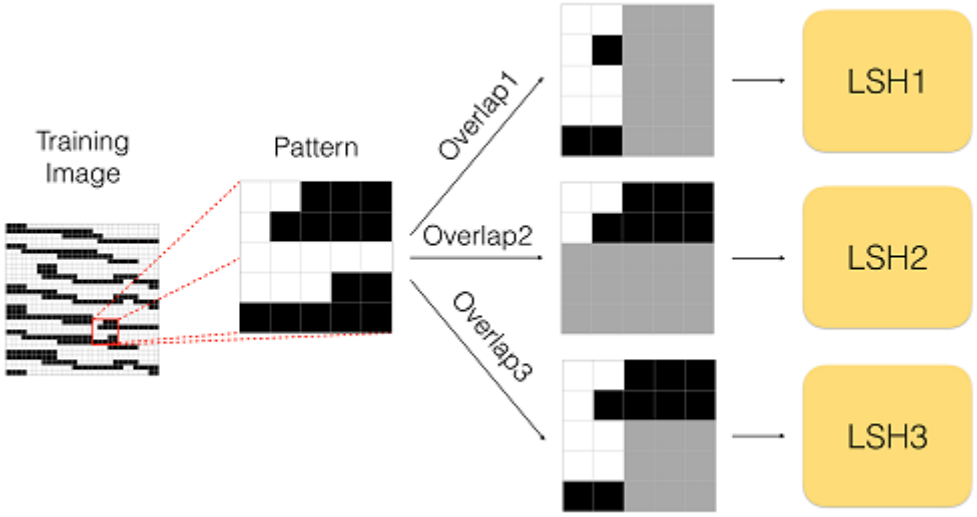

Figure 5.2: Preprocessing phase of LSHSIM.

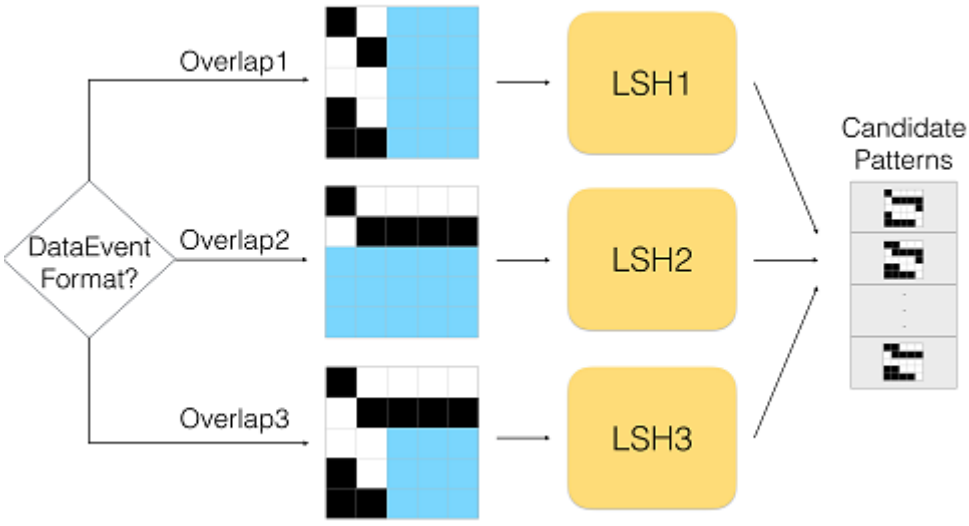

Figure 5.3: Search phase of LSHSIM.

in `cand`. As an example, by setting $K = 10$ and $L = 30$, the probability of including a pattern with similarity 0.8 is 95% while the probability of including a pattern with similarity 0.5 is less than 3%.

The extension of LSHSIM to 3D TIs makes use of 7 sets of LSH tables, each set corresponding to one of the 7 possible types of overlap regions that exist in three-dimensional patterns. Apart from that, the preprocessing and search phases proceed analogously as described above.

We first preprocess the TI, so as to compress it following the RLE approach. In fact, we compress each block of the image and store it, where a block corresponds of a submatrix of The TI iif The image describes this step.

5.1.2 Computing Hamming Similarity via RLE

We adapted the method of Section 4.1.1 to calculate the Hamming similarity, instead of the convolution. In this sense, the preprocessing phase is the same as described, but the convolution phase has small modifications which we now describe. We now preprocess a data event D to obtain a 3-dimensional structure Sum where $Sum[f, i, j]$ stores the number of times facie f occurs between the i-th and the j-th position of D. Then, the Hamming similarity between a data event D and a pattern P, with RLE representation $\{(c_1, v_1), \ldots, (c_k, v_k)\}$, is given by:

$$HammingSimilarity(P, D) = \sum_{i=1}^{k} Sum\left[v_i, \sum_{j=1}^{i-1} c_j, \left(\sum_{j=1}^{i} c_j\right) - 1\right].$$

The computation of the Hamming similarity between a data event D and each pattern in a given list $(P_1, \ldots, P_m)$ is made in $\mathcal{O}(|D| + \sum_{i=1}^{m} p_i^R)$, where $|D|$ is the size of D and p_i^R is the size of the RLE representation for P_i.

As an example, the Figure 5.4 exhibits a small TI, with facies 0 and 1, and a pattern P of size 4×4 delimited by the red dashed line. If we scan P following a horizontal continuous path, its RLE is $\{(5, 1), (3, 0), (3, 1), (5, 0)\}$, where the first value in each pair denotes the number of repetitions and the second one denotes the facie. The Hamming similarity between the data event D and the pattern P is given by:

$$HammingSimilarity(P, D) = Sum[1, 0, 4] + Sum[0, 5, 7] + Sum[1, 8, 10] + Sum[0, 11, 15] = 8$$

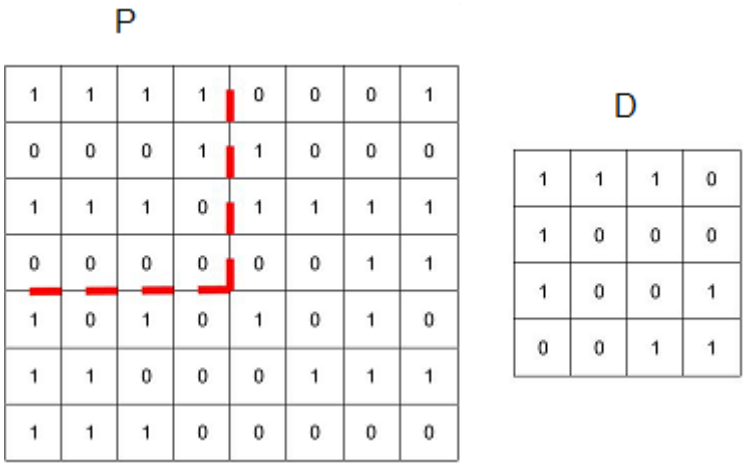

Figure 5.4: An example TI, a possible pattern P and a data event D.

5.1.3 Minimum Error Boundary Cut

The Minimum Error Boundary Cut (MEBC) was first proposed by Efros & Freeman (2001) in the Image Quilting method. It is used by the majority of recent pattern-based methods which perform a raster path and we also employ it in LSHSIM. This method is applied when a pattern is pasted in the realization, so as to find an optimal boundary cut in the overlap region that maximizes the image's continuity.

Figure 5.5 on top shows two patterns P_1 and P_2, from a given training image, of size $T \times T$ and their overlap regions, delimited by the red line, of size $T \times OL$. Then, Figure 5.6 exhibits a naïve pasting, i.e., without using the MEBC method, of pattern P_2 over the overlap region of P_1, and a pasting of both patterns employing MEBC, where the boundary cut obtained is denoted by the green line. In both cases, an image of size $T \times (2T - OL)$ is obtained. Therefore, in the overlap region of the pasting with MEBC, values to the left of the boundary cut belong to pattern $P1$ and values to the right come from pattern $P2$. It is possible to verify that the use of the MEBC method corrected some discontinuities in the channels on top and on bottom of the image.

Typically, if a pattern-based method that performs a raster path does not employ MEBC, the realizations tend to be patchy, where one can notice the effect of rectangular shapes in the pasting areas. In addition to this benefit of improving the realization's continuity, the fact that the patches are cut and assembled in a coherent manner results in new patterns being created, whereas most MPS algorithms are limited to only using the patterns present in the TI (Mahmud et al., 2014).

The pseudocode of MEBC is described in Algorithm 5.1.2. This algorithm

is based on the dynamic programming (DP) strategy (Cormen et al., 2009) and it has complexity $\mathcal{O}(T \times OL)$. We describe it for vertical overlaps, but the adaptation for horizontal overlaps is straightforward. When there is both a vertical and a horizontal overlap, the overall minimum is chosen for the cut in the shared area.

In line 2, an error surface e is defined corresponding to the squared difference of the overlap regions of patterns P_1 and P_2, having a rectangular size $T \times OL$, such as depicted by the bottom of Figure 5.5. The DP minimizes this error surface, computing the cumulative minimum error E over the overlap region (lines 3 - 15). The DP recurrence (lines 5 - 12) defines the value of $E_{i,j}$ equals to $e_{i,j}$ if it's the first row; otherwise, the value of $E_{i,j}$ is obtained using the 3 closest pixels on the previous row (or only 2, if it's on an edge of the overlap region). Line 16 retrieves the minimum value of the last row of E, which is the arrival point of a cut of minimum cost. Finally, in line 17, it goes backward from this arrival point and traces back the minimum value for each row, thus recovering the minimum cut.

Algorithm 5.1.2: Pseudocode for MEBC

Result: Minimum error cut $minCut$

1 **MEBC** (P_1, P_2, T, OL)
2 $e \leftarrow (P_1^{ol} - P_2^{ol})^2$
3 **for** $i \leftarrow 1, \ldots, T$ **do**
4 **for** $j \leftarrow 1, \ldots, OL$ **do**
5 **if** $i = 1$
6 $E_{i,j} \leftarrow e_{i,j}$
7 **else if** $j = 1$
8 $E_{i,j} \leftarrow e_{i,j} + \min\{E_{i-1,j}, E_{i-1,j+1}\}$
9 **else if** $j = OL$
10 $E_{i,j} \leftarrow e_{i,j} + \min\{E_{i-1,j-1}, E_{i-1,j}\}$
11 **else**
12 $E_{i,j} \leftarrow e_{i,j} + \min\{E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}\}$
13 **end if**
14 **end for**
15 **end for**
16 $minCut(T) \leftarrow \min_j\{E_{T,j}\} \;\; 1 \leq j \leq OL$
17 traceBack(T, E, $minCut$)
18 **end for**
19 **return** $minCut$

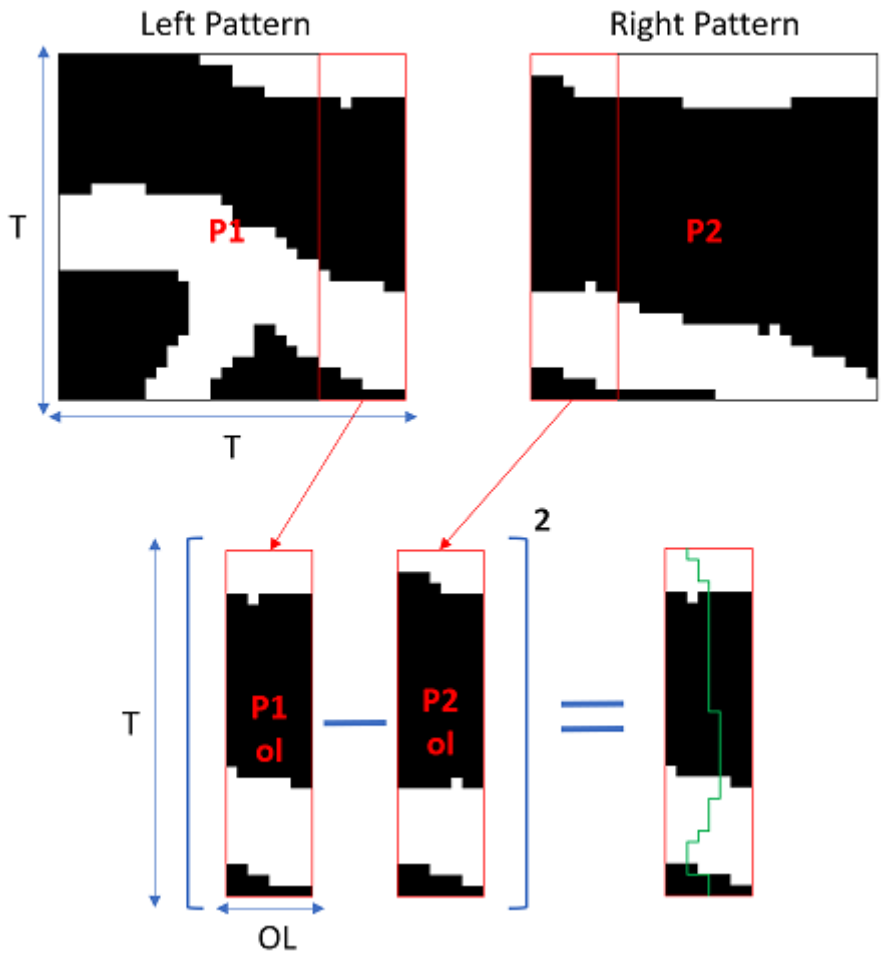

Figure 5.5: Two patterns, their overlap regions and obtained minimum boundary cut.

5.1.4 Conditioning

We adapted LSHSIM so as to consider conditioning data. In this sense, we introduced an additional filter when searching for a given data event. After applying the LSH scheme and obtaining the set of candidate patterns, we filter this set to those patterns which honor all the hard data associated with the data event. Finally, we perform a RLE based search in this reduced set of candidates, looking for the most similar ones to the data event in the overlap region. In case this reduced set of candidate patterns is empty, we perform a RLE search in the training image.

It shall be noted that, in order to avoid low quality realizations, the α parameter should be increased with respect to unconditional simulations, since we now have this additional filter that restricts the set of candidate patterns to those which honor the hard data.

The experiments performed showed that LSHSIM is able to achieve good

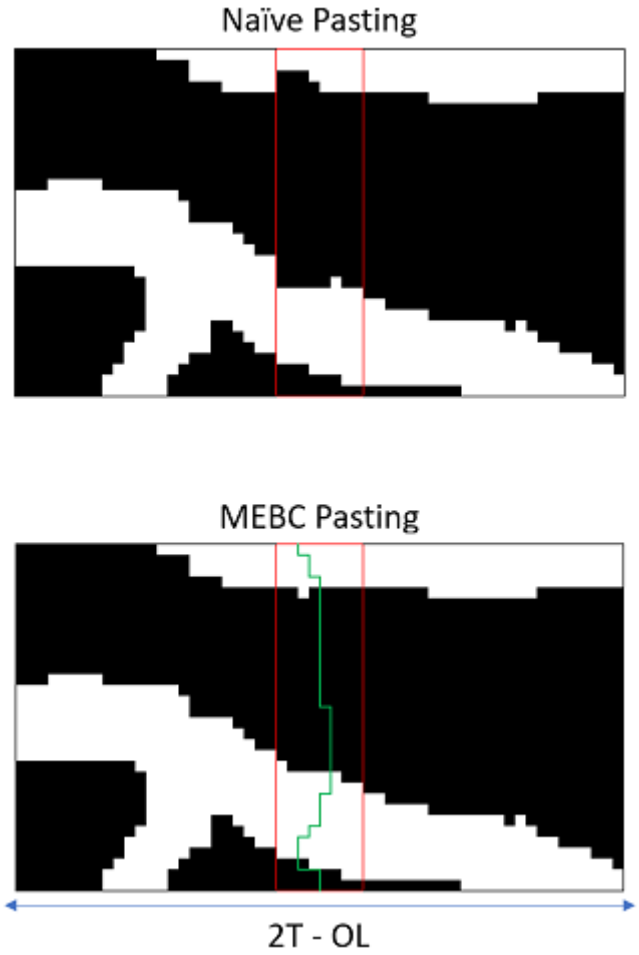

Figure 5.6: Comparison of a naïve pasting and a pasting employing MEBC.

quality realizations while honoring conditioning points. The computational times are higher than those for unconditional realizations due to the increased value of α used. These experiments are described in details in Section 5.2.4.

5.2 Experimental Study

In our experiments, we considered a set of four 2D and three 3D TIs available in (TrainingImagesLibrary, 2016), which is a repository of TIs associated with the book of Mariethoz & Caers (2014), so as to evaluate our proposed solution. The images are presented in Figure 5.7, while their main properties are described in Table 5.1. The image (A) is the well known TI proposed by Strebelle (2002), while the image (C) is a ternary and less compressible one. The Stonewall image (D) was selected to validate our method with continuous data. Lastly, the images (E), (F) and (G) are categorical 3D TIs used to validate our method with 3D models.

All experiments were executed under the following settings of hardware

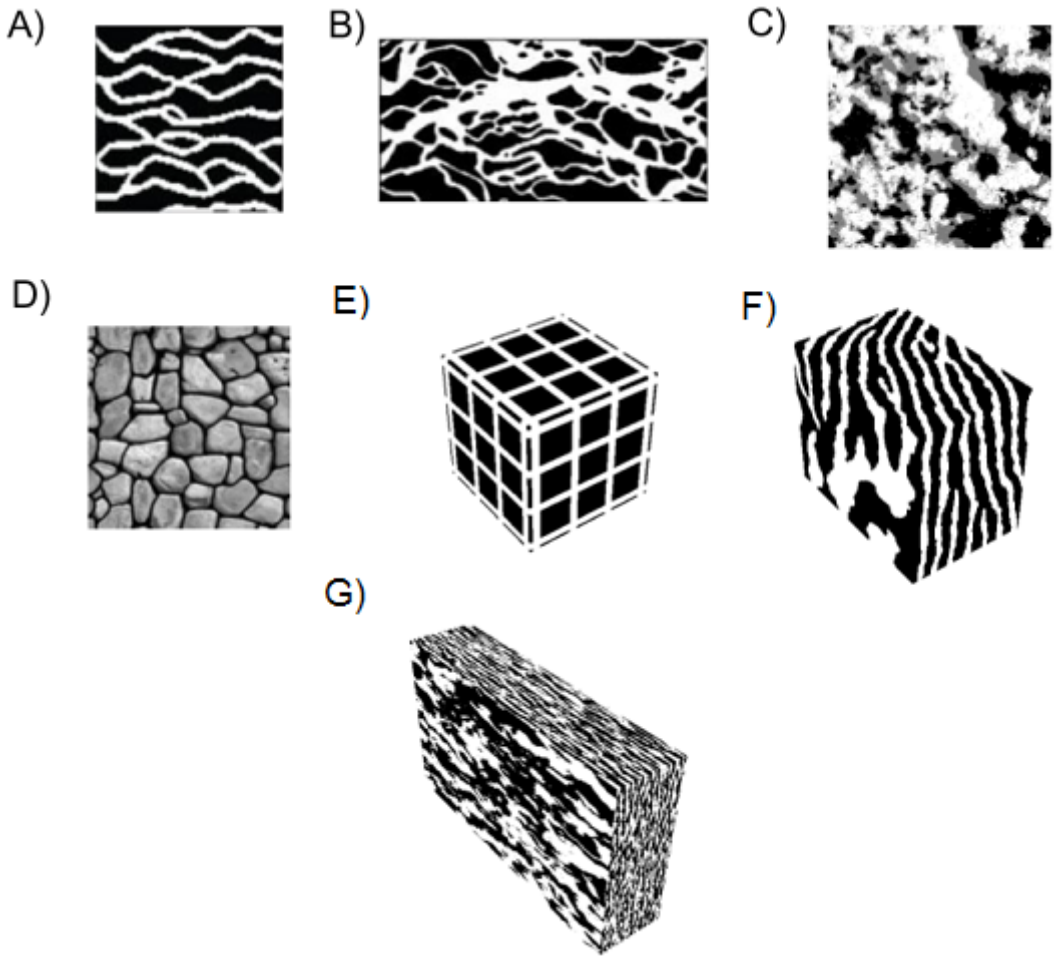

Figure 5.7: Training images adopted in our experiments: available in (TrainingImagesLibrary, 2016).

and software: Intel Core i7-3960X CPU @ 3.30GHz running Windows 7 64 bits, with 32 GB of memory. All codes of our method were implemented in C++ and compiled using Visual Studio 2015 update 1. Regarding the MS-CCSIM method, we adopted the following strategy: we used the MATLAB code available in (MS-CCSIM, 2016) to generate realizations and we also implemented a version in C++, employing the OpenCV library version 2.4.11 (OPENCV, 2016), in order to compare its computational time with LSHSIM's time. Note that this library is an optimized code belonging to the computer vision area, having very efficient implementations for some of the techniques required to implement MS-CCSIM as the fast Fourier transform (FFT) and multi-scale algorithms.

For parametrization of the MS-CCSIM method, both in MATLAB and C++ implementations, we set the number of scales to 3, which is the highest in its MATLAB code. Regarding LSHSIM, when dealing with categorical TIs,

TI	Image size	Dimensions	Type
(A) Strebelle	250×250	2D	Binary
(B) Bangladesh	768×243	2D	Binary
(C) C_Wlticat	400×400	2D	Ternary
(D) Stonewall	200×200	2D	Continuous
(E) Checker	$50 \times 50 \times 50$	3D	Binary
(F) Fold_Categorical	$180 \times 150 \times 120$	3D	Binary
(G) Maules_Creek	$340 \times 200 \times 80$	3D	Binary

Table 5.1: Main properties of the images used for the experimental study.

we defined L and K, the LSH parameters, to 30 and 10, respectively. On the other hand, for continuous TIs, we set L and K to 30 and 8, respectively. For both methods, we also set `maxCandidates`, which is the number of most similar patterns returned in a search, to 10, while varying template and overlap sizes according to the experiment being made.

To determine a suitable value of the α parameter, that is to say, the one that achieves a good balance between computational time and realization's quality, we performed several experiments for different configurations of template and overlap sizes. We end up with α equals to 0.5% for categorical 2D TIs, 1% for categorical 3D TIs and 5% for the continuous 2D TI. We perform a sensitivity analysis of the α parameter, as well as L and K, in Section 5.2.5.

Both MATLAB and C++ implementations of MS-CCSIM apply the MEBC approach to the patchiness problem.

5.2.1 CPU performance

In this subsection, we evaluate the performance of our method regarding the computational time for generating unconditional realizations. More specifically, for each TI under consideration, we generated 20 realizations for different configurations of template and overlap sizes.

We then measured the time taken for performing each realization and calculated its average. For 2D categorical TIs, we compare the performance of LSHSIM with our implementation of the MS-CCSIM in C++. Table 5.2 shows these times in milliseconds, where the best one for each configuration is in bold.

For binary images, LSHSIM was able to outperform MS-CCSIM by a factor of approximately 7 on average. The difference was bigger for the Strebelle image, the most compressible one, for which our method was 8 times faster. Regarding the ternary one, our method was 3.70 times faster than MS-CCSIM on average, ranging from 2.19 to 6.35 times. This difference is explained by

Image	Real. size	Temp. size	Overlap	LSHSIM	MS-CCSIM	Ratio
Strebelle	256×256	16×16	4	**11.85**	106.62	9.00
Strebelle	256×256	32×32	4	**3.82**	29.40	7.70
Strebelle	256×256	32×32	8	**4.91**	36.34	7.40
Strebelle	400×400	16×16	4	**29.64**	262.31	8.85
Strebelle	400×400	32×32	4	**9.59**	71.68	7.47
Strebelle	400×400	32×32	8	**12.79**	93.60	7.32
Bangladesh	256×256	16×16	4	**32.29**	272.92	8.45
Bangladesh	256×256	32×32	4	**11.46**	56.86	4.96
Bangladesh	256×256	32×32	8	**14.11**	70.90	5.02
Bangladesh	400×400	16×16	4	**78.78**	671.19	8.52
Bangladesh	400×400	32×32	4	**28.47**	139.93	4.91
Bangladesh	400×400	32×32	8	**35.88**	183.92	5.13
C_Wlticat	256×256	16×16	4	**40.17**	254.90	6.35
C_Wlticat	256×256	32×32	4	**23.08**	50.46	2.19
C_Wlticat	256×256	32×32	8	**26.36**	65.98	2.50
C_Wlticat	400×400	16×16	4	**100.38**	597.56	5.95
C_Wlticat	400×400	32×32	4	**55.53**	130.02	2.34
C_Wlticat	400×400	32×32	8	**58.81**	169.96	2.89

Table 5.2: Average realization time in milliseconds for 2D categorical images.

the fact that these images are less compressible and hence each exhaustive search, which uses the RLE similarity calculation, takes longer. In summary, LSHSIM was about one order of magnitude faster than MS-CCSIM concerning all categorical TIs. We shall note that we are not taking into account the preprocessing time in this specific evaluation.

Table 5.3 exhibits, for the same configurations, the preprocessing time required for building the LSH data structure and applying the RLE compression to the training image. This preprocessing time is on average equivalent to the time of 48 realizations, which yields a non-negligible overhead for applications that only require the generation of a few realizations. For applications that involve a large number of simulations the preprocessing time of LSHSIM becomes almost irrelevant. The results of the experiments discussed so far, with MS-CCSIM and LSHSIM, suggest that the latter outperforms the former for applications where more than a dozen of realizations have to be generated. Moreover, the larger the number of realizations the larger is the advantage towards LSHSIM.

With regard to 2D continuous data, we do not compare our method with MS-CCSIM, since the latter does not deal with this kind of variable. In this sense, Table 5.4 exhibits, for the same configurations as above, the preprocessing and realization times in milliseconds using LSHSIM for the Stonewall TI. It can be noted that our method was able to obtain satisfactory

Image	Real. size	Temp. size	Overlap	Preprocessing time
Strebelle	256×256	16×16	4	418.08
Strebelle	256×256	32×32	4	510.90
Strebelle	256×256	32×32	8	519.48
Strebelle	400×400	16×16	4	426.66
Strebelle	400×400	32×32	4	509.34
Strebelle	400×400	32×32	8	520.26
Bangladesh	256×256	16×16	4	1460.95
Bangladesh	256×256	32×32	4	1948.45
Bangladesh	256×256	32×32	8	1878.25
Bangladesh	400×400	16×16	4	1469.53
Bangladesh	400×400	32×32	4	1909.45
Bangladesh	400×400	32×32	8	1853.29
C_Wlticat	256×256	16×16	4	1835.35
C_Wlticat	256×256	32×32	4	2666.06
C_Wlticat	256×256	32×32	8	2737.04
C_Wlticat	400×400	16×16	4	1830.67
C_Wlticat	400×400	32×32	4	2650.46
C_Wlticat	400×400	32×32	8	2695.7

Table 5.3: Preprocessing time in milliseconds for 2D categorical images.

realization times for this continuous TI.

Image	Real. size	Temp. size	Overlap	Preproc. time	Real. time
Stonewall	256×256	16×16	4	4034.97	101.40
Stonewall	256×256	32×32	4	6442.84	31.98
Stonewall	256×256	32×32	8	9906.84	67.08
Stonewall	400×400	16×16	4	4197.21	278.46
Stonewall	400×400	32×32	4	6364.06	79.56
Stonewall	400×400	32×32	8	9773.46	180.18

Table 5.4: Preprocessing and realization times in milliseconds for continuous image.

Finally, Table 5.5 gives the preprocessing and realization times in seconds obtained by applying LSHSIM to the 3D categorical TIs for some selected configurations. These times are three to four orders of magnitude larger than the ones exhibited in Table 5.2 for 2D images. However, this is not surprising since the sizes of the 3D realizations are about two to three orders of magnitude larger than the ones for 2D images. We shall remark that, for these 3D images, the preprocessing time of our method is generally much smaller than the time taken for performing a single realization.

Image	Real. size	Temp. size	Overlap	Preproc. time	Real. time
Checker	$256 \times 256 \times 256$	$10 \times 10 \times 10$	2	1.88	3.88
Checker	$256 \times 256 \times 256$	$12 \times 12 \times 12$	4	1.84	5.67
Checker	$256 \times 256 \times 256$	$16 \times 16 \times 16$	4	1.53	2.69
Checker	$400 \times 400 \times 400$	$10 \times 10 \times 10$	2	1.89	15.37
Checker	$400 \times 400 \times 400$	$12 \times 12 \times 12$	4	1.85	22.43
Checker	$400 \times 400 \times 400$	$16 \times 16 \times 16$	4	1.52	10.24
Fold_Categorical	$256 \times 256 \times 256$	$10 \times 10 \times 10$	2	76.82	165.75
Fold_Categorical	$256 \times 256 \times 256$	$12 \times 12 \times 12$	4	81.18	221.04
Fold_Categorical	$256 \times 256 \times 256$	$16 \times 16 \times 16$	4	91.99	102.85
Fold_Categorical	$400 \times 400 \times 400$	$10 \times 10 \times 10$	2	74.20	651.95
Fold_Categorical	$400 \times 400 \times 400$	$12 \times 12 \times 12$	4	71.09	804.84
Fold_Categorical	$400 \times 400 \times 400$	$16 \times 16 \times 16$	4	84.47	394.16
Maules_Creek	$256 \times 256 \times 256$	$10 \times 10 \times 10$	2	128.76	379.24
Maules_Creek	$256 \times 256 \times 256$	$12 \times 12 \times 12$	4	143.82	467.81
Maules_Creek	$256 \times 256 \times 256$	$16 \times 16 \times 16$	4	188.30	242.48
Maules_Creek	$400 \times 400 \times 400$	$10 \times 10 \times 10$	2	145.97	1588.56
Maules_Creek	$400 \times 400 \times 400$	$12 \times 12 \times 12$	4	153.37	1937.34
Maules_Creek	$400 \times 400 \times 400$	$16 \times 16 \times 16$	4	172.87	877.82

Table 5.5: Preprocessing and realization times in seconds for 3D images.

5.2.2 Realization's quality

We now analyse LSHSIM concerning simulation's quality. For this purpose, we compare LSHSIM's simulations with MS-CCSIM's for the configurations defined in the last section. In addition, we also show realizations of MS-CCSIM using only 1 scale, since it improves its quality, although at the cost of increasing the computational time by a factor of approximately 10 with respect to the times presented in the previous section.

Figure 5.8 (A), (B) and (C) shows two realizations generated with LSHSIM, MS-CCSIM with 3 scales and MS-CCSIM with 1 scale, respectively, for the Strebelle TI. Each realization has 256×256 pixels, template size of 32×32 and overlap of 4. Moreover, Figure 5.9 (A), (B) and (C) shows two realizations for the Bangladesh TI using LSHSIM, MS-CCSIM with 3 scales and MS-CCSIM with 1 scale, respectively. Both have 256×256 pixels, and they were generated using template size of 32×32 and overlap size of 4. Note that these images are resized to better fit the thesis.

For these two binary TIs, Strebelle and Bangladesh, we notice that LSHSIM generated realizations with good quality, in the sense that it reproduced well the spatial continuity of the TIs. Both LSHSIM and MS-CCSIM generated realizations containing low level of patchiness, since they employ the minimum-error boundary cut approach.

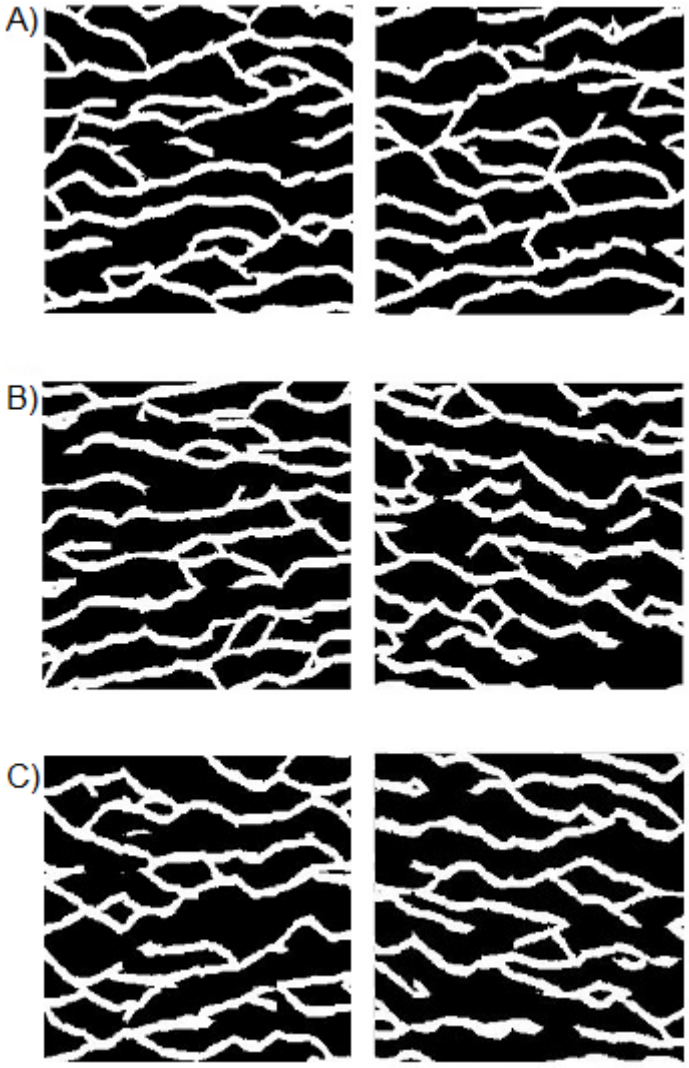

Figure 5.8: Unconditional realizations for the TI of Fig. 5.7 (A): using LSHSIM (A), using MS-CCSIM with 3 scales (B) and using MS-CCSIM with 1 scale (C).

Figure 5.10 (A), (B) and (C) presents two realizations generated with LSHSIM, MS-CCSIM with 3 scales and MS-CCSIM with 1 scale, respectively, for the C_Wlticat image, setting the realization size to 400×400 pixels, template size to 16×16 and overlap to 4. Again, LSHSIM was able to achieve a good quality, that is to say, representing well the image's characteristics.

To sum up, the simulations of both LSHSIM and MS-CCSIM are improved when the MEBC correction is applied, as it can be seen by the low level of patchiness obtained. We shall note, however, that the application of MEBC has a bigger impact in MS-CCSIM's realizations, since LSHSIM's simulations naturally present a lower level of patchiness even when the MEBC is not used.

LSHSIM was also able to produce realizations with good quality for

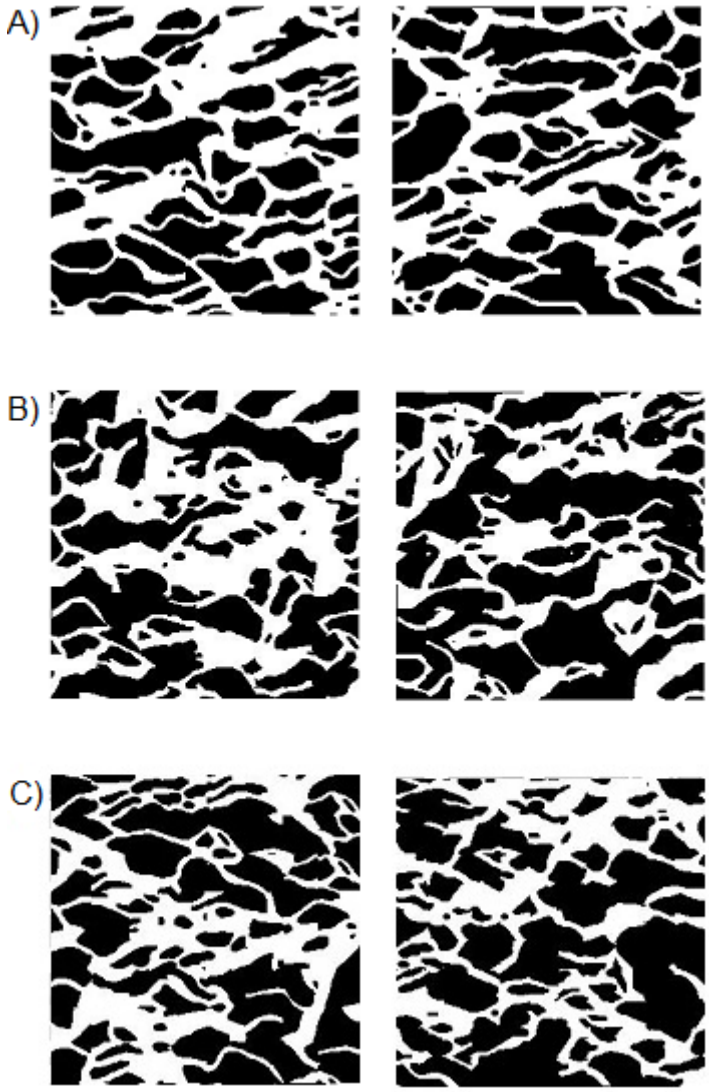

Figure 5.9: Unconditional realizations for the TI of Fig. 5.7 (B): using LSHSIM (A), using MS-CCSIM with 3 scales (B) and using MS-CCSIM with 1 scale (C).

continuous data. In this sense, Figure 5.11 (B) and (C) shows two realizations generated with LSHSIM for the Stonewall TI (A). Each of them has 256×256 pixels, template size of 16×16 and overlap of 4. One can notice that LSHSIM was able to express well the TI's spatial continuity.

Finally, LSHSIM was also successful for 3D images. Figure 5.12 presents realizations for the Checker TI (A), for the Fold_Categorical TI (B) and for the Maules_Creek TI (C), setting the realization size to $256 \times 256 \times 256$, template size to $16 \times 16 \times 16$ and overlap size to 4. Analogously, Figure 5.13 exhibits realizations for the same TIs with $400 \times 400 \times 400$ pixels, template size of $16 \times 16 \times 16$ and overlap of 4.

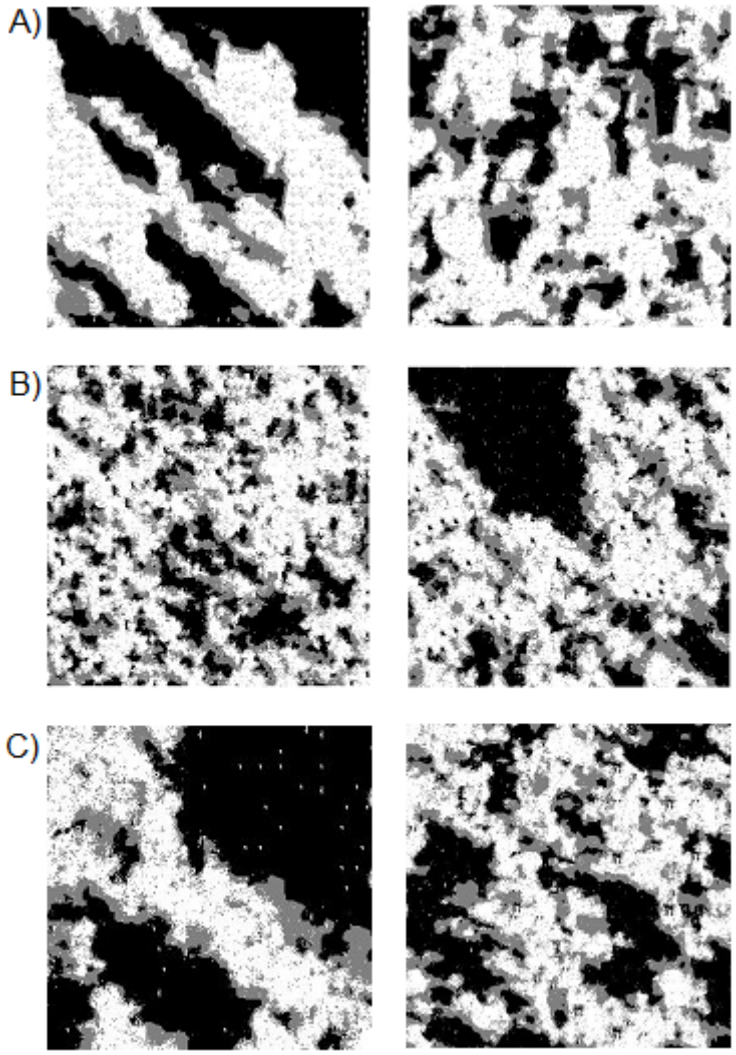

Figure 5.10: Unconditional realizations for the TI of Fig. 5.7 (D): using LSHSIM (A), using MS-CCSIM with 3 scales (B) and using MS-CCSIM with 1 scale (C).

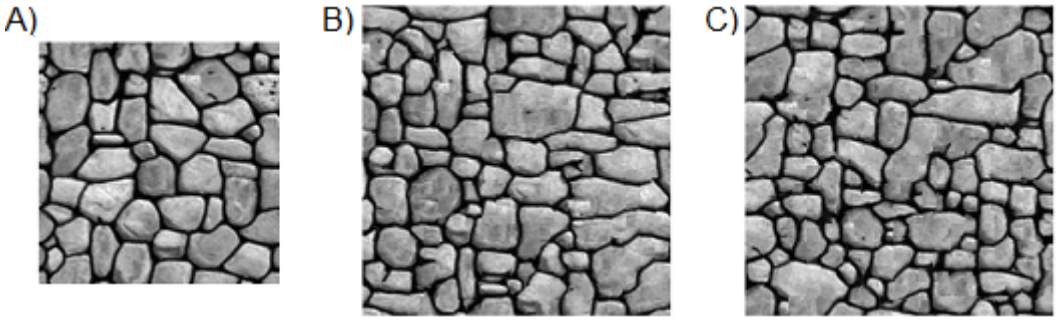

Figure 5.11: Unconditional realizations using LSHSIM for continuous data: the Stonewall TI (A) and two generated realizations (B) and (C).

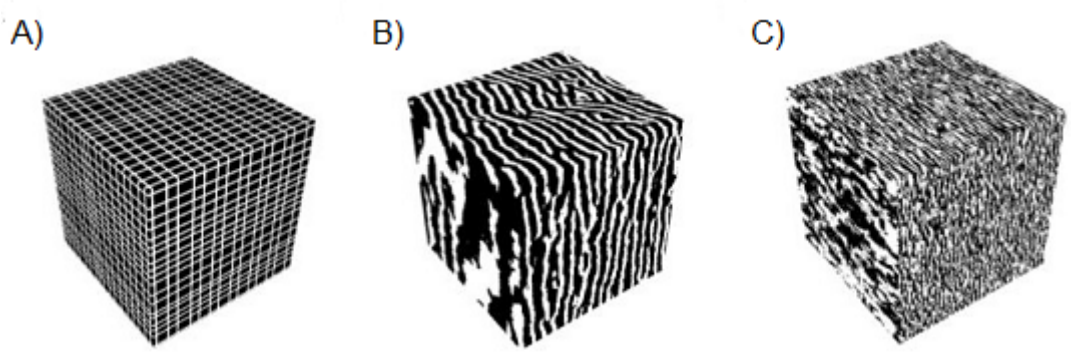

Figure 5.12: Unconditional realizations using LSHSIM for 3D data: for the Checker TI (A), for the Fold_Categorical TI (B) and for the Maules_Creek TI (C).

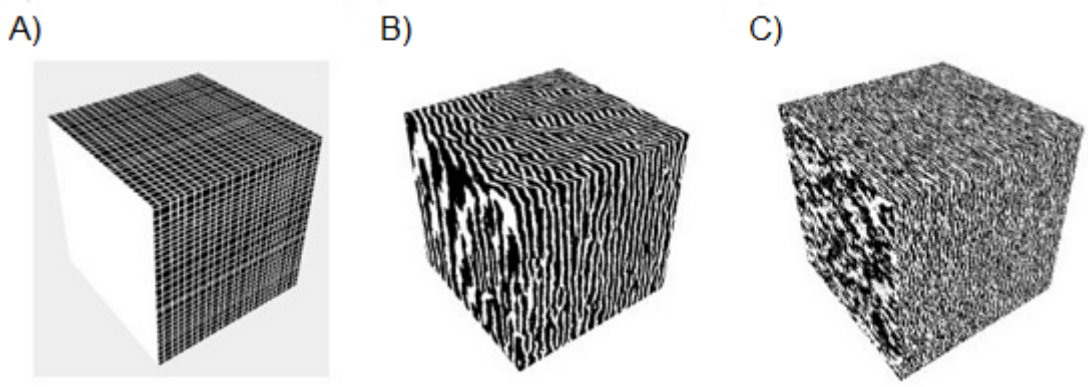

Figure 5.13: Unconditional realizations using LSHSIM for 3D data: for the Checker TI (A), for the Fold_Categorical TI (B) and for the Maules_Creek TI (C).

5.2.3 Comparing uncertainty space

We now analyse our uncertainty space following the analysis of distance (ANODI) method proposed by Tan et al. (2014). It is a way of comparing different algorithms and composing a rank, following two criteria: (i) pattern reproduction, given by the distance between realizations and TIs; and (ii) space of uncertainty, which is the distance between realizations.

We focus on ANODI's visual approach, which consists of the MDS technique with the Jensen-Shannon divergence as a measure of distance. It represents the realizations and the TI as points in a two or three-dimensional space, where the relative distances between each realization and the TI are preserved as much as possible. We used a MATLAB implementation of ANODI

available in (ANODI, 2016).

We generated 50 realizations with both methods for two TIs. Figure 5.14 shows the MDS plot for the Strebelle TI with the following settings: realization size of 256 × 256 pixels, template size of 32 × 32 and overlap of 4. Similarly, Figure 5.15 shows the MDS plot for C_Wlticat, which is a ternary image, using a realization of 400 × 400 pixels, a template size of 16 × 16 and an overlap of 4. In each plot, the black dot denotes the TI, while the green and blue points represent realizations generated with LSHSIM and MS-CCSIM, respectively. The numbers close to some points indicate the rank of that realization, among the ones generated by the same method, with respect to the distance to the TI in the original space. Note that the axis are not shown because the focus is on the relative distances between points.

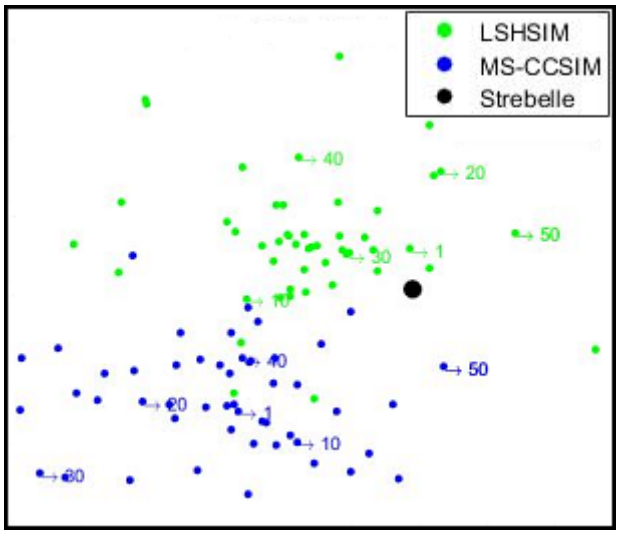

Figure 5.14: MDS plot illustrating the variability of LSHSIM and MS-CCSIM methods by using the TI in Fig. 5.7 (A).

One can observe that, for Figure 5.14, LSHSIM achieved a good pattern reproduction such that its realizations are close to the TI. In addition, both LSHSIM and MS-CCSIM had similar spreading of their realizations. Concerning the plot depicted in Figure 5.15, LSHSIM generated realizations close to the TI, thus reproducing well the TI patterns. Again, both methods had a similar variability, since LSHSIM's space of uncertainty is almost as large as MS-CCSIM's.

5.2.4 Conditioning

We also investigated the impact of conditioning on LSHSIM. Following the approach of Yang et al. (2016), we randomly selected 20 points from the

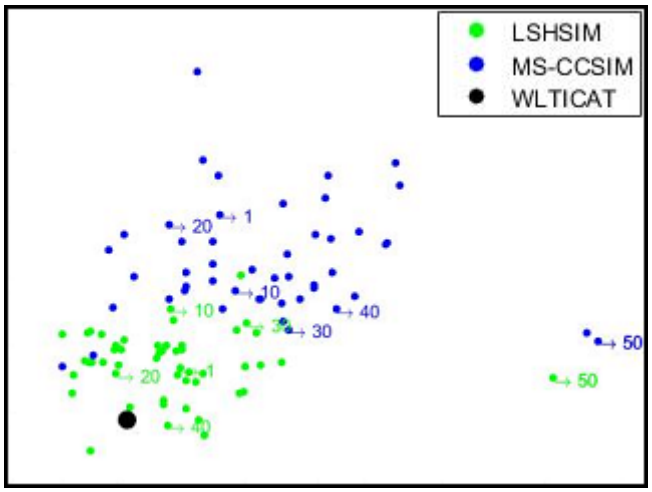

Figure 5.15: MDS plot exposing the variability of both methods by using the TI in Fig. 5.7 (C).

Strebelle TI to be used as hard data, ensuring that at least 1/3 of these points corresponded to channel facies (white values). Figure 5.16 shows the TI (A) and its selected points (B). We then generated 100 conditional realizations with LSHSIM for some selected configurations, setting $L = 30$ and $K = 10$. The α parameter was set to 15%, which is a higher value than the one used in unconditional realizations, such as explained in Section 5.1.4. Table 5.6 summarizes the average preprocessing and realization times, in milliseconds, obtained for these experiments.

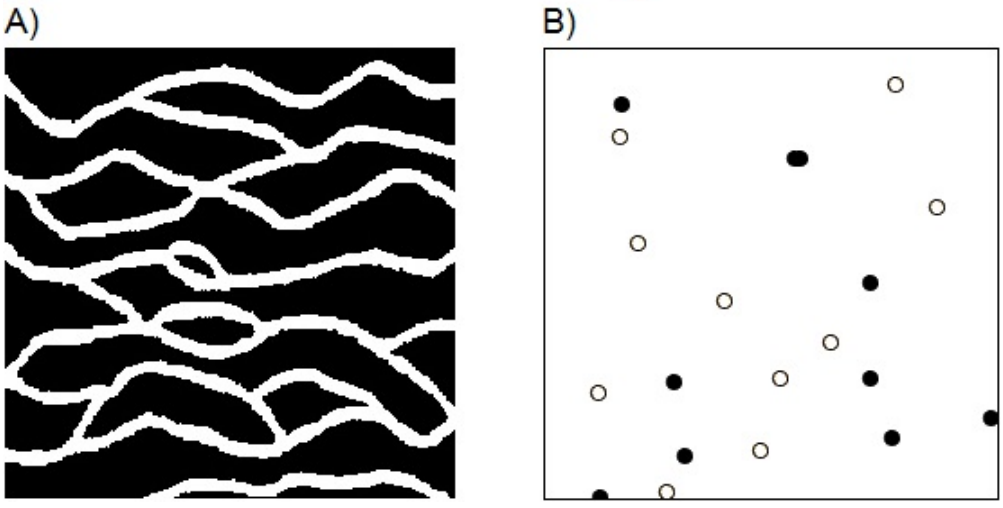

Figure 5.16: Strebelle TI (A) and selected conditioning points (B).

For the configuration having realization size of 256×256 pixels, template size of 32 and overlap of 4, Figure 5.17 exhibits three conditional realizations. Besides, Figure 5.18 shows the ensemble average scaled to the interval $[0, 1]$ of

Image	Real. size	Temp. size	Overlap	Preproc. time	Real. time
Strebelle	256×256	16×16	4	428.53	162.07
Strebelle	256×256	32×32	4	519.64	55.20
Strebelle	256×256	32×32	8	525.56	67.47
Strebelle	400×400	16×16	4	438.36	410.00
Strebelle	400×400	32×32	4	518.39	132.99
Strebelle	400×400	32×32	8	529.62	173.64

Table 5.6: Preprocessing and realization times in milliseconds for conditional simulations.

all the 100 generated realizations. It depicts a heat map in which each position corresponding to a black conditioning point should be close to blue and each one associated with white conditioning points should be close to red. Therefore, one can observe that all hard data are consistently honored.

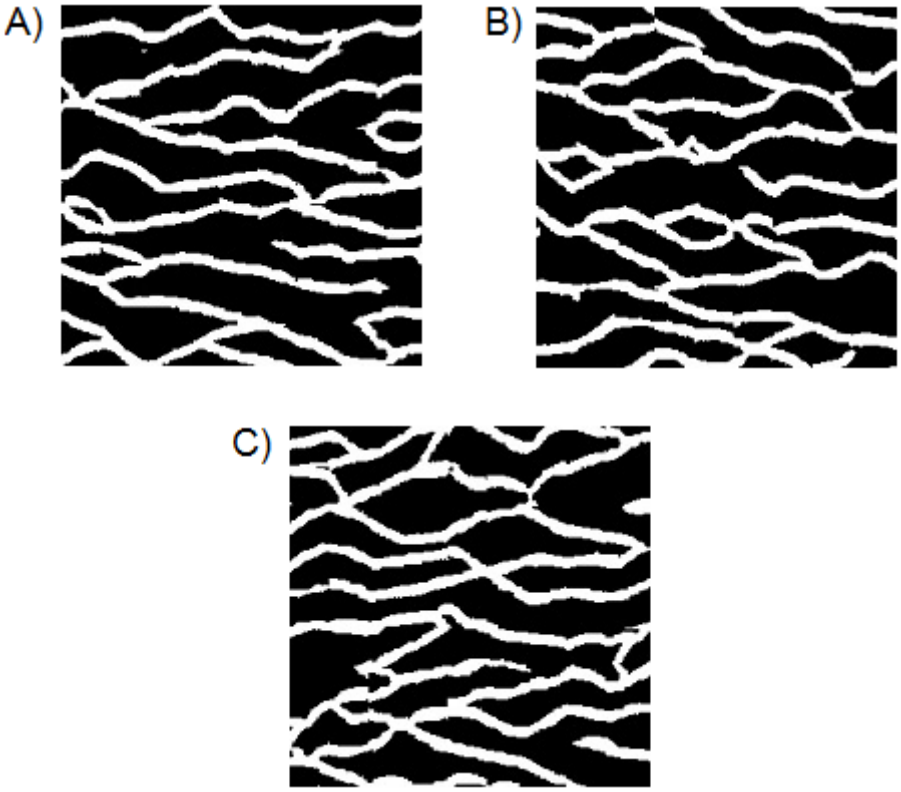

Figure 5.17: Three conditional realizations for the Strebelle TI honoring the conditioning points from Figure 5.16.

5.2.5 Sensitivity Analysis

In this section, we perform a sensitivity analysis of LSHSIM concerning the following parameters: α, L and K.

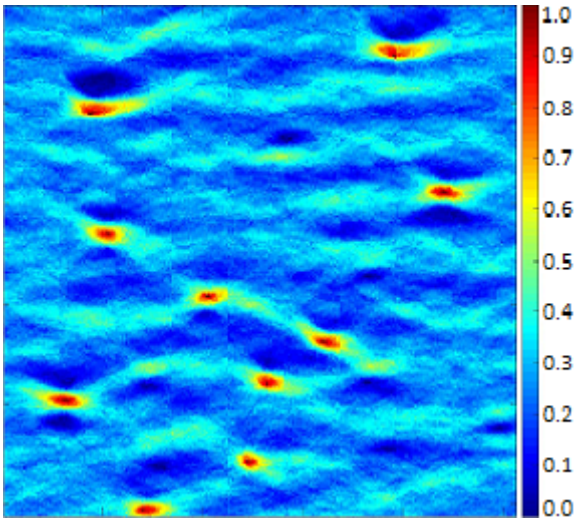

Figure 5.18: Ensemble average obtained for 100 conditional realizations.

Methodology. In order to better understand the effect of each parameter, three different test batches were performed. In each of them, two of the parameters were fixed to the values reported in the last subsections ($L = 30$; $K = 10$; $\alpha = 0.5\%$) and the remaining one varied in a range of predefined values, which will be described in the next subsections.

We generated 20 simulations for every configuration so as to obtain the average realization time and have more statistically consistent data. All experiments used the Strebelle TI, adopting a realization of 256×256 pixels, pattern size of 32×32 and overlap of 4.

Parameter α. This parameter, which is a value between 0 and 100, serves as a threshold to limit the size of the set of candidate patterns, that is, the set returned by the LSH structure, in which the (RLE based) search for a good/optimal pattern is performed. If $\alpha = U$ then at most $U\%$ of the patterns in the training image can be considered in this search. Thus, α has a great influence in the query time and, consequently, in the realization time.

In order to perform this experiment, we have chosen the following values for α: $\{0.05\%; 0.1\%; 0.3\%; 0.5\%; 1\%; 3\%; 5\%; 10\%\}$. Figures 5.19 and 5.20 exhibit the realization time in milliseconds and number of candidates obtained, respectively, for the predefined values of α. As expected, the larger the value of α the larger is the realization time and the number of candidates considered in the search. This parameter, in contrast with L and K, has no influence in the preprocessing time.

With respect to realization's quality, Figure 5.21 shows some examples of realizations for different values of α. There is a noticeable decrease in quality as α assumes lower values. This is explained by the fact that less candidate

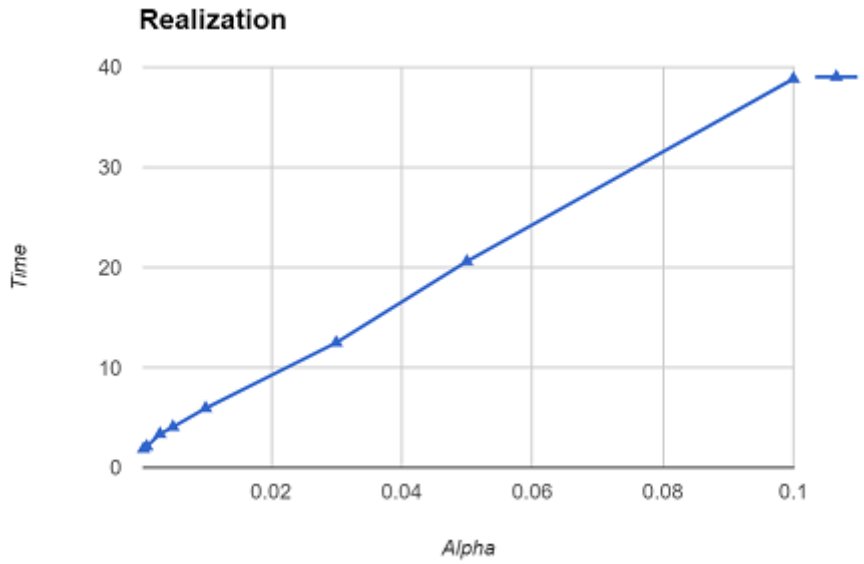

Figure 5.19: Realization time in milliseconds obtained as α varies.

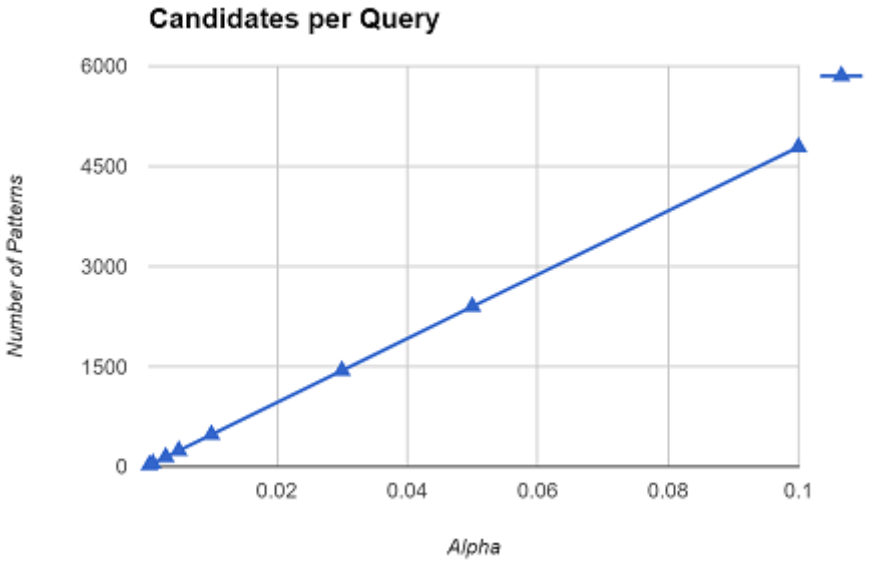

Figure 5.20: Number of candidates per query obtained as α varies.

patterns are considered and consequently a worse pattern can be chosen. On the other hand, as alpha increases, the quality of produced realizations gets better, since a larger number of patterns is considered as candidates and we tend to choose a good one. Finally, setting $\alpha = 0.5\%$ seems to be a good balance between quality and time.

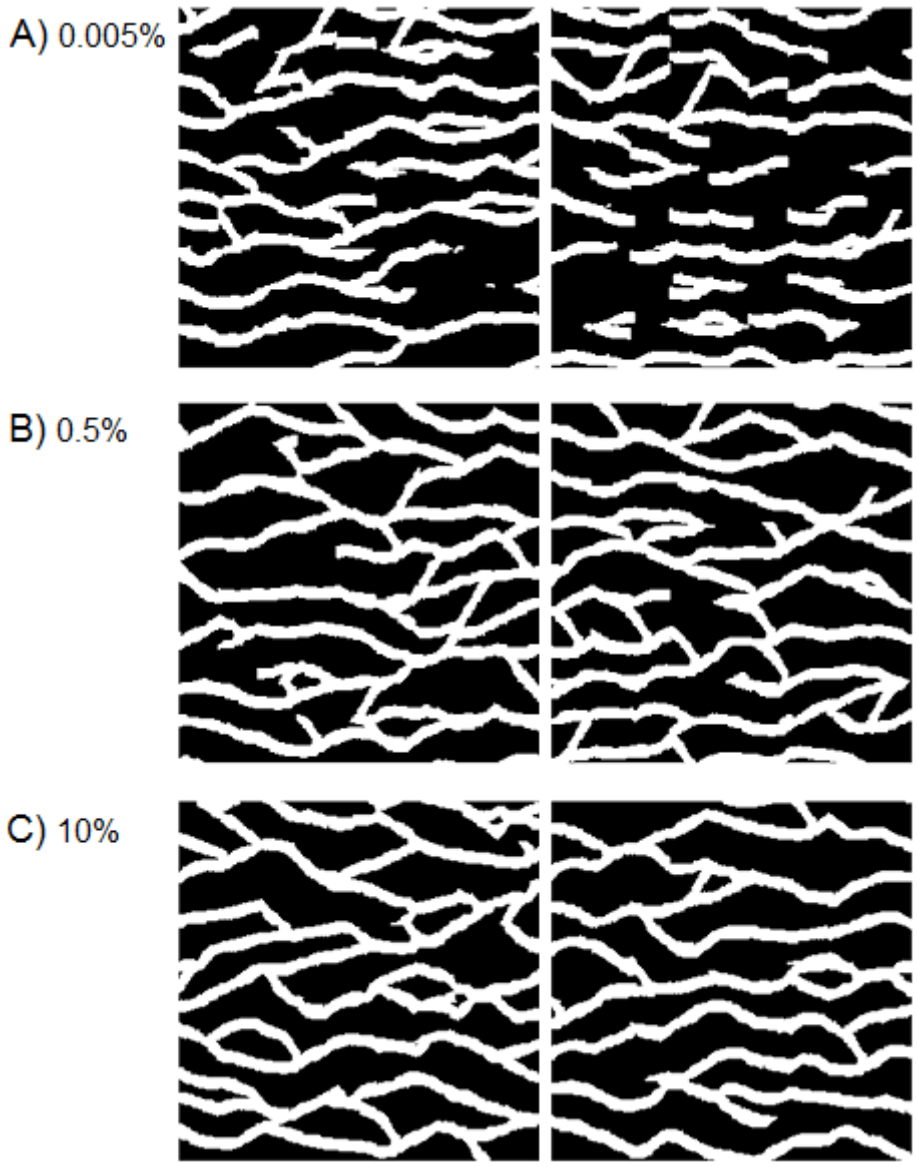

Figure 5.21: Examples of realizations performed setting α to the following values: 0.05% (A), 0.5% (B) and 10% (C).

Parameter L**.** The L parameter represents the number of hash tables used in the LSH structure. When searching for a pattern similar to a given data event, each one of the L hash tables are accessed and the candidates retrieved from each of them are joined to form the candidate set, in which the search

for a good pattern is executed. In order to test LSHSIM's sensitivity, we have chosen the following values for L: $\{1; 10; 15; 20; 25; 30; 35; 40; 45; 50\}$.

Figures 5.22, 5.23 and 5.24 illustrate the preprocessing time in milliseconds, realization time in milliseconds and number of candidates obtained, respectively, for simulations performed with the predefined values of L.

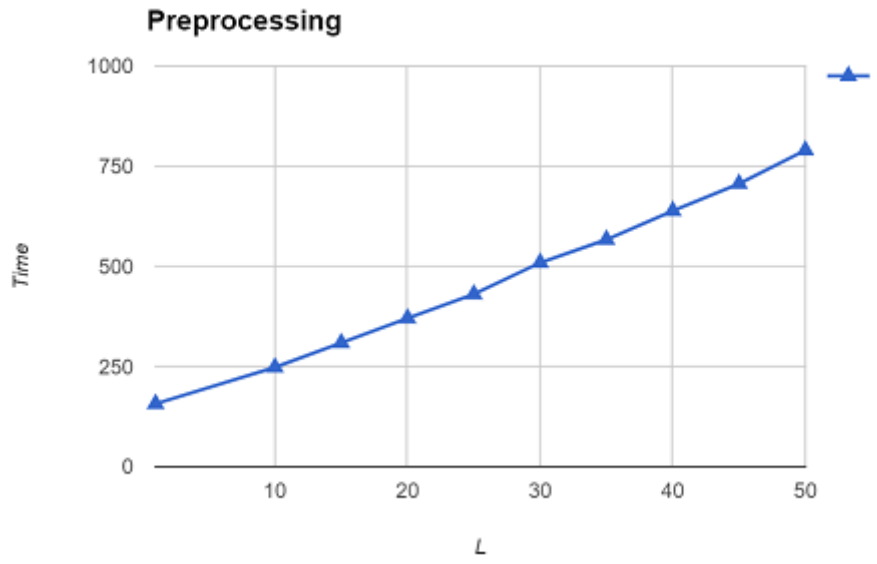

Figure 5.22: Preprocessing time in milliseconds obtained as L varies.

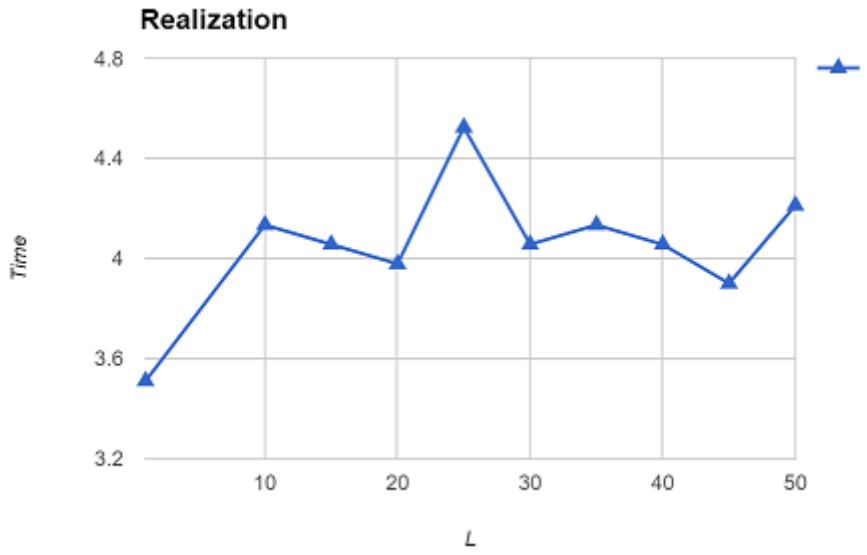

Figure 5.23: Realization time in milliseconds obtained as L varies.

As expected, the preprocessing time increases with the growth of L since more hash tables have to be created and populated. With respect to realization times, no clear tendency can be observed, supposedly indicating that no direct relation must exist here. However, it can be also noticed that the number of

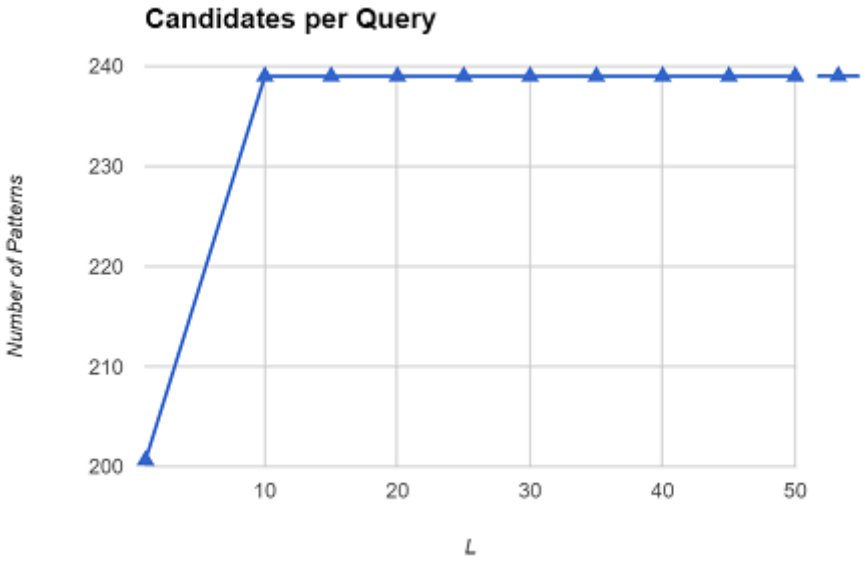

Figure 5.24: Number of candidates per query obtained obtained as L varies.

candidates found reaches a limit, which is exactly the restriction imposed by the α parameter. This explains the lack of tendency.

To have a better understanding of the effect of L on the realization time, the results for a bigger α (10%) are shown in Figure 5.25 and Figure 5.26. The motivation is to minimize the impact of the α parameter in the algorithm. In this new scenario, it can be observed that the realization time increases with the growth of L and then it stabilizes for L around 20. This happens, once again, due to the constraint imposed by parameter α.

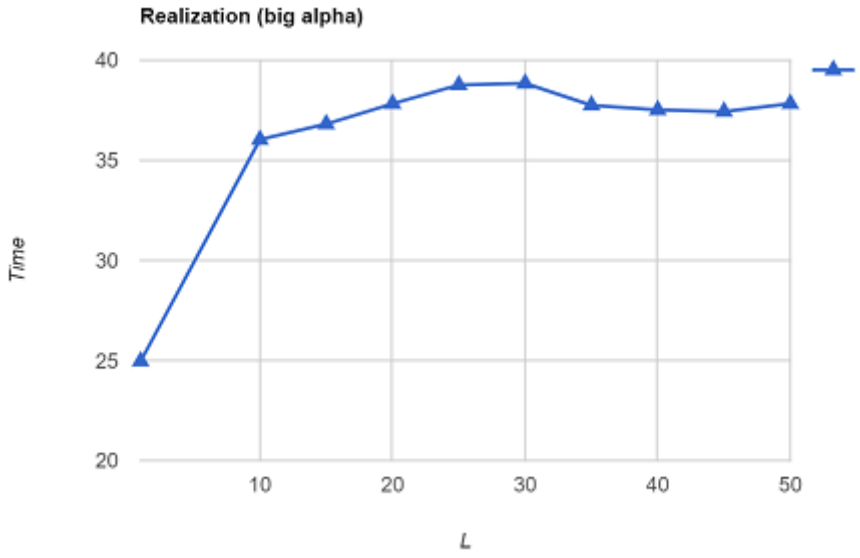

Figure 5.25: Realization time in milliseconds obtained as L varies having $\alpha = 10\%$.

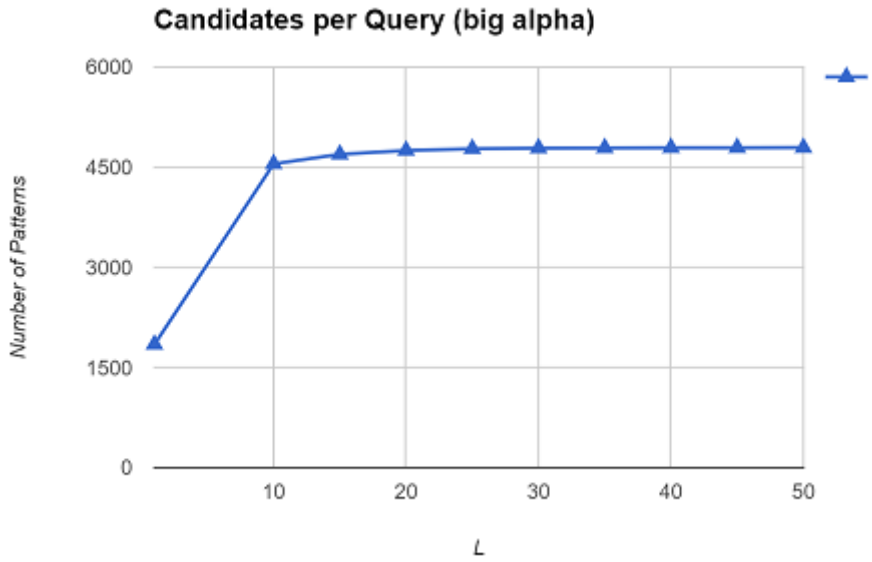

Figure 5.26: Number of candidates per query obtained as L varies having $\alpha = 10\%$.

Figure 5.27 shows some examples of realizations for three different values of L. It can be observed a gain in realization's quality as L is increased. This is not surprising since the set of candidates gets bigger when L gets larger. Finally, $L = 30$ seems to provide a good balance between realization's quality and computational time.

Parameter K. In the LSH structure, K represents the number of values from the overlap region that are used in the similarity's calculation.

Figures 5.28, 5.29 and 5.30 exhibit the preprocessing time in milliseconds, realization time in milliseconds and number of candidates obtained, respectively, for simulations performed with the predefined values of K.

Again, as one could expect, the preprocessing time increases with the growth of K due to the extra work to insert each one of the TI patterns in the hash tables of the LSH structure. Regarding the realization time no clear trend can be observed. The α parameter is once again having an impact on these times since it is limiting the number of candidates. Note that number of candidates for all values of K in Figure 5.30 are very close despite of the shape of the curve.

In order to have a better understanding and minimize the effect of α in the simulations, the results for a bigger value of α (10%) are presented in Figures 5.31 and 5.32. In these results, the limit imposed by α affects the number of candidates for K smaller or equal than 10. Beyond this value, the number of candidate patterns is not affected by α, since LSH becomes more selective and returns a smaller quantity of patterns.

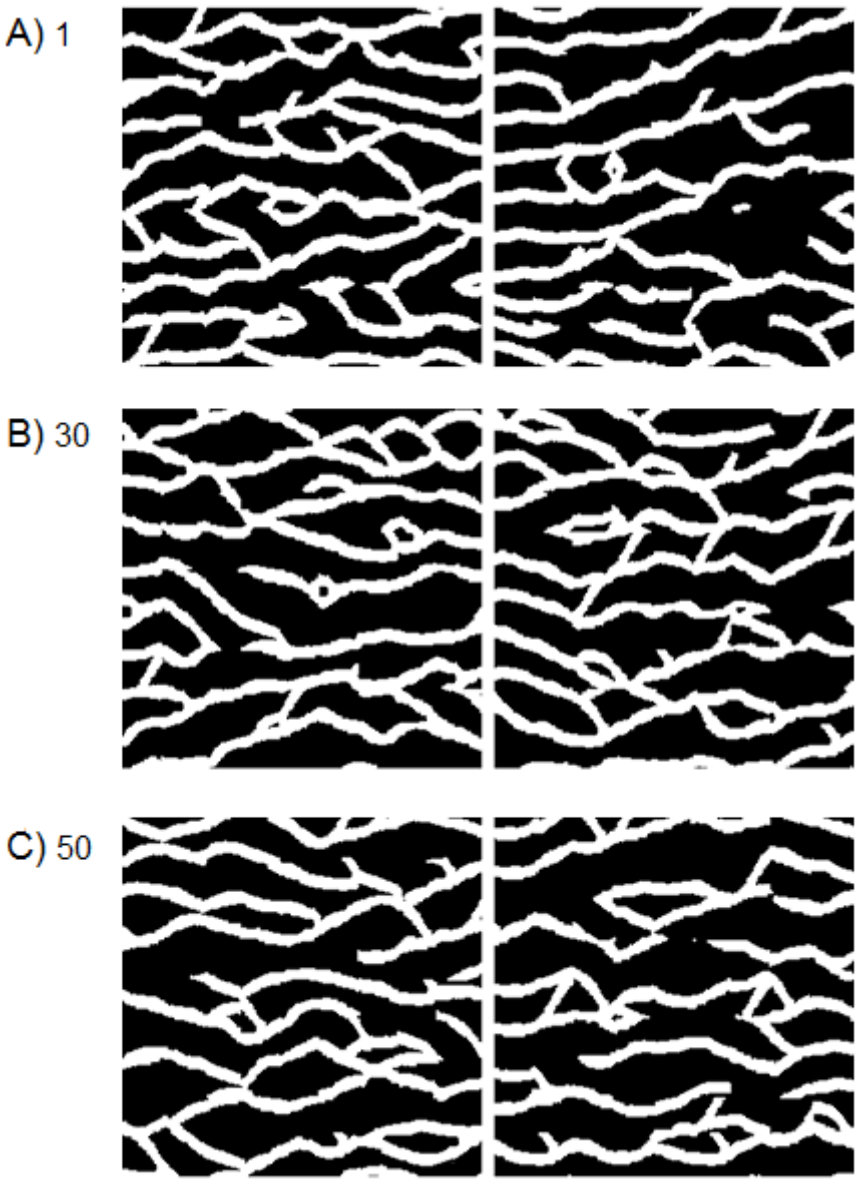

Figure 5.27: Examples of realizations performed setting L to the following values: 1 (A), 30 (B) and 50 (C).

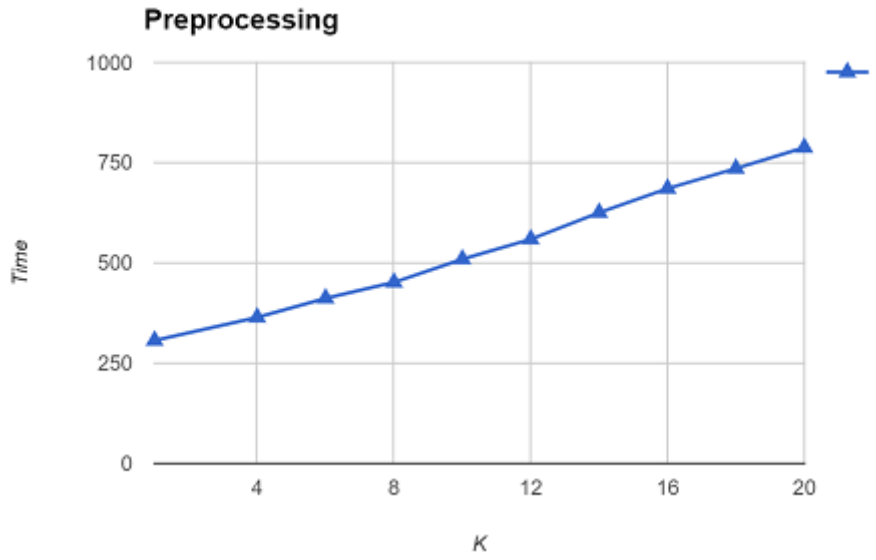

Figure 5.28: Preprocessing time in milliseconds obtained as K varies.

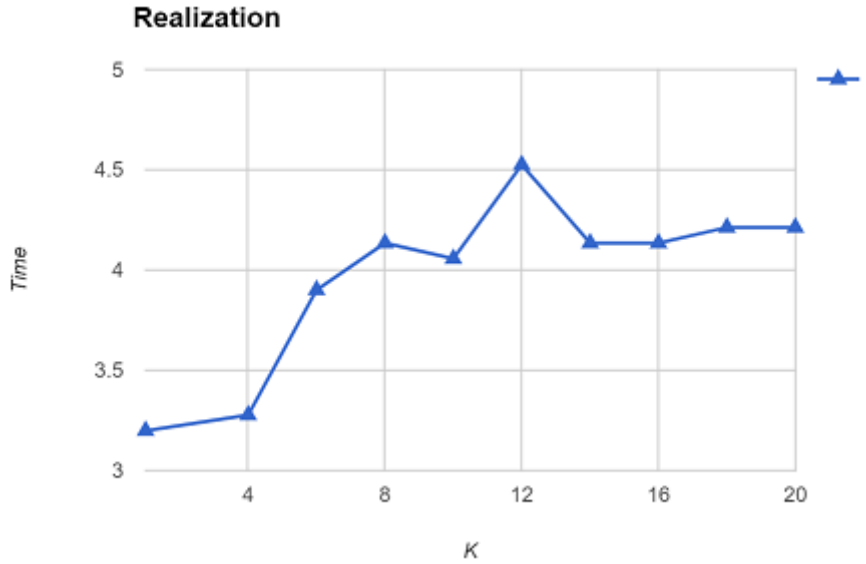

Figure 5.29: Realization time in milliseconds obtained as K varies.

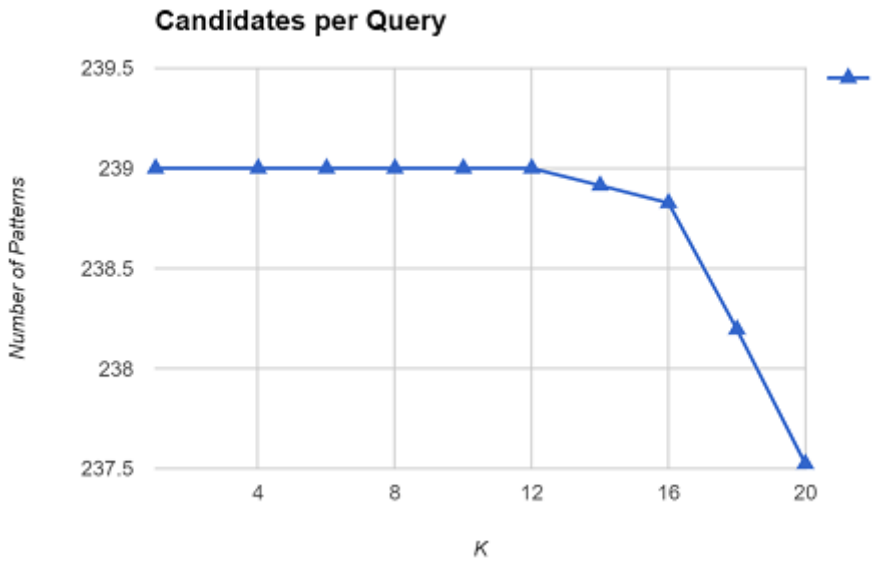

Figure 5.30: Number of candidates per query obtained obtained as K varies.

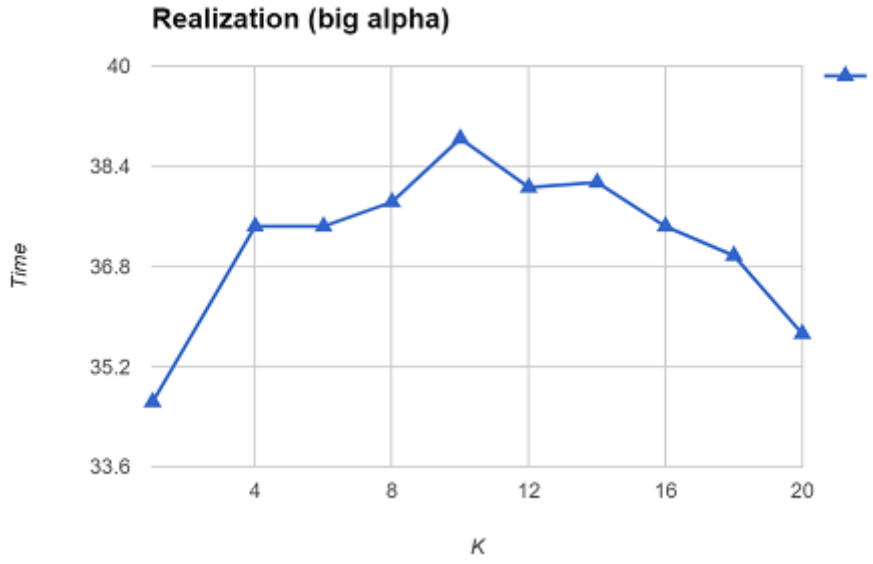

Figure 5.31: Realization time in milliseconds obtained as K varies having $\alpha = 10\%$.

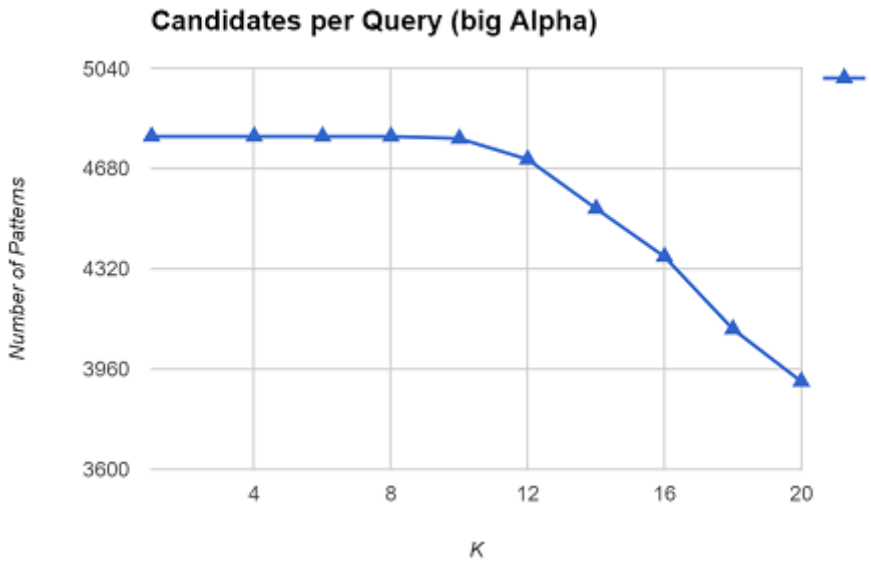

Figure 5.32: Number of candidates per query obtained as K varies having $\alpha = 10\%$.

Figure 5.33 shows some examples of realizations for three different values of K. For lower values of K, the quality of realizations is poor since LSH cannot distinguish well between patterns, thus being less selective. On the other hand, for higher values of K, LSH returns more refined results. Finally, $K = 10$ generates good quality realizations in a reasonable computational time.

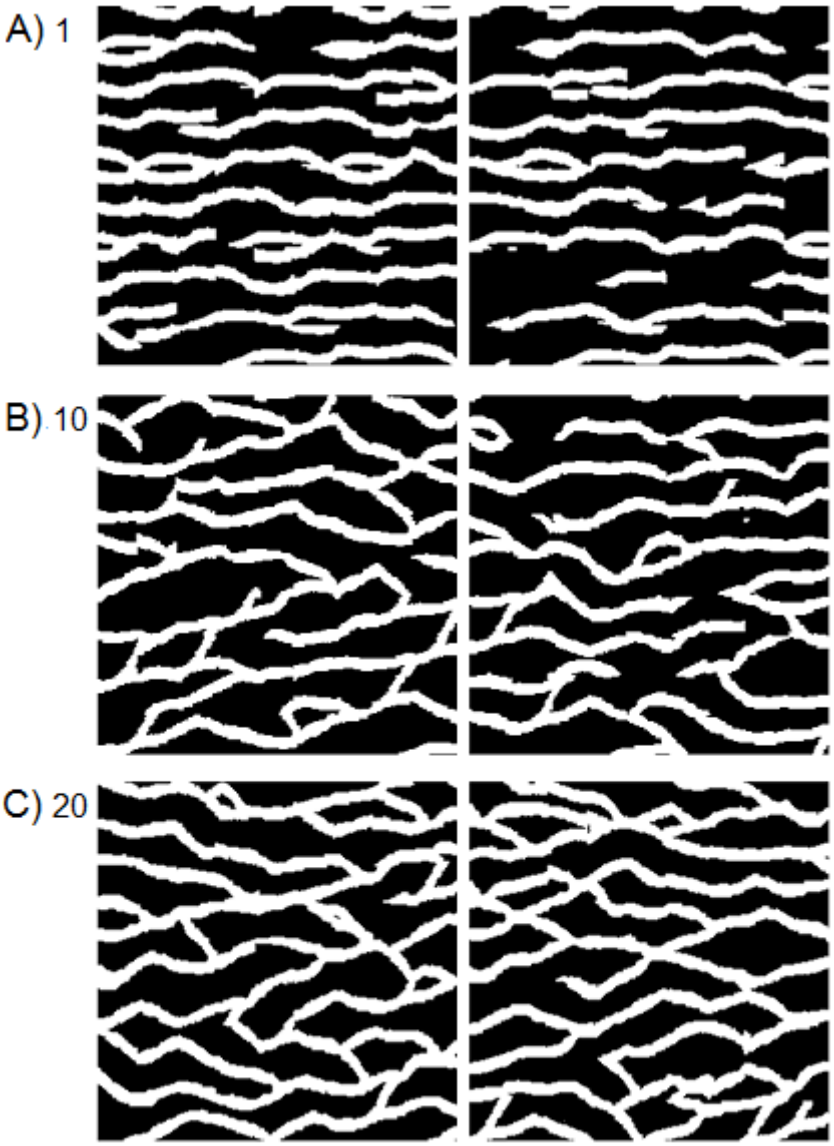

Figure 5.33: Examples of realizations performed setting K to the following values: 1 (A), 10 (B) and 20 (C).

6
Conclusions

6.1
Final Considerations

Multiple-point simulation has proved to be a very important methodology that provides a variety of techniques to model reservoirs and simulate possible scenarios. In this sense, motivated by reducing the time taken to generate realizations, the methods evolved from a pixel-based approach, which were typically slow and presented bad quality realizations, to pattern-based ones, dealing with a group of pixels in each step. Pattern-based methods use specialized techniques to cope with the dimensionality of patterns in an efficient way.

In this thesis, we first addressed the problem of searching for the most similar patterns when performing a realization. For this purpose, we proposed the use of compression techniques to accelerate the computation of convolutions. We have performed an investigation of the potential of RLE and LZ based methods for efficiently calculating convolutions of patterns of a fixed size and a given image, proposing new methods and variants of existing ones. Our experimental study indicated that the RLE method, the fastest one, outperformed a highly optimized implementation of the FFT algorithm for patterns up to a certain size.

On top of everything, we presented LSHSIM, a new and fast method to generate realizations that are based on the characteristics of a given training image. The method introduces new ideas to accelerate the simulation process such as the use of the LSH technique and the RLE based similarity computation. Experiments carried over a set of 7 selected TIs indicate that LSHSIM is almost one order of magnitude faster than MS-CCSIM for categorical images. In addition, the quality of our realizations is competitive with those generated by MS-CCSIM, in the sense that the spatial continuity of the TIs was well expressed. Our MDS plots depicted that LSHSIM's space of uncertainty has a good spread and the realizations are close to the TI.

To assert the extensibility of LSHSIM to continuous data, we have applied our method to a continuous TI and obtained good results regarding time and

quality of simulations. In this case, the RLE approach shall not be used and the LSH scheme should be based on the Euclidean distance instead of the Hamming distance. Besides, we also extended the LSHSIM method to 3D TIs. In this sense, we have applied LSHSIM for some selected 3D TIs and the obtained results were also satisfying, taking into consideration the time and quality of the realizations produced.

6.2 Future Works

The following topics are considered as interesting future works:

1. An ongoing research consists of adapting LSHSIM to work with a new concept called cost matrix, which is a more general way of modeling the similarity between patterns. These matrices express, for each possible pair of facies i and j, the cost of replacing a facie i from a data event with a facie j from a pattern, or vice-versa. Note that the purpose is to let a specialist, such as a geologist, define its values according to a specific semantic of the region being studied. Therefore, we aim to explore how to integrate the cost matrix into the LSH approach;

2. Integrate the hard data concept into the LSH approach, such that the adopted similarity measure prioritizes these data when filtering patterns in a search;

3. Explore the approach proposed by Lv et al. (2007) in the multi-probe LSH, which permits to use a fewer number of hash tables, when dealing with applications which contain space constraints. To achieve this, when performing a search, the method applies a perturbation scheme to the query q, so as to probe other buckets from the same hash table. The property of LSH guarantees that objects which are close to q, but not hashed to the same bucket as q, are likely to be in a bucket "close by". The method then aims to find these buckets through the application of perturbations to q;

4. Investigate the use of the LSH technique in the GOSIM method (Yang et al., 2016), instead of PatchMatch algorithm, such as discussed in Section 2.1.2;

5. Compare LSHSIM's performance with the recently published work of Hoffimann et al. (2017), which proposes a new strategy to the problem and claims to be faster than any other MPS algorithm published so far.

Bibliography

Abdollahifard, M. J. (2016), 'Fast multiple-point simulation using a data-driven path and an efficient gradient-based search', *Comput. Geosci.* **86**(C), 64–74. DOI 10.1016/j.cageo.2015.10.010. ISSN 0098-3004.

Abdollahifard, M. J. & Nasiri, B. (2017), 'Exploiting transformation-domain sparsity for fast query in multiple-point geostatistics', *Computational Geosciences* **21**(2), 289–299. DOI 10.1007/s10596-016-9612-1. ISSN 1573-1499.

ANODI (2016), 'Matlab code of the anodi method', https://github.com/SCRFpublic/ANODI. Accessed: 2016-04-10.

Arpat, G. B. & Caers, J. (2007), 'Conditional simulation with patterns', *Mathematical Geology* **39**(2), 177–203. DOI 10.1007/s11004-006-9075-3. ISSN 1573-8868.

Barnes, C., Shechtman, E., Finkelstein, A. & Goldman, D. B. (2009), 'Patchmatch: A randomized correspondence algorithm for structural image editing', *ACM Trans. Graph.* **28**(3), 24:1–24:11. DOI 10.1145/1531326.1531330. ISSN 0730-0301.

Borg, I. & Groenen, P. (2005), *Modern Multidimensional Scaling: Theory and Applications*, 2 ed., Springer.

Caers, J. (2002), 'Geostatistical history matching under training-image based geological model constraints', *Society of Petroleum Engineers*.

Caers, J. (2011), *Modeling Uncertainty in the Earth Sciences*, Wiley.

Chilès, J. P. & Delfiner, P. (2012), *Geostatistics: Modeling Spatial Uncertainty*, 2 ed., Wiley, Hoboken, New Jersey.

Cooley, J. M. & Tukey, J. W. (1965), 'An algorithm for the machine calculation of complex fourier series', *Math. Comp.* **19**, 297.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009), *Introduction to Algorithms, Third Edition*, 3rd ed., The MIT Press.

Efros, A. A. & Freeman, W. T. (2001), 'Image quilting for texture synthesis and transfer', *Proceedings of SIGGRAPH 2001* pp. 341–346.

FFTW (2015), 'Fastest fourier transform in the west', http://www.fftw.org. Accessed: 2015-10-06.

Freschi, V. & Bogliolo, A. (2010), 'A faster algorithm for the computation of string convolutions using LZ78 parsing', *Inf. Process. Lett* **110**(14-15), 609–613.

Frigo, M. & Johnson, S. G. (2005), 'The design and implementation of FFTW3', *Proceedings of the IEEE* **93**(2), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

Gardet, C., Le Ravalec, M. & Gloaguen, E. (2016), 'Pattern-based conditional simulation with a raster path: a few techniques to make it more efficient', *Stochastic Environmental Research and Risk Assessment* **30**(2), 429–446. DOI 10.1007/s00477-015-1207-1. ISSN 1436-3259.

Gionis, A., Indyk, P., Motwani, R. et al. (1999), Similarity search in high dimensions via hashing, *in* 'Proceedings of the International Conference on Very Large Data Bases', number 6, pp. 518–529.

Guardiano, F. B. & Srivastava, R. M. (1993), *Geostatistics Tróia '92: Volume 1*, Springer Netherlands, Dordrecht, chapter Multivariate Geostatistics: Beyond Bivariate Moments, pp. 133–144.

Hamming, R. (1950), 'Error Detecting and Error Correcting Codes', *Bell System Technical Journal* **29**, 147–160.

Hassanieh, H., Indyk, P., Katabi, D. & Price, E. (2012), 'Simple and practical algorithm for sparse fourier transform', *SODA* pp. 1183–1194.

Hoffimann, J., Scheidt, C., Barfod, A. & Caers, J. (2017), 'Stochastic simulation by image quilting of process-based geological models', *Computers & Geosciences* **106**, 18 – 32. DOI http://dx.doi.org/10.1016/j.cageo.2017.05.012. ISSN 0098-3004.

Honarkhah, M. & Caers, J. (2010), 'Stochastic simulation of patterns using distance-based pattern modeling', *Mathematical Geosciences* **42**, 487–517.

Indyk, P. & Motwani, R. (1998), Approximate nearest neighbors: towards removing the curse of dimensionality, *in* 'Proceedings of the thirtieth annual ACM symposium on Theory of computing', ACM, pp. 604–613.

Leskovec, J., Rajaraman, A. & Ullman, J. D. (2014), *Mining of Massive Datasets*, 2nd ed., Cambridge University Press, New York, NY, USA.

Lv, Q., Josephson, W., Wang, Z., Charikar, M. & Li, K. (2007), Multi-probe lsh: Efficient indexing for high-dimensional similarity search, *in* 'Proceedings of the 33rd International Conference on Very Large Data Bases', VLDB '07, VLDB Endowment, pp. 950–961.

Mahmud, K., Mariethoz, G., Caers, J., Tahmasebi, P. & Baker, A. (2014), 'Simulation of earth textures by conditional image quilting', *Water Resources Research* **50**(4), 3088–3107. DOI 10.1002/2013WR015069. ISSN 1944-7973.

Mariethoz, G. & Caers, J. (2014), *Multiple-point Geostatistics: Stochastic Modeling with Training Images*, 1 ed., Wiley-Blackwell.

Mariethoz, G. & Lefebvre, S. (2014), 'Bridges between multiple-point geostatistics and texture synthesis: Review and guidelines for future research', *Computers & Geosciences* **66**, 66 – 80. DOI http://dx.doi.org/10.1016/j.cageo.2014.01.001. ISSN 0098-3004.

Mariethoz, G., Renard, P. & Straubhaar, J. (2010), 'The direct sampling method to perform multiple-point geostatistical simulations', *Water Resources Research* **46**(11), 1–14. DOI 10.1029/2008WR007621. ISSN 1944-7973, W11536.

MS-CCSIM (2016), 'Matlab code of the ms-ccsim method', https://github.com/SCRFpublic/MS_CCSIM. Accessed: 2016-04-10.

OPENCV (2016), 'Opencv - open source computer vision', http://opencv.org. Accessed: 2016-07-15.

Parra, Á. & Ortiz, J. M. (2011), 'Adapting a texture synthesis algorithm for conditional multiple point geostatistical simulation', *Stochastic Environmental Research and Risk Assessment* **25**(8), 1101–1111. DOI 10.1007/s00477-011-0489-1. ISSN 1436-3259.

Pyrcz, M. J. & Deutsch, C. V. (2014), *Geostatistical Reservoir Modeling*, Oxford university press.

Rytter, W. (2003), 'Application of Lempel-Ziv factorization to the approximation of grammar-based compression', *Theoretical Computer Science* **302**, 211–222.

Simard, P. Y., Bottou, L., Haffner, P. & LeCun, Y. (1998), Boxlets: A fast convolution algorithm for signal processing and neural networks, *in* 'NIPS', pp. 571–577.

Strebelle, S. (2002), 'Conditional simulation of complex geological structures using multiple-point statistics', *Mathematical Geology* **34**(1), 1–21. DOI 10.1023/A:1014009426274. ISSN 1573-8868.

Tahmasebi, P. & Sahimi, M. (2016*a*), 'Enhancing multiple-point geostatistical modeling: 1. graph theory and pattern adjustment', *Water Resources Research* **52**, 2074–2098. DOI 10.1002/2015WR017806. ISSN 1944-7973.

Tahmasebi, P. & Sahimi, M. (2016*b*), 'Enhancing multiple-point geostatistical modeling: 2. iterative simulation and multiple distance function', *Water Resources Research* **52**(3), 2099–2122. DOI 10.1002/2015WR017807. ISSN 1944-7973.

Tahmasebi, P., Hezarkhani, A. & Sahimi, M. (2012), 'Multiple-point geostatistical modeling based on the cross-correlation functions', *Computational Geosciences* **16**(3), 779–797. DOI 10.1007/s10596-012-9287-1. ISSN 1573-1499.

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

Tan, X., Tahmasebi, P. & Caers, J. (2014), 'Comparing training-image based algorithms using an analysis of distance', *Mathematical Geosciences* **46**(2), 149–169. DOI 10.1007/s11004-013-9482-1. ISSN 1874-8953.

Tanaka, T., I, T., Inenaga, S., Bannai, H. & Takeda, M. (2013), Computing convolution on grammar-compressed text, *in* A. Bilgin, M. W. Marcellin, J. Serra-Sagristà & J. A. Storer, eds, '2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013', IEEE, pp. 451–460.

TrainingImagesLibrary (2016), 'Training images library', http://www.trainingimages.org/training-images-library.html. Accessed: 2016-03-02.

Vazirani, V. V. (2001), *Approximation algorithms*, Springer.

Werman, M. (2003), Fast convolution, *in* 'WSCG'.

Yang, L., Hou, W., Cui, C. & Cui, J. (2016), 'Gosim: A multi-scale iterative multiple-point statistics algorithm with global optimization', *Comput. Geosci.* **89**, 57 – 70.
DOI http://dx.doi.org/10.1016/j.cageo.2015.12.020. ISSN 0098-3004.

Zhang, T., Switzer, P. & Journel, A. (2006), 'Filter-based classification of training image patterns for spatial simulation', *Mathematical Geology* **38**(1), 63–80. DOI 10.1007/s11004-005-9004-x. ISSN 1573-8868.

Ziv, J. & Lempel, A. (1978), 'Compression of individual sequences via variable rate encoding', *IEEE Trans. Inf. Theory* pp. 530–536.