Rozprawy
Monografie 345

345 ROZPRAWY
MONOGRAFIE

PAWEŁ TOPA

Automat komórkowy jako wydajny i elastyczny schemat obliczeniowy na potrzeby modelowania i symulacji rzeczywistych, wielkoskalowych zjawisk złożonych

WYDAWNICTWA AGH

KRAKÓW 2019

DISSERTATIONS
MONOGRAPHS 345

PAWEŁ TOPA

Cellular Automata as an efficient and flexible computational framework for modeling and simulation of real-world, multiscale complex systems

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY PRESS KRAKOW 2019

Published by AGH University of Science and Technology Press

Author of the monograph is an employee of
AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30
30-059 Krakow, Poland

Desktop publishing: *Paweł Topa*
Technical editor: *Joanna Ciągała*

ISBN 978-83-66016-63-7
ISSN 0867-6631

Wydawnictwa AGH (AGH University of Science and Technology Press)
al. A. Mickiewicza 30, 30-059 Kraków
tel. 12 617 32 28, 12 636 40 38
e-mail: redakcja@wydawnictwoagh.pl
www.wydawnictwa.agh.edu.pl

Contents

PAWEŁ TOPA

Cellular Automata as an efficient and flexible computational framework for modeling and simulation of real-world, multiscale complex systems

Abstract

Computer simulation is now a fully-fledged scientific method. It is especially useful for systems or phenomena that have the property of complexity. It means that it is impossible to deduct the behavior of the whole system from the analysis of its individual parts. It is a result of interactions and feedbacks (positive and negative) that occur between components. Such systems are called Complex Systems and the only reliable method to simulate them is tracking their evolution step-by-step by using models that directly represents all components. Intensive investigations into modeling methods were motivated by the need for new methods that are able to efficiently model a Complex System. The Cellular Automata is a modeling and simulation method often used to simulate Complex Systems. It allows to decompose the system into set of interacting entities.

In this monograph, three issues related to the usage of this approach are considered. First, the possibilities of implementing parallel Cellular Automata models, especially for massively parallel architecture, such as a GPU, are investigated. Two Cellular Automata models are used as a case study: water flow and pedestrian dynamics. The researches include various strategies for constructing parallel models and their algorithms.

Another issue, investigated by the Author, is multiscality which is difficult to represent using the original Cellular Automata approach. Thus, a new extension is proposed which is capable of encompassing processes occurring in different spatio-temporal scales. Efficiency and universality of this new modeling tool is demonstrated and discussed using natural phenomena like a vascular system and a mycelium.

The last issue is the ability to use the Cellular Automata not only as a specific modeling tool but also as a flexible and efficient modeling framework. This approach was tested using a model of evolution and population dynamics. The model uses agent-based methodology to represent individuals, their behavior and life strategy

and the Cellular Automata represents the agents' habitat. Additionally, the Cellular Automata is used as a framework to organize the processing of agents. In particular, it allows to employ tested and efficient architectures of parallel computations. The monograph ends with an outline of the most important contributions and concluding remarks summarizing the results.

PAWEŁ TOPA

Automat komórkowy jako wydajny i elastyczny schemat obliczeniowy na potrzeby modelowania i symulacji rzeczywistych, wieloskalowych zjawisk złożonych

Streszczenie

Symulacja komputerowa jest obecnie uważana za pełnoprawną metodę naukową. Jest ona szczególnie przydatna w przypadku systemów lub zjawisk, które określamy mianem złożonych. Termin ten oznacza, że nie można przewidzieć zachowania całego systemu na podstawie analizy wyłącznie jego poszczególnych części, a jest ono rezultatem interakcji i sprzężeń zwrotnych, które zachodzą między elementami składowymi. Tego typu układy określane są mianem systemów złożonych (ang. *Complex Systems*) i najskuteczniejszą metodą ich symulacji jest odtworzenie ich ewolucji krok po kroku za pomocą modelu komputerowego. Potrzeba stosowania wiernej i wydajnej symulacji systemów złożonych stanowi motywację do prowadzenia intensywnych prac nad nowymi narzędziami modelowania. Automat komórkowy jest często stosowany w przypadku modelowania i symulacji systemów złożonych. Ta metoda pozwala przedstawić system w postaci zbioru elementów, które oddziałują ze sobą zgodnie ze zdefiniowanymi regułami.

W niniejszej monografii rozważane są trzy zagadnienia związane z wykorzystaniem automatu komórkowego. Analizowane są kwestie równoległych implementacji modeli, w szczególności dla architektur masywnie równoległych, np. procesorów graficznych (GPU). Jako przykład efektywnego wykorzystania tej architektury przedstawiono dwa modele oparte na metodzie automatu komórkowego: model przepływ wody i dynamiki pieszych. Rozważano w tym przypadku różne strategie konstruowania modeli i ich algorytmów.

Następnym zagadnieniem, jakie zostało poddane analizie, jest możliwość modelowania zjawisk wieloskalowych. Ich symulacja za pomocą tradycyjnie rozumianego automatu komórkowego jest trudna lub niemożliwa. Konieczne jest wprowadzenie rozszerzeń, które umożliwią uchwycenie w jednym modelu procesów zachodzących w różnych skalach przestrzenno-czasowych. Wydajność i uniwersalność tego nowego narzędzia jest demonstrowana i omawiana w zastosowaniu do trzech przykładowych zjawisk naturalnych: rzeki anastomozującej, układu krwionośnego oraz grzybni.

Ostatnie zagadnienie to możliwość wykorzystania automatu komórkowego nie tylko jako konkretnej metody modelowania i symulacji, ale także jako elastycznego i wydajnego schematu konstruowania i implementacji modeli komputerowych opartych na innych metodologiach. Koncepcja ta została przetestowana w modelu ewolucji i dynamiki populacji wykorzystującego paradygmat agentowy. Proponowane podejście zakłada, że automat komórkowy nie tylko modeluje habitat agentów, ale także organizuje przetwarzania agentów. W szczególności umożliwia to zastosowanie istniejących i przetestowanych schematów obliczeń równoległych i rozproszonych. Monografię kończy podsumowanie najistotniejszych elementów pracy i osiągniętych wyników.

Preface

The computer simulation is now one of the most important scientific methods. By analogy to experimental methods used in biology, *in vivo* and *in vitro*, the term *in computo* is promoted by many experts. This "third paradigm" of scientific reasoning (one out of the recognized four: experiment, theory, computer modeling, data modeling) is especially applicable when the spatio-temporal evolution of an investigated phenomenon at particular moment of time cannot be calculated simply by solving an equation (or a system of equations). In systems consisting of many interacting parts, positive and negative feedbacks between these parts lead to emergent behavior. Such systems are called Complex Systems and in this monograph they are discussed in more detail in Chapter 1. For this type of systems, the only method of modeling is computer simulation where the evolution of phenomenon is tracked step-by-step — this property is called computational irreducibility [1, 2].

There are dozens of modeling and simulation methods that assume that basic components of the phenomenon are represented directly. The classical examples of such an approach are methods based on N-body dynamics, including molecular dynamics and other particle methods (Dissipative Particle Dynamics, Fluid Particle Models, etc.). They assume that a complex system is represented by virtual particles that move within the simulation domain, interact with each other according to some potentials and rules of collisions. The idea that lies behind these models is very rudimentary and intuitive, but their efficient implementation is challenging. It is mainly caused by the necessity of calculating in each time-step which pairs of particles interact. The other methodologies in which particles or other interacting objects are pinned or may occupy-only defined places (grid) can undoubtedly surpass the performance of the particle models.

The Cellular Automata was proposed first by John von Neumann (with support from Stanislaw Ulam) to model a population of self-replicating entities [3]. He was motivated by the desire to understand the basic principles of information processing that underlie self-replication, the potential long-term use of programmable self-replicating machines, and the possibility of getting insights into biological replication

and the origins of life. Von Neumann focused on constructing an optimal set of rules which result in such behavior. Also, the most famous Cellular Automata — the Game of Life by John Conway — represents an abstract system composed of entities that are born, live and die according to a very simple set of rules [4]. Further investigations also focused on searching for rules that produce complex behavior. In this area the capabilities of the Cellular Automata are amazing, including the ability to perform computations — it was proven that Cellular Automata with special rules are equivalent to the Universal Turing Machine [5].

The Game of Life and similar Cellular Automata rules might be treated as an idealization of some specific phenomena (i.e. a population of individuals) but they were not developed with some specific natural phenomena in mind. Obviously, the bridge between some abstract rule and the real phenomena can be established with some assumptions. The Rule 184 (using Wolfram code for Elementary Cellular Automata [6]) can be used as a simple model of traffic flow in a single lane of a highway. However, the same rule can be used to model deposition of particles onto a surface and to model ballistic annihilation. Rules similar to 184 arouse admiration due to their simplicity connected with great abilities, but for real phenomena they must be extended to include processes and factors that are specific for a given phenomenon. For example, the road traffic model (single lane without junctions) must include at least acceleration and deceleration of cars. Valuable and reliable results can be provided by the model that, apart the core processes, include additional factors that tune the behavior of the modeled phenomena in a noticeable way.

From the point of view of implementation, Cellular Automata is a very convenient modeling method. Computations are performed on a regular, static grid within a static and short distance neighborhood. Evolution of cells is expressed in the form of rules that use as an input only states of cells from a small neighborhood. All these properties mean that even single-thread implementations are relatively efficient. Notwithstanding, the Cellular Automata are inherently parallel — all the cells should be updated at the same time and rules use the state of cells from the previous step only. In the past, architectures with distributed memory like clusters were the computational environment used most often. Static topology of computational nodes and a relatively small amount of data that were exchanged at each step of the computations provide satisfactory performance in this application. The introduction of graphic processing units (GPU) with programmed shader units provided an opportunity to bring new quality in processing Cellular Automata models. These processors were designed to process in parallel streams of data with the same structure (vertices and fragments) by applying short functions called shaders. Cells in the Cellular Automata can be processed in the same way. Initially, GPU programing for solving general problems (GPGPU) required low level programming with deep knowledge

of how processing units, schedulers and memory work. Fortunately, this platform developed very quickly and in subsequent generations of processors and programming platforms (i.e. CUDA provided by Nvidia) new features that facilitate programming are constantly introduced. Despite that, this architecture still requires programmers to construct algorithms in which the flow of instructions and synchronization are strictly controlled. The same rules applied to the Cellular Automata where proper algorithms are the key factor to achieve high performance.

Within the frames of their original definition, Cellular Automata have limited capabilities when applied to model natural phenomena. Some extensions and modifications are necessary. One of the most useful is using of continuous values to describe the state of cells. It allows one to represent the state of a system using parameters more closely related to some physical properties, i.e., density of some substances. Attempts to use Cellular Automata to model various phenomena induced the introduction of many extensions and modifications that facilitate capturing the specific properties or behavior of a particular phenomenon. Part of this monograph (see Chapter 1) is devoted to an extensive review of these modifications.

Many natural phenomena have the property of multiscality. They are composed of one or more components for which individual evolution occurs at different times and on different scales. Admittedly, it is possible to "bring all fractions to a common denominator" and define a model that will be sufficiently "fine grained" in terms of time and scale in order to cover all processes. This approach has several drawbacks. First of all, such a model will be extremely computationally demanding. In some cases, like in molecular dynamics, simulations, even using high performance computing, are able to cover only small pieces of matter over a very short period of time (10^{-9} – 10^{-12}). Additionally, due to incompleteness of our knowledge the expected macroscale behavior might not emerge from the microscale rules. Thus, building multiscale models involving different techniques is reasonable, but the biggest problem in constructing such models is bridging the gaps between different scales.

The Cellular Automata are a method that is able to simulate phenomena in single spatio-temporal scale defined by the size of cells and the length of time-steps. It is worth emphasizing that choosing the proper values of these parameters itself is not trivial and requires scaling of the process speed expressed in the form of a local interaction rule. In some models, it might be relatively easy, like in models of pedestrian dynamics where the size of cells and the time step can be chosen based on measuring the real speed of pedestrians. Unfortunately, in this case the homogeneity of Cellular Automata makes it difficult to differentiate the speed of pedestrians. The most straightforward solution is to differentiate the resolution of the grid in places or at moments where it is necessary. In most models that were proposed so far, the multiscality is implemented by coupling a few single-scale Cellular Automata that

mutually interact across the scales. Since the components of the phenomenon from different scales might have a very different structure and behavior, the choice of the Cellular Automata sub-models might be challenging.

In an effort to provide modeling methodologies and tools that are able to deal with the challenges mentioned above, the research activities were focused on three goals:

- **Extending the Cellular Automata methodology for modeling and simulation multiscale phenomena.**
- **Demonstrating that the Cellular Automata is an efficient and flexible methodology for modeling and simulation of complex systems.**
- **Introducing new algorithms for the Cellular Automata models optimized for stream processing in order to provide high efficiency on modern massively parallel architectures.**

In order to fulfill these goals, the following research tasks are reported in this monograph:

1. State-of-the-art review of Cellular Automata models, emphasizing extensions and modifications that were added to the original definition by von Neumann and Ulam.
2. Presentation and discussion of the Cellular Automata model proposed for the anastomosing river phenomenon. This model might be treated as classical since the only modification is the using continuous values to represent cell states.
3. Presentation and discussion of approaches and methodologies used to conduct large scale Cellular Automata simulations, including GPU (Graphic Processing Units) platforms.
4. Discussion on the multiscale properties of anastomosing river systems and introduction of a new extension to the Cellular Automata approach. The new modeling method combines Cellular Automata with graphs (called Graph of Cellular Automata), which makes it possible to model multiscale system composed of transportation network and consuming (or producing) environment.
5. Applying the Graph of Cellular Automata to phenomena that consist of a transportation network and consuming or producing environment: development of a vascular system (induced by carcinogenesis) and growth of a mycelium inside plant tissue.

6. Using Cellular Automata as a framework for large scale population models based on the paradigm of Individual Based Modeling. Members of populations are represented by agents while Cellular Automata model the environment and additionally organize the processing of agents.

The structure of this monograph is as follows. **Chapter 1** begins with an introduction the Complex Systems term. Since complexity is observed in every natural phenomena and process, the knowledge of properties of such systems is mandatory when modeling methods are considered. Thus, the main properties of complex systems are enumerated and briefly discussed. The nature of these system means that classical mathematical methods based on solving equations often fail. Therefore, other methodologies must be employed and Cellular Automata is the leading approach in this area. This chapter also contains a definition of the original Cellular Automata paradigm, in the form that was proposed by von Neumann and Ulam, and later used by John Conway in his famous Game of Life. The main properties and applications of this version of Cellular Automata are briefly presented and discussed. In the next part of this chapter, an extensive enumeration and discussion of various extensions and modifications is given. All these extensions were proposed by scientists in an attempt to express the specific properties of various phenomena by using the Cellular Automata approach. As a result, almost all components of the traditional Cellular Automata definition were modified. Based on this review, the common properties that were still preserved in all these model are enumerated. Finally, Cellular Automata are proposed as a modeling framework rather than a specific modeling method.

Chapter 2 presents the phenomena of anastomosing river and the Cellular Automata model which was proposed for simulating this phenomena. This model used the Cellular Automata paradigm in their classical meaning: regular grid of cells, homogeneous set of rules, simple set of states etc. The crucial algorithm and their implementations are briefly presented. Sample results obtained from the simulations are proposed and discussed. Finally, the main drawbacks of this model is enumerated and explained.

The Cellular Automata models usually require large grids due to the size of the modeled phenomena or their microscale nature. **Chapter 3** discusses methods for conducting high performance simulations with Cellular Automata. When the anastomosing river model was proposed, the leading architecture for conducting parallel simulations entailed various platforms with distributed memory and a message-passing library. The best cost to performance ratio was offered by the cluster systems. Thus, the model dedicated to this platform is presented and discussed. Since 2008, graphic processors (Graphic Processing Unit — GPU) began to be applied for implementing algorithms designed for purposes other than rendering.

Cellular Automata which have data structures similar to graphic data became one of the methodologies most often implemented on GPU platforms. The crucial part of the anastomosing river model — water distribution algorithms — was optimized according to the principles of streaming processing. The construction thereof and the results of performance test are presented and discussed. Researches related to these algorithms also include usage of a different type of memory provided by the GPU. The results of these investigations are presented and some practical conclusions are given. Finally, the Cellular Automata model of pedestrian dynamics implemented for a GPU is presented and also discussed briefly as another example of the effectiveness of using a GPU for implementing Cellular Automata models.

The property of multiscality is observed for many natural phenomena. The research on the anastomosing river system model was the inspiration for introducing a new methodology which can be treated as an extension to the traditional Cellular Automata approach. **Chapter 4** starts with a reminder of the drawbacks of the traditional Cellular Automata approach. Next, the general idea for combining graphs and Cellular Automata is outlined. This new methodology was called the Graph of Cellular Automata since the graph structure is used to connect (or rather define a neighborhood) cells. The formal definition of the model of anastomosing river employing this idea is presented with sample results.

In this monograph the Graph of Cellular Automata is developed into mature tool for modeling a general class of multiscale systems composed of transportation network and consuming (or producing) environment. Its application is presented using two biological phenomena as examples. The first model is related to the Tumor Induced Angiogenesis phenomena. It is the process of formation of blood vessels stimulated by the development of a solid tumor. This phenomena can also be classified as having two main components: a transportation network which circulates the blood carrying nourishing resources and the surrounding tissue which consumes these resources and produces metabolites. For this phenomena the Graph of Cellular Automata was extended by including a submodel of blood circulation. Additionally, a methodology based on graph descriptors and multidimensional analysis is introduced to verify simulations and results, and calibrate and fine-tune the model. The Graph of Cellular Automata naturally supports this methodology. The second model presented and discussed in this chapter is also related to living organisms but from a different kingdom: Fungi. The *Fusarium Graminearum* is responsible for great losses to cereal crops. The mycelium of this fungi invades and destroys wheat ears. In that case, the presence of transportation networks which receives nutrients from surrounding plant tissue is also observed. The specific property of the mycelium network which can be also expressed using the Graph of Cellular Automata is the ability to partition a larger network into several smaller parts that develop separately.

This example demonstrates that the Graph of Cellular Automata is able to model systems with a decentralized structure — there is no root in these networks.

Cellular Automata and Agent-based Systems share some properties. Some experts classify Cellular Automata as a special case of Multi-agent Systems. **Chapter 5** presents how Cellular Automata handle the environments for agents representing living marine creatures. This chapter presents a model of the ecology and evolution of single-cellular marine organisms called Foraminifera. Such a model is desired by micropaleontologists and microbiologists due to the fact that Foraminifera belong to a group of organisms that have a perfectly preserved fossil record covering 500 million years, and they still live in salt waters. The model provides an opportunity to verify some ecological and evolutionary hypotheses that appear during investigations related to this organisms. The most accurate method for modeling populations is Individual Based Modeling — the paradigm proposed in computational ecology 40 years ago. Now its popularity has grown significantly due to support from agent-based modeling platforms and availability of powerful computational systems. In the proposed model, Cellular Automata are used to model the environment (open ocean and seabed) with dynamic and complex processes, i.e., currents, insolation, temperatures, nutrition. Cells can be occupied by one or more agents that are influenced by local conditions, gather food stored in cells and interact with agents in the same cell. Additionally, at the implementation level, Cellular Automata organize the computations — the model is processed in the same manner as a typical Cellular Automata model. The implementations for a distributed platform and for a GPU platform are presented with some sample results.

The **final chapter** of the monograph is a summary. It is a brief review of all the investigation results. It also includes some conclusions. The final part of this monograph constitutes a bibliography.

* * *

Hereby I would like to thank all the people without whom this monograph might not have been written. First of all, I wish to express my gratitude to my mentors: Professor Jacek Kitowski and Professor Witold Dzwinel. My thanks are also extended to the Head of Department of Computers Science, Marek Kisiel-Dorohnicki and to Aleksander Byrski and Piotr Faliszewski who strongly supported me during the process of writing this text.

I would also like to thank my colleagues and collaborators: Jarek Tyszka (Polish Academy of Sciences), Maciek Komosiński (Poznań University of Technology), Jarek Wąs, Konrad Kułakowski and Kamil Piętak (AGH University of Science and

Technology). My thanks also go to my students: Andrzej Andrzejewski, Maciej Kuźniar, Paweł Młocek, Maciej Bassara and Adrian Klusek for their help in preparing experimental results presented in this monograph.

Finally, I would like to express my heartful gratitude to my wife Agnieszka and my sons for supporting me in every possible way right from the start.

Paweł Topa
Kraków, December 2017 — October 2018

1. Introduction

Complex systems are a 'hot' topic today. This comes from the fact that complexity is recognized in almost all natural and artificial phenomena. The notion of complex systems seems to be more or less clear to everyone, but the formal definition is not widely known. Additionally, this term might be understood ambiguously: complex also means made of many parts. Thus, the best way of defining this term is to enumerate the main properties of complex systems.

Everyone agrees that:

> A complex system is composed of **many** interacting components

These properties result in the fact that complex systems are difficult to simulate using a traditional approach; i.e., Partial Differential Equations. This respectable methodology describes the global behavior of phenomena and usually treats it as continuous in space and time. Any attempts to employ this approach to model phenomena that are naturally discrete meet the problem of linking a local solution to a global one.

Multiplication of components is nothing without interactions:

> Complex behavior emerges as a result of **interactions** between components.

In fact, the interactions create the complexity by establishing various positive and negative feedback that drives the components into new trajectories. As a result, the completely new behavior of a whole system might emerge during the simulation.

Unfortunately, the multiplicity of this feedback results in the fact that:

> A complex system is a phenomenon whose evolution cannot be deduced from an analysis of the behavior of its single components.

Thus, it is impossible to split the phenomena into separated components, model them separately, and finally come to a conclusion about the global behavior. When working together, the components create positive and negative feedback whose existence might not be suspected when the components are analyzed separately. The only way to discover it is to follow the phenomena via computer simulations, where components and interactions can be modeled as precisely as possible. Methodologies that are able to do this were invented many years ago: Cellular Automata of the late 40s of the 20th century, swarm intelligence of the 80s, and the agent-based modeling of the 70s and 80s. Today, they are also supported by computing power provided by modern computer architectures.

It is always possible to distinguish the main components that constitute a particular system as well as the basic relationships or interactions between them. It is usually enough to construct a simple phenomenological model that might be effective in imitating reality. However, in most cases it is not enough to accept such a model as a scientifically valuable tool. Models that are able to produce results useful for science and engineering usually have to include more precise rules (or submodels) that control the components and their interactions. In some cases, this requires us to modify the modeling methodology by introducing new assumptions that help to better represent the specific features of the modeled phenomena.

1.1. Modeling Complex Systems

When we mention a mathematical model of a system or process, it is usually expressed in the form of ordinary differential equations and partial differential equations. The differential equations are usable when the model has a small number of independent variables and their evolution proceeds in a continuous and smooth manner. As these equations are well-known, the influence of the parameters on the outcomes are clearly visible and understandable. However, their analytical solutions do not always exist; only a few differential equations have a closed-form solution. Thus, numerical methods are essential parts of equation-based models. The use of these methods brings several related challenges, like truncation, round-off errors, conditioning, and numerical stability. Notwithstanding, the numerical algorithms used in practice in science and engineering are generally resistant to these problems.

In the case of complex systems, the application of a traditional mathematical approach is problematic. Two main issues have to be concerned: the presence of nonlinearity, and the number of details that must be expressed in a model. The nonlinear differential equations that can be used in a model to express nonlinearity are often analytically unsolvable (which entails the use of numerical methods). However, it is much more difficult to compress a model of a particular phenomenon into the form

of a single general equation. In turn, when a model is composed of many entities, it is difficult to analytically deal with a very large set of variables and equations. An alternative solution is to use modeling methods that directly represent the simpler components of the system and the interactions between them, hoping that the complex behavior will emerge itself during the simulation.

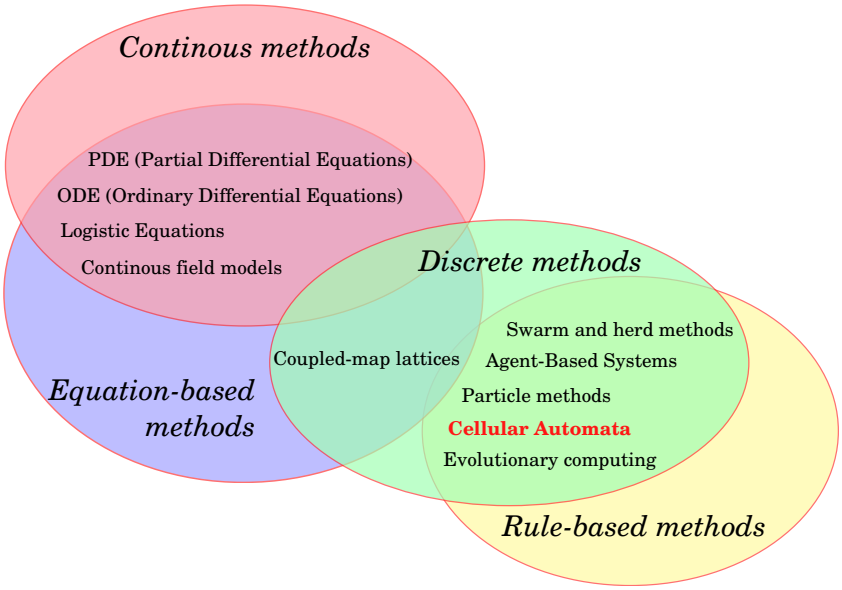

Figure 1.1. Brief taxonomy of main modeling and simulation methods used in research on Complex Systems

Despite the limitations mentioned above, the methods based on mathematical equations can be used to model complex systems. Figure 1.1 presents simplified taxonomy of main methodologies used to model and simulate Complex Systems — more comprehensive attempt to this topic can be found in [7]. These methods were divided according the two criteria: a discreteness of representation and a method of representing the system evolution. Continuous and equation-based methods usually use two or more basic equations coupled together like in reaction-diffusion systems [8]. To this group belongs also methods that use iterative maps or logistic equations [9]. The discrete and rule-based methods are more popular in this applications due to problems discussed above. In this group, the Cellular Automata most clearly demonstrate these features. There are also many other methods of that type since their general ideas are intuitive and often inspired by nature. Some methods, like Coupled-map Lattices [10], are hybrid and combine discrete representation with equation-based evolution.

1.2. Simulation as scientific method

Nowadays, computer simulations are considered an equal scientific method to the two standard methods of induction and deduction [11]. It works like deduction, starting an investigation from a set of assumptions, but it does not provide a theorem proof. Compared to typical induction, the data provided by simulations are produced by a set of rules instead of measurements of the real phenomenon. Therefore, the simulation modeling mainly support deductive reasoning by providing some kind of virtual laboratory (experiments *in computo*). The induction methodology can still be applied to the results of simulation in the same way as in the case of real observation. Summarizing: building a model and running a simulation can be seen as a method that help us to verify the accuracy and completeness of our knowledge (or intuition) on particular phenomena.

There is one more function of computer models that should be emphasized. Programming languages can be treated as kinds of formal languages interpreted by computers during program execution. They can be used as an unequivocal formalism that allows us to describe a model of phenomenon analogously to mathematical equations. In addition, this formalism allows us to easily verify the assumptions by directly executing the rules of a model. Finally, the results generated by the model can be investigated using induction methodology in the same way as empirical data.

The Universal Turing Machine is a model of any kind computation, and Epstein [12] pointed out that an implemented model can be translated into a set of rules for the Turing machine. In turn, for every Turing machine, a corresponding Partial Recursive Function exists [13, 14]. Thus, equivalent equations exist in theory for every computer model — even if they involve recursive functions. Therefore, we might consider that a computer simulation is equivalent to an appropriate equation-based model.

1.3. Cellular Automata

The Cellular Automata are one of the most widely known modeling and simulation paradigm. It was developed by John von Neumann and Stanisław Ulam in the late 40s of the 20th century [3] for simulating replicating systems. The worldwide career of Cellular Automata started in 1970 when Martin Gardener presented the "Game of Life" rule invented by mathematician John Horton Conway in Scientific American [4]. This extremely simple rule appeared to be the source of very complex behaviors and was later investigated in hundreds of publications.

There are many definitions of Cellular Automata, but all of them emphasize the discreteness of the representation of the components:

> Cellular automata are mathematical idealization of physical system in which **space and time are discrete**, and physical quantities take on a **finite set of discrete values**. (*Stephen Wolfram, Statistical mechanics of Cellular Automata, 1983* [15]).

More precisely, the classical Cellular Automata model consists of the following:

- a regular Cartesian grid of finite state automata in each node,
- the state of each automaton is described using a value from a discrete set of states,
- the states of the automata are synchronously updated using a defined set of rules,
- a new state of an automaton is calculated using a previous state of this automaton and its closest neighbors (4 — von Neumman neighborhood, 8 — More neighborhood).

The discreteness of space, time, and state variables is a very convenient feature from the point of view of computer implementation. These assumptions resulted in the fact that the approximations of continuity are not necessary and that all of the related issues are absent here. Also, the schema of neighborhood facilities computer implementation. Unlike meshless methodologies, Cellular Automata models do not need any methods nor optimizations for calculating a neighborhood.

Another feature of Cellular Automata that is convenient for implementors is the fact that dynamics are expressed in the form of rules. This fits naturally into the paradigm of computer programming, which surely relies on defining rules for manipulating data. The evolution of an automaton is also designed and directly encoded into the form of short programs.

The Cellular Automata paradigm is natively parallel. In fact, this feature has to be emulated in a basic sequential implementation when the automata are processed one by one. When the model is implemented for parallel computers with distributed memories, the neighborhood schema also provides benefits like static communication topology and a small amount of data that must be exchanged between computational nodes. Since GPUs (Graphics Processing Units) became the widely used platform for non-graphics processing (GPGPU — General Purpose computations on a GPU), Cellular Automata models gained a computing platform in which the processing organization fits perfectly to the Cellular Automata concept.

The Cellular Automata methodology provides a very clear and convenient method of building models of any kinds of systems of phenomena. The modeled system is represented by regularly distributed entities (automata) that only interact

synchronously with their closest neighborhoods. The modeler must partition the system into elementary fragments (areas, particles, etc.) and then define the interactions between them. Unlike in mathematical models (where a proper equation must be found), the interactions can be defined based on empirical observations or intuition. In fact, this is the only way for some phenomena (e.g., the movement of pedestrians).

On the other hand, when the dynamics of a system are expressed in the form of rules instead of mathematical equations, the model might be treated as a toy without any scientific value. For example, one of best-known Cellular Automata models is the model of the oscillating chemical Belousov-Zhabotinski reaction. In reality, this chemical phenomenon is very complex and is thought to involve 18 different steps (chemical reactions). In a model of this reaction, the component processes are radically simplified. Moreover, the model seems to have nothing in common with chemistry — it is named the Hodgepodge Machine [16]. In the world of this model, the cells can be healthy, infected, or ill. Once again, the cells are only influenced by their direct neighbors. Healthy cells that are surrounded by infected and ill cells become infected. Infection means that a cell "feels" worse and worse until it achieves the final state of "illness." Ill cells magically become healthy (but not immune). Notwithstanding, this toy-model appears to be a perfect metaphor of a complex chemical phenomenon.

When Cellular Automata is applied to model a particular physical phenomenon and the goal is to obtain results that are not only qualitatively but also quantitatively correct, the model-metaphor is usually not suitable. Without rules that try to copy the real processes that constitute a particular phenomenon, it is usually impossible to adjust the spatial and temporal scale of a model with reality. Additionally, the model need to be calibrated and validated against empirical observations (see the examples and discussions in many publications: [17, 18, 19]). In the case of some rules, it was possible to prove that at they are equivalent to the respecting continuous mathematical models; e.g., the Lattice Gas Cellular Automata model [20] is equivalent to the Navier-Stokes equations (fluid dynamics) at a macroscopic level.

Rules for Cellular Automata models can be formulated from the continuous models of a particular phenomenon and is usually expressed in the form of partial of ordinary differential equations. This approach based on the similarity between a finite difference scheme and finite-state Cellular Automata. The finite difference scheme is a method of discretizing a differential equation on a grid of points. In discrete time steps, the values of the equation at each point are calculated. The Cellular Automata rules derived from basic differential equations (heat equation, reaction-diffusion equation, wave equation, etc.) have been published and analyzed in many papers [21, 22, 23]. As the models behind these rules are elementary, they are mostly used as a component in more-complex models.

From the point of view of the implementation and running of a simulation, this paradigm brings great benefits. Cellular Automata are inherently parallel and use only local interactions with a limited range (usually the closest neighborhood). The models that use CA can be implemented and run with high efficiency on most parallel platforms like clusters and GPU processors. The latter architecture especially provides very interesting results. GPUs are primarily designed to render graphical data: vertices that constitute scenery and pixels that fill a screen buffer. This data has a quite similar structure to the regular lattice of Cellular Automata. Thus, only a little effort is necessary to catch the idea of Cellular Automata programming for GPUs. In Chapter 3, these issues are discussed with more details and with examples.

1.4. Cellular Automata: extensible modeling and simulation tool

Cellular Automata rules have been intensively explored and investigated for years as general models of dynamic systems manifesting complex behaviors. However, much effort is also devoted to applying this paradigm to modeling specific phenomena or process. Every two years during the ACRI conference (International Conference Cellular Automata for Research and Industry)[1], new ideas for using Cellular Automata are presented. The Journal of Cellular Automata[2] also presents a few examples of new applications of this paradigm in each issue. It is obvious that the assumptions of classic Cellular Automata are too limiting for representing the complex structure and evolution of most natural phenomena. Modifications and extensions of the classical definition are necessary; in a later part of this section, several examples of Cellular Automata models with extensions are briefly presented.

Cellular Automata with non-discrete states

The original definition of Cellular Automata assumes that the state of each automaton is described using a discrete set of values. However, this assumption significantly limits the application of the Cellular Automata when the models of natural phenomena are considered. The components of such models can have more states than merely “alive” and “dead,” or their states can be described by using continuous values. Floating point numbers allow us to more precisely and efficiently describe the state of such an automaton. A clear example of the benefits from this extension is a group of models that were developed to simulate phenomena related to volcanic eruptions and landslide processes [24, 25, 26, 27]. In these models, the grid of the

[1] ACRI 2018: http://acri2018.disco.unimib.it/

[2] http://www.oldcitypublishing.com/journals/jca-home/

automata represents an area with lava flows or mud. In the basic model, each cell of the Cellular Automata represents a portion of this area, which is characterized by the elevation over a specific reference level and by the thickness of the lava or mud layer. In more advanced models, additional parameters related to the physical properties of lava or mud were introduced in order to model the process more realistically.

The Cellular Automata with state described using continues values can be compared to Coupled Map Lattices [10, 28]. Both of these methodologies use discrete representation of modeled systems. However, the Couple Map Lattices have very precise definition of how the next state of each site is calculated — they used logistic maps, i.e., $x_{n+1} = rx_n(1 + x_n)$. Logistic map can include into the computation neighboring sites in the similar way as in the case of the Cellular Automata. This approach can be treat as an variant of the Cellular Automata for which the definition of how the state is updated is not given so precisely. In fact, the Coupled Map Lattices in scientific literature are clearly distinguished from the Cellular Automata but they applications for modeling natural phenomena are not widespread.

Another extension related to describing the state of a cell is using fuzzy logic [29]. The main purpose of using this tool is the introduction of uncertainty into the model. There are several examples of using Fuzzy Cellular Automata for modeling traffic flow, where the randomness of driver behavior can be critical for the dynamics of these phenomena [30, 31, 32]. Other areas in which this framework has been successfully applied are for modeling urban processes [33, 34] and image processing [35, 36, 37].

Cellular Automata with virtual particles

Lattice gas automata (or lattice gas cellular automata – LGA) are another group of models that have roots in the original methodology of Cellular Automata. These methods was introduced for modeling complex fluid flows for which the traditional approach based on solving the Navier-Stokes equation might fail. Their authors [38] proposed the use of a Cellular Automata grid as a space for virtual particles that jump among the automata. Each automaton can be in one of two states: “empty” and “occupied by a particle.” In order to properly model the movements of particles, the updating consists of two stages: propagation and collision. During the collision stage, the conflicts between the particles that move into the same automaton are resolved by using defined rules. The first model (called HPP – from the names of the authors) was highly anisotropic due to the lack of rotational invariance. This drawback was eliminated in the second FHP model (by Frish, Hasslacher, and Pomeau [20]) by introducing a hexagonal mesh and new rules for collisions.

The next step in the development of fluid models based on the Cellular Automata paradigm was the Lattice Boltzmann method (LBM) [39, 40]. In this model,

the binary representation of particles is replaced by a particle distribution function (a function that represents the probability of finding particles at a given position at a given time with a specified velocity) based on the Lattice Boltzmann equation [41]. Additionally, the simple set of collision rules is replaced by a collision operator. The Lattice Boltzmann method was designed to be a remedy for the statistical noise that influences the macroscopic quantities in LGA methods. Additionally, it eliminated other problems of LGA methods, like the lack of Galilean invariance or velocity-dependent pressure [42]. Moreover, it has been proven that Navier-Stokes equations can be derived from the Lattice Boltzmann method [43].

All of the lattice gas methods inherit their potentially high performance from the Cellular Automata approach in simulations. This is mainly achieved by a simple schema of parallelization. Thus, the Lattice Boltzmann method is one of the most important methods in the area of Computational Fluid Dynamics (CFD). There are hundreds of examples of its application in various scientific areas where the flow through complex geometries is investigated (like blood arteries [44, 45] or cracked rock [46, 47]) and when the flows include mixtures of fluids with various parameters [48, 49]. Also, the Lattice Boltzmann method is popular in computer graphics as an efficient method for generating realistic-looking flows [50, 51].

The idea of a virtual particle traveling across a Cellular Automata grid is used in many other models; i.e., models of granular flow[3]. This type of phenomenon includes the movement of a large number of individual objects like grains, particles, pedestrians, cars, etc. The dynamics of this flow emerge as a result of the interaction between moving objects; hence, this type of process can be naturally modeled by using the Cellular Automata paradigm. Typical granular flow involves the movement of particles made of solid materials. The models proposed for simulating these systems assume the existence of virtual particles that move between nodes of a grid and bounce off each other. Thus, these models are generally defined analogically to Lattice Gas Automata models, but they use different types of potentials of interactions between particles (see [52, 53, 54]).

Traffic and pedestrian flows can also be considered a granular flow. Cars and pedestrians move towards POIs (Points of Interests) and interact in order to avoid direct collisions. These models have huge practical applications. Models of pedestrian dynamics are used for testing evacuation scenarios [55] and designing escape routes in buildings and public facilities; e.g., [56, 57, 58, 59]. In this area of investigation, the Cellular Automata paradigm is a very attractive approach due to several benefits; e.g., the easy representation of complex interiors of buildings and opportunity to model huge gatherings with relatively high efficiency [60, 61]. However, agent-based modeling also seems to be a very useful methodology; the Cellular Automata

[3]Conference on Traffic and Granular Flow, Biannual Conference, 1995–2017

paradigm is able to provide greater efficiency here without losing accuracy. During an evacuation, people usually have a simple strategy — to move towards exits — so considering the complex behavior of agents is not necessary. Nevertheless, agent-based models and models mixing Cellular Automata with agent methodology are often proposed [62, 63].

The Cellular Automata models of pedestrian dynamics need to be equipped with a mechanism that mimics the knowledge of pedestrians on area topology – especially the location of POIs. In most cases, this is realized using floor (or potential) fields [58]; this component is discussed in a more detailed way in Section 3.3 of this monograph.

One of the first models of traffic flow was the one-dimensional Cellular Automata rule proposed by Nagel and Shreckenberg [64]. In fact, if we turn off some features in this model, it is identical to rule 184 by Stephen Wolfram. There are hundreds of models of these phenomena considering the various configurations of roads, lanes, and crossroads [65, 66, 67, 68]. The models were used to investigate how various factors (traffic lights, hindrances, parking places, collisions, and driver behavior) affect traffic flow and traffic jams [69, 70]. As in the case of the models of pedestrian dynamics, the models of traffic flow are used not only to investigate these phenomena but also to support the design and management of real traffic flow (see [71, 72]). Even the simple Nagel-Shreckenberg model was employed to forecast traffic flow in the German state of North Rhine-Westphalia [73].

Cellular Automata with irregular grid

A distinctive feature of all Cellular Automata models of traffic flow is a grid that usually is not in the form of a regular Cartesian mesh. One-dimensional Cellular Automata is a natural method to represent a one-lane road. Two-lane roads can be represented as two one-dimensional grids with the appropriate rules that govern the process of changing lanes. In order to model the crossroads, a network of linking the stripes of the one-dimensional Cellular Automata is created. In this case, it is necessary to have rules that apply to the cells that are located in the junctions. Using this method, more-complex crossroads (i.e., roundabouts/traffic circles, throughabouts, multilane crossroads) can also be represented, preserving the high precision of the simulation [74, 75].

The regular Cartesian lattice is one of the most fundamental features of Cellular Automata. However, like in the case of other features, alternative topologies of the lattices have been tested in several models. The regular hexagonal mesh proved its advantages in lattice gas automata. Irregular lattices are considered in the case of phenomena that have natural irregularity; its approximation by a regular lattice might influence the model accuracy and efficiency.

Investigations on urban processes is an area in which Cellular Automata models with irregular lattices are often used [76, 77, 78]. Cities usually have very irregular structures due to the long-term incremental development. Streets, plots of land, buildings, and the other municipal infrastructure form a complicated network in which the relationships of neighborhood might be not obvious. A reliable source of data in this case are various GIS (Geographic Information Systems) that combine various data into complete information about the locations, shapes, boundaries, and mutual interactions of objects. Another source of such data is a cadastre (which mainly provides information about ownership). Such data might be useful in models developed for forecasting changes in land values [79].

Cellular Automata with a regular lattice is used for modeling urban processes; e.g., [80, 81]. However, this representation might be too fine-grained in some applications. The irregularity might be partially addressed by fixing some cells into a predefined state [82] — this method is analogous to those methods used by models of pedestrian dynamics.

In general, irregular lattices provide a more accurate and efficient representation in Cellular Automata urban models. This type of lattice can be simply generated by using Voronoi polygons [83]. Also, graph theory might be directly employed to represent a lattice [84]. In such an approach, the definition of Cellular Automata is extended by treating the lattice as a form of a graph with a very specific topology (planar graph). This approach has several advantages that mainly result from the application of the formality and tools of graph theory.

Applications of Cellular Automata with irregular lattices to phenomena other than urban dynamics are less frequent, but they are also related to the case of phenomena with natural irregularity. In [85], the authors proposed a model for managing a network of wireless sensors [85]. In their model, the lattice is formed from randomly and spatially scattered sensors. An irregular and random lattice is also applied to model recrystallization and grain growth [86]. The motivation for using this method is the heterogeneous nature of this process, where local variations have a substantial influence on the patterns on a greater scale.

An irregular lattice represented in the form of a graph was also applied to a model network of vascular vessels in the process of Tumor Induced Angiogenesis [87, 88, 89]. These models were developed by the author and are explained with all details in Chapter 4.

When an irregular lattice is used, the Cellular Automata paradigm loses one of its greatest advantages. A regular lattice only requires the simplest data structure (like arrays). It radically simplifies the algorithms and positively influences their efficiency. The regularity of a lattice also facilitates the handling of the neighborhood schema. Irregular lattices definitely require more effort. Some investigations suggest

that lattice topology does not influence the crucial aspects of the dynamics of Cellular Automata models much [90, 91]. Another conclusion that appeared from these investigations was the fact that irregular lattices open up the possibilities for new interesting hypotheses on Cellular Automata dynamics.

Cellular Automata with asynchronous updating

Synchronous updating is very convenient from the point of view of designing and implementing models — only one simple updating schema is used. However, there are some reasons to question this assumption [92]. First of all, the external clock that is necessary to implement synchronous updating is an artificial assumption. In nature, there are no mechanisms that synchronize the processes between the components of the system [93, 94]. The issue of an external clock is also important from the point of view of parallel computing. The necessity of synchronization brings additional overhead, which impacts the performance of simulations. Asynchrony in Cellular Automata raises questions about sensitivity or the resistance of its dynamics to various method of updating.

The dynamics of asynchronously updated Cellular Automata have been investigated in many publications. Mostly, they show that asynchrony changes the dynamics produced by the Cellular Automata rules as compared to synchronous updating. According to [95], the asynchronous updating used in the "Game of Life" resulted in removing its well-known chaotic behavior — for any initial configuration, the convergence into a fixed state was observed. However, subsequent works [96] undermined these conclusions and suggested that this "stabilizing effect" is not common but rather depends on the parameters of the updating schema (e.g., the frequency of applying rules to individual cells). The qualitative transition from "convergent" to "active" behavior was observed when the probability of updating was changed [97]. This phase transition appeared to belong to the percolation universality class, which means that changes in the order parameters (e.g., density) obey the power law at the critical point. The connection between the frequency of updating and Cellular Automata behavior was also noticed in [98], where the spectral analysis was aimed at investigating whether the asynchronous updating disturbs the $\frac{1}{f}$ noise existing in the synchronous Cellular Automata [99]. The examinations showed that, for a higher probability of updating the $\frac{1}{f^2}$, noise is rather observed. However, some investigations show that some rules are less sensitive to asynchronous updating. The elementary Cellular Automata in [100] were classified into three categories: invariant for asynchronous updating, robust to perturbation of asynchronous updating, and prone to perturbation of asynchronous updating.

Despite the problems and ambiguities mentioned above, attempts were taken to define the asynchronous Cellular Automata that have the same fundamental

properties as classic Cellular Automata. The rules that are able to simulate a universal Turing machine are presented in [98]. Also, the capability of self-reproduction was achieved with the set of rules proposed in [101].

Due to the fact that a direct comparison of the dynamics of synchronous and asynchronous Cellular Automata gives unclear and ambiguous results, many researchers were focused on developing a new methodology for the analysis of the influence of asynchrony on the dynamics of Cellular Automata. In [102], asynchronous Cellular Automata were studied using the Lyapunov exponent, which is the classical indicator of stability in dynamical systems. This was applied to evaluate the stability of the Cellular Automata model with five different updating schemas. Agapie et al. [103] used the Markov chain theory to analyze the properties of asynchronous Cellular Automata. They assumed that asynchronously updated totalistic one-dimensional Cellular Automata induces the ergodic time-homogenous Markov chain. They mainly tested their methodology for a few variants of a neighborhood. Another method that was based on the observation of Cellular Automata behavior was proposed in [104]. The authors measured how asynchronous updating influenced the density of the patterns generated by various elementary automata. The rules were classified according to their robustness to asynchronism. Additionally, an interesting conclusion from this analysis was the fact that there is no coincidence between this classification and the Wolfram classification. The influence of asynchrony on the dynamics of Cellular Automata was also investigated by using the formal properties of Cellular Automata rules, like transitivity, sensitivity, and equicontinuity [105, 106].

Asynchronous updating means that the cells are not processed in parallel (or quasi-parallel) but rather in a defined order [107, 108]. This method requires the specification of the order in which the cells are processed [109]. Asynchronous methods are divided into two groups: 'time-driven,' in which the updating of a cell is triggered by its internal clock (time variable), and 'step-driven,' which requires a queuing algorithm. Such an algorithm can implement various strategies [109]:

- predefined, fixed order,
- a randomly generated sequence for the whole simulation,
- a new random sequence at each step,
- a completely random selection of cells for updating.

The 'time-driven' method allows us to simulate continuous time in discrete models [109]. Normally, the time is quantified in Cellular Automata, and the events only occur in equal time periods. Internal clocks allow us to activate cells at specific times, which is usually governed by a defined distribution (i.e., exponential distribution).

As mentioned above, some researchers argue that asynchronous updating better corresponds to the nature of natural phenomena. However, this method is not very popular in practice and is mainly applied to model systems in which the asynchrony

of events or activities is explicitly given. An example of such a system is a network of sensors that are randomly triggered by external factors [110, 111, 112]. The main purpose of these models is to find a strategy for the efficient use of these devices, which includes power management, optimization of radiation level, etc. Another interesting example is the model of illumination in public areas presented in [113, 114].

It is worth emphasizing that, in many models of urban processes, irregularity goes together with asynchronous updating [115, 116]. Once again this method is introduced because the elementary parts of the modeled phenomena evolve according to their individual clocks; e.g., their construction proceeds at different rates.

Cellular Automata with extended neighborhood

The range and shape of a neighborhood area is another component of Cellular Automata that was modified with respect to the original concept. One-dimensional Cellular Automata use a simple neighborhood that includes two cells: before and after a particular cell. Traditionally in two-dimensional Cellular Automata, a particular cell changes its state using the four closest neighbors that lie north, south, east, and west of the central cell. This most simple schema is called the von Neumann neighborhood (see Fig. 1.2). The Moore neighborhood includes an additional four cells that lie in diagonal directions. These schemes are the most popular and most often used. They simplify the implementation of algorithms and provide high efficiency for the models due to their simple and regular access to memory.

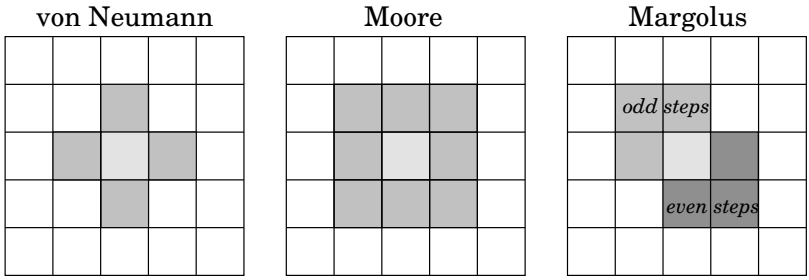

Figure 1.2. Three Cellular Automata neighborhood schemes: von Neumann neighborhood, Moore neighborhood and Margolus neighborhood

Apart from the two canonical schemes mentioned above, there are also several other schemes that were invented for some particular models. One of the best-known is the Margolus neighborhood [117] (see Fig. 1.2). This schema assumes that the lattice is divided into four-cell blocks (a size of 2×2 cells) that are shifted by one cell along each dimension on alternate timesteps. Margolus proposed a specific set of local transformations called "block rules" that are applied to the whole block. Unlike

the von Neummann and Moore neighborhoods, all of the cells inside the block are affected. These types of Cellular Automata have the property of reversibility [118]. The "block rules" are a one-to-one transformation; thus, it is possible to uniquely retrieve the previous configuration from the current one. Cellular Automata with this type of neighborhood are also called Partitioning Cellular Automata [119], since the lattice is partitioned into a collection of disjoint pieces called blocks. An additional feature that resulted from the use of this neighborhood is the lack of "long distance" interactions that appear in the form of undesirable patterns like waves or oscillations. [120]. The Margolus neighborhood was initially applied in the Cellular Automata version of Friedkin's Billiard Ball Model (BBM) and later in the sand pile model [121]. Additionally, in [122], Toffoli and Margolus concluded that the Margolus neighborhood is equivalent to the scheme of updating used by the HPP lattice gas model. In one-dimensional Cellular Automata, the same properties are obtained using the brick neighborhood (see Fig. 1.3). Also, Morite at al. [123] proposed and investigated Partitioning Cellular Automata for a hexagonal mesh.

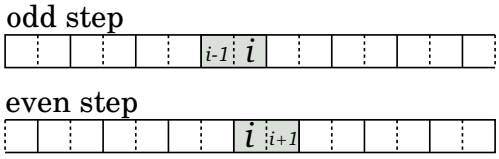

Figure 1.3. Brick wall neighborhood

Cellular Automata with triangular and hexagonal lattices also have a few neighboring schema that differentiate in shape and the number of neighbors. Figure 1.4 demonstrates three examples: honeycomb, tripod, and hexagonal stars. The last schema is characterized by including cells that lie at distances greater than 1.

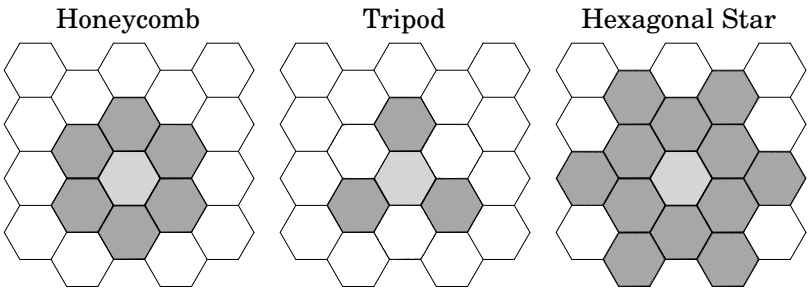

Figure 1.4. Three neighborhood schemes constructed on hexagonal mesh

A hexagonal mesh in Cellular Automata is best-known from the FHP lattice gas automata [20]; however, the neighborhood schema is basic in this case (only adjacent

neighbors). A good example of using a hexagonal lattice with a larger neighborhood is demonstrated in models that simulate the spreading of fire in forests [124, 125]. In these models, larger and asymmetric neighborhoods are used to include the effects of wind, topography, and the specific behavior of fire.

In fact, any new neighboring scheme can be defined by using any tessellation method; i.e., Voronoi tessellation (see [126]). A neighborhood can also be defined by using methods other than adjacency in a square or hexagonal mesh. Another method of defining neighborhoods is using graphs or networks; i.e., [127, 128]. Such a method is used in a methodology called Graph of Cellular Automata [129] (which is described in Chapter 4 in a more detailed manner).

The use of a neighborhood that includes more-distant cells in practice means that more information is used to calculate a new state. Only seemingly does this have a positive influence on the results generated by the models. In [130], the influence of the Moore neighborhood with a range of 2 was investigated, and the obtained results lead to the conclusion that "more does not mean better." The predator-prey model used in this case produced unstable and chaotic behavior that was far from what was expected. However, long-distance interactions are necessary in many applications; i.e., models of urban processes, buildings, and parcels might directly influence more-distant objects. In this area, the influence of the neighborhood was carefully examined [131, 132]. Different sizes of neighborhoods were compared in a series of simulations using the same domain. The results strongly supported the conclusion of the sensitivity of the models on a chosen neighborhood.

The belonging of a cell to the neighborhood of a particular cell in most cases is a 0-1 relation. However, an approach in which the influence of neighboring cells on the central cell can be tuned according to distance, position, or cell state can be reasonable. In an urban land-use model [133], the weighting function is used to classify cells in a really large neighborhood. The model definition assume that all cells within a radius of 6 cells (113 cells around the central one) are included into the calculations. Another example is the model of pedestrian dynamics based on proxemic theory [134]. The authors assume that the new state of a cell occupied by a pedestrian is calculated after a simplified analysis of the closest neighborhood. Cells with a pedestrian in a specific position might have a greater contribution than another. An alternative solution for a weighted neighborhood is a dynamic neighborhood. This method assumes that the schema of a neighborhood can be modified to better represent the mutual influence of various objects in a model. In the model of land-use changes [135], such a solution is proposed as a remedy for the excessive sensitivity of other models to a chosen neighborhood scheme.

In models of traffic flows, the neighborhood can have an unusual form for obvious reasons. Already, the first model of cars moving in one direction on a one-lane

street have a modified asymmetric neighborhood [64]. When the next action of a car is calculated, only the cell in front of the car is taken into consideration. In the case of more-complex topologies like two-lane streets [65, 136], bidirectional streets [68], and crossroads [137, 138, 139], the neighborhoods can have very diverse forms that reflect what a driver is able to see.

Any modified and extended neighborhoods might influence the complexity of the algorithms and the efficiency of the model. In the case of parallel implementations, this means that the communication between computational nodes can be more complex and more data must be exchanged.

Non-homogenous Cellular Automata

Homogeneity (or uniformity) is one of the fundamental properties of the original Cellular Automata. This means that all cells are identical and the same transition rule (or set of rules) is applied. Thus, heterogenous (also called non-uniform or inhomogeneous) Cellular Automata are characterized by applying different transition rules to different cells [140]. Additionally, the rules might change over time according to a defined strategy or rule.

The properties of heterogeneous Cellular Automata were studied using basic models in which the transition rules were selected from a defined set at random or by using other strategies. The analysis showed that some properties are lost (like injectivity, expansivity, and surjectivity) [141, 142, 143, 144], while others are preserved; i.e., it has been proven that these types of Cellular Automata are still able to do universal computations (what is more, with a smaller set of states) [145].

Heterogenous Cellular Automata were designed to model systems in which heterogeneity is crucial for their evolution. In models of traffic flow, they were used to simulate a stream of bicycles, motorcycles, and scooters through a crossroad [146]. The heterogeneous Cellular Automata facilitate to catch the dispersion phenomenon that is observed when such a stream is really massive (like on the streets of a major Asian city). In [147, 148], a stream composed of cars and motorcycles moving together is modeled. In this case, the heterogeneity is reasonable since these vehicles behave in a different way on streets. Another example of using this methodology is the model of a multi-lane roundabout [149] used by vehicles of various types. The heterogeneity was also used to represent different conditions of driving on the different lanes of the roundabout.

Modeling frameworks supporting heterogeneity, like agent-based systems, are very useful in models of population dynamics. In case of traditional Cellular Automata approach this property can be achieved by choosing an appropriate set of states that is able to represent the various features and abilities of the members of the population [150]. However, heterogeneous Cellular Automata more clearly address

this issue and might facilitate additional functionality. In [151, 152], each cell carries its own transition function, which is treated as a genome and can be changed using genetic operators. Heterogeneity in transition rules can be used to represent the influence of external conditions like in the model of the treatment of HIV infection. It assumes that different rules are applied to unaffected cells and cells affected by therapy, which obviously influences the spreading of this virus.

One use of heterogeneous Cellular Automata that is a bit exotic is presented in [153], where it is used to compose music. In this work, Cellular Automata is used to generate patterns treated as piano rolls. For each pitch, a different set of rules is used. The rules are learned (generated) from existing musical works using neural networks.

Most definitions of Cellular Automata use the term of homogeneous to emphasize that a system is composed of identical cells that undergo the same set of rules. There is a little mismatch by using the term heterogeneous. Some definitions assume that this is related only to using different rules or sets of rules. In fact, many models are heterogeneous in the sense that the cells have additional parameters that control their behavior when the transition rule is applied.

Cellular Automata and Agent-Based Systems

Cellular Automata has been applied several times for modeling population dynamics. In fact, the Game of Life might be included in this group. Cellular Automata has also been proposed many times for the classic predator-pray system [154, 155, 156]. It is worth noticing that this paradigm reveals some features that are common in biological systems: the locality of interactions and parallel evolution of entities. Also, the interactions between individuals constituting a population can be naturally expressed in the form of relatively simple rules. The mobility of populations (which is an inherent feature of many biological systems) can also be included in a CA model using various methods similar to those used in pedestrian modeling, for example.

Some of the extensions and modifications mentioned above result in the fact that Cellular Automata partially resembles the Agent-Based Modeling approach [157, 158]. In fact, some researchers treat Cellular Automata models as a specific case of a broader group of agent-based models. A clearly visible difference between Cellular Automata and agent-based systems is the lack of a grid; however, it in fact has a secondary importance. Agents can also be located on a grid, and their movement and interaction can be governed by this grid. The capital difference between the Cellular Automata approach and agent-based systems are the issues of independence and goal. According to most definitions of agents, their fundamental property is the ability to act independently of other agents [159, 160]. The cells in Cellular Automata do not have this property. Their state strictly depends on the state of their neighbors. Also, the agents have goals or tasks that they strive to reach or accomplish.

Cellular Automata cells simply execute the algorithm encoded in the form of a set of rules. However, as it was illustrated above, the Cellular Automata approach can be modified and extended to provide a better method for modeling some phenomena. It is also possible to introduce the independence of acting and goals to a Cellular Automata framework.

There are some combinations of Cellular Automata and agent-based modeling. The main philosophy that lies behind such a combination is to exploit the simplicity and efficiency of Cellular Automata where it is possible and apply agents where their flexibility is necessary. In models of a population interacting with its surrounding environment, such a combination provides a very convenient and clear framework [161]. An environment with all related parameters and processes is modeled using Cellular Automata, while the individuals are represented as agents. In [162], a model of bacteria colonization that is constructed by using this schema is presented. The bacteria individuals are modeled as agents, and the environment (the colonization surface, nutrients, and pH level) is represented by the Cellular Automata. The same schema has been applied to model the mobility of city residents [163] or other urban-related processes [164]. Another example of such a combination will be presented and discussed with all details in Section 5.2.

Cellular Automata in scientific investigations

Cellular Automata is one of the most popular modeling and simulation framework used in a variety of scientific disciplines. Table 1.1 presents the results of searches for the phrase "cellular automata" in three main services that index scientific publications. Google Scholar provides the highest values since it includes all publications found on the Internet (including those that have a low scientific value). Scopus and Web of Science are more valuable sources of science metrics because the publications registered in their databases are selected according to specific criteria.

Table 1.1. Number of scientific papers that have phrase "cellular automata" in title after three main bibliographic databases (author's own analysis, September 2018)

Database	All	2017	2018
Google Scholar	159,000	8340	4490
Scopus	20,073	1182	724
Web of Science Core collection	6370	264	165

Each year, more than 1000 scientific papers related to Cellular Automata are published. Only a small number of them present investigations on the properties of models based on the traditional definition. Most of these works present new applications

of this paradigm, usually with various modifications (as was briefly illustrated above). Thus, Cellular Automata is, in fact, a term referring to the family of models that share some common features.

1.5. Cellular Automata as flexible and efficient modeling framework

The main purpose of this book is to demonstrate the universality of the Cellular Automata framework. Although the original definition of Cellular Automata is very strict, even a cursory survey of publications with "Cellular Automata" in the title shows that this definition is constantly being extended. In fact, the only assumptions that are common for most Cellular Automata-based models are as follows:

- time and space is discrete,
- the behavior of each component is described using the rules of local interactions.

These two assumptions are crucial from the point of view of computer implementations. The most natural method for writing rules for Cellular Automata is the use of any imperative programming language. Also, the inherent discreteness removes the challenges related to the approximation of continuity with discrete values.

Cellular Automata can be considered a kind of modeling framework that is natural for computers and computing environments. In the previous section, it was demonstrated that Cellular Automata can be applied to a broad range of phenomena and processes. In a further part of this book, a few Cellular Automata models developed by the author are presented and discussed with more details, especially emphasizing the extensions introduced to the modeling framework and the benefits obtained.

2. Cellular Automata for modeling natural phenomena

It was shown in the previous chapter that Cellular Automata is widely used to simulate various natural phenomena. In this chapter, one example of such a model is presented and discussed. The model of an anastomosing river was a starting point for further investigation, which led to the formulation of a new extension to the Cellular Automata method and the introduction of new optimized parallel algorithms. This model also illustrates how the Cellular Automata paradigm is applied to model typical natural phenomena that are spatially extended. The model as well as the results of the simulations were published in a series of papers published from 2002 through 2004: [165, 165, 166].

In order to model surface water flow with Cellular Automata, the simulation domain must be decomposed into connected containers. The flow can be simply represented by moving some amount of water between these containers. Such a procedure is implemented by the rule of the minimization of differences [25]. In 2001 [167], D'Ambrosio applied this rule to simulate soil erosion by running water. This model was aimed to precisely reflect the processes of erosion in sloped terrains. It was able to create the characteristic pattern of a drainage network, and the amounts of eroded materials predicted by the model roughly corresponded to the observed values. In fact, this model was focused on only the local processes of transport between adjacent cells, and the final pattern was influenced by the landform given as an initial configuration. Another application of this rule also corresponds to the situation where the global evolution of a system is determined by the landform (like in the model of catastrophic floods and runoffs [168, 169]). The rule of the minimization of differences is unable to represent the momentum of flowing water; therefore, it represents slow flows better. In order to include the momentum of water, the rules must include the speed and direction of the flowing water. Such a model focused on the formation of a braided pattern was proposed in [170]. It uses Cellular Automata to implement a simple routing scheme for streams whose hydraulic model was governed by the Navier-Stokes equations for shallow water. A similar approach was applied to model

meandering rivers where the crucial governing factor is the erosion of riverbanks by river bends [171].

2.1. Anastomosing river systems

The factors that govern the evolution and anastomosis of river system were investigated by Prof. Gradziński et al. [172, 173] on the Narew River in eastern Poland. Gradziński and his team described the crucial factors that drive the evolution of this system. The models presented below were developed to acknowledge their theories through computational experiments. The term of an "anastomosing river" refers to a river system that possesses a highly complex network of forking and joining channels (see Fig. 2.1).

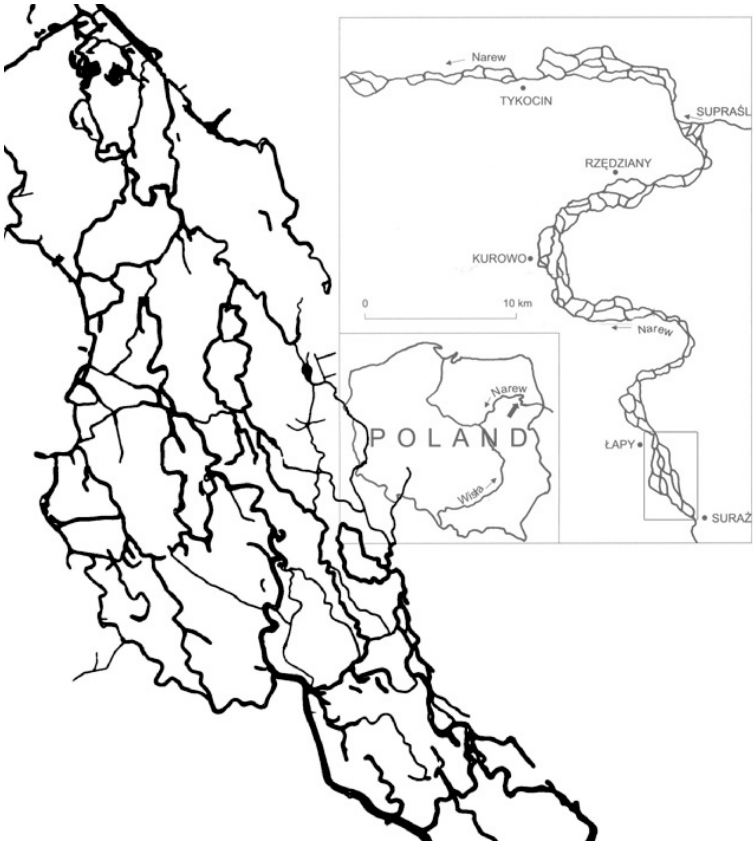

Figure 2.1. Part of Narew River in eastern Poland with clearly visible anastomosing pattern (map courtesy of dr Mariusz Paszkowski form Polish Academy of Sciences)

Anastomosing rivers represent the fourth main type of rivers (after straight, meandering, and braided [174]). Anastomosing rivers develop on flat and wide areas with a very small slope in the bottom valley (about 5 cm per 1 km). There are many other examples of such a system, including fragments of the upper Columbia River (in southeastern British Columbia, Canada), the Ob (Siberia), the Okawango (Africa), and more.

Using the Narew River as an example, Gradzinski et al. [172, 173] proved that the creation and existence of anastomosing systems is a result of plant vegetation (see Fig. 2.2). The necessary organic resources are carried by the river and penetrate the surrounding soil. The products of plant decay are accumulated in the interchannel areas as peat bogs. The accretion of a peat layer (vertical accumulation) is accompanied by sedimentation on the bottoms of the channels. The rate of plant growth is controlled by the nutrient supply level; other factors such as sunlight can be considered as spatially invariant. The gradient of the nutrient saturation that appears perpendicular to the channel axis results in the faster accumulation of peat near the banks and the slower accumulation on distant areas. The accretion of bottom sediment and peat along the channel banks results in rises in the water levels inside the channels.

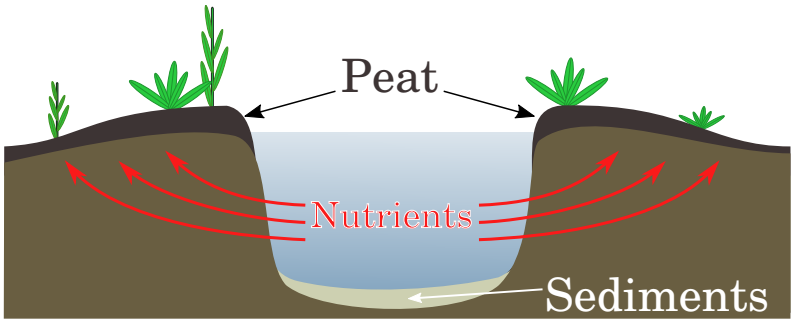

Figure 2.2. Cross-section of anastomosing river channel. Nutrients are transported from water to surrounding area, where they are available for plants through root systems [165]

Vegetation inside the riverbeds causes jams that lead to avulsion (the sudden displacement of all or part of an entire river channel [175]. Avulsions are primarily driven by aggradation[1] of the channel belt and/or the loss of channel capacity and throughput by in-channel deposition. Both processes are triggered by a low floodplain gradient. A new branch is initiated above the jam zone and may join the main channel somewhere downstream. Its route is mainly determined by the topography; however, on almost flat areas, new channels usually form a winding and chaotic pattern. The

[1] Deposition of solid material by a river, stream, or current

evolution of newly created channels proceeds in a similar fashion to the evolution of the main stream. Finally, a very complex system of hierarchically branching, joining, and interconnecting channels is created.

2.2. Cellular Automata model of anastomosing river

Cellular Automata with states represented as continuous values are used to formulate a fine-grained model of an anastomosing river. The simulation domain is composed of square pieces of land (they may be covered by water or not). The size of these pieces should be chosen to provide a sensible level of accuracy in representing the smallest structures. An anastomosing river can have very narrow channels (about one meter in width) — this can be very small when compared to the size of the valley (which may be several kilometers wide). Thus, it is necessary to find a compromise between the accuracy and cost of the computations. The three main processes contributing to the formation of the anastomosing river are represented by the rules of local interactions. The flow of water through the terrain is modeled as the pouring of water between neighboring plots. Peat bog formation is simply presented as an accumulation of peat in pieces with rates dependent on the nutrient level. The nutrient transfer from the riverbed to the neighboring soil is modeled using the well-known diffusion rules.

CA-based model of fluid distribution over landscape/terrain

The Cellular Automata model of anastomosing was inspired the model of lava flow proposed by Di Gregorio et al. [176]). More formally, the Cellular Automata model for simulating water flow in a terrain can be presented as follows:

$$CA_{FLOW} = \langle Z^2, A_i, A_o, X, S, \delta \rangle \tag{2.1}$$

where:

- Z^2 is the set of cells located on regular mesh and indexed by (i, j);
- $A_i \subset Z^2$ is the set of inflows;
- $A_o \subset Z^2$ is the set of sinks;
- X is the Moore neighborhood;
- $S = \{(g_{i,j}, w_{i,j})\}$ is the set of parameters:
  - $g_{i,j}$ is the terrain altitude,
  - $w_{i,j}$ is the amount of water.
- $\delta : (g^t_{i,j}, w^t_{i,j}) \rightarrow (g^{t+1}_{i,j}, w^{t+1}_{i,j})$ — rule implementing algorithm of water distribution.

A regular grid of Cellular Automata represents a terrain through which the water flows. Arbitrarily defined cells act as sources that supply water to the system. At each step of the simulation, the amount of water in these cells is supplemented to a fixed value. Similarly, another group of cells acts as drains and removes water from the system.

The crucial part of this model is the algorithm of the minimization of differences [176, 165], which calculates the distribution of fluid from a particular cell to its neighborhood. The flow of water is driven by the differences in the water levels in the cells. The water level in a particular cell is calculated as sum of terrain altitude g and amount of water w (see Fig. 2.3).

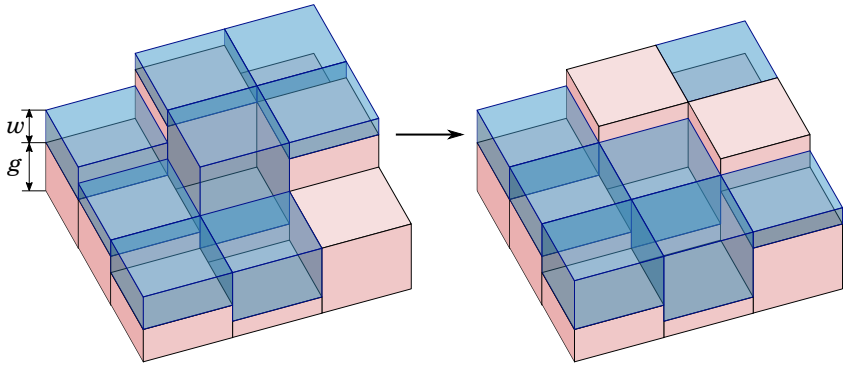

Figure 2.3. Distribution of water between surrounding cells [177]

First, the algorithm calculates an average level for all cells (central and neighboring). Next, cells with a water level higher than the newly calculated average level are eliminated from further calculation. These operations are repeated as long as a cell can be eliminated. Finally, for all cells that not eliminated, the new average level is calculated, and the amount of water that must be moved from the central cell is calculated and stored in an additional temporary array. When all of the cells are already processed, this array is used to update the cells (amounts of water). This rule attempts to equalize the levels of water in this cell as well as its neighbors.

Model of anastomosing river

In order to simulate the phenomenon of an anastomosing river, the model of water flow is connected with a model of the changes in the surrounding environment (the area of the valley). An additional parameter is introduced: thickness of peat layer p. This value is added to altitude parameter g; together, they describe the elevation of a cell. Unlike the altitude parameter, the thickness of the peat layer changes during the simulation.

Cells that contain water are the sources of nutrients, which are distributed to their neighbors. An algorithm of nutrient distribution provides the gradient of nutrient saturation around the cells containing water. At each step, the thickness of the peat layer is increased. The rate of peat growth depends on the nutrient saturation (as well as additional modulating parameters). Another phenomenon that is taken into consideration is evaporation. This factor is necessary because it may lead to situations in which the isolated areas of water might grow infinitely. Some amount of water is constantly removed from all cells.

The areas covered by water (inside the river channel) also grow, but this occurs due to sedimentation. Solid materials carried by the water slowly accumulate at the bottom of the river channel. In the model, this process is represented by an additional parameter.

Concluding, there are three main parameters that control the simulation:

- γ — gradient of nutrient distribution,
- ρ — peat bog vertical growth rate,
- μ — sedimentation rate.

Using the same formality as above, the model for an anastomosing river is defined as follows:

$$\boldsymbol{AN_{CA}} = \langle Z^2, A_I, A_O, X, Q, \sigma \rangle \tag{2.2}$$

- Z^2 is the set of cells with integer coordinators in 2D Euclidean space;
- $A_I \subset Z^2$ is the collection of cells supplying water to the system ("sources");
- $A_O \subset Z^2$ is the collection of cells removing water from the system ("sinks");
- $X(i, j)$ defines the neighborhood of a (i, j) cell;
- the finite set S of states of elementary automaton:

  $$S_{ij} = (g_{ij}, w_{ij}, n_{ij}, p_{ij}),$$

  where the substates are as follows:

  - g_{ij} — cell altitude,
  - w_{ij} — amount of water,
  - n_{ij} — amount of nutrients,
  - p_{ij} — thickness of peat layer;
- $\sigma : S^t \to S^{t+1}$ is a deterministic state transition function for the cells in Z^2. The σ function is composed of three steps performed for each cell:

  1. compute the water distribution to the neighborhood;
  2. compute the nutrient distribution to the neighborhood;
  3. update the thickness of the peat layer.

In Figure 2.4a, a river network is generated in the simulation with a mesh size of 530×530 cells. The terrain has a very small slope (0.05%); additionally, it was disturbed by a random function that creates minor roughness. There is only one source cell, which makes the evolution of the system very slow. For a qualitative assessment, the obtained pattern is compared with a small section of the Narew River (Fig. 2.4b). It is worth emphasizing that a similar backbone and very characteristic small flood areas are observed in both patterns. A detailed presentation of the simulation results as well as their discussion can be found in [178, 179, 180].

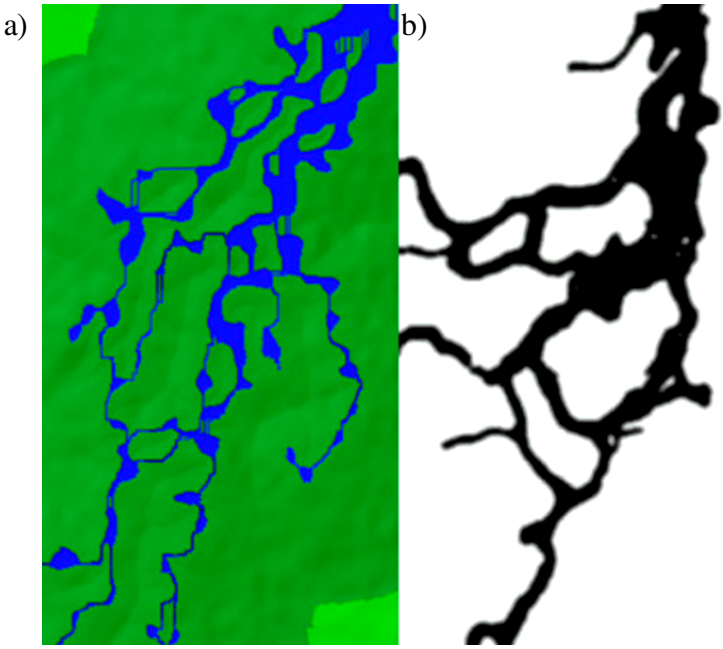

Figure 2.4. Visual comparison of river pattern generated by the model and small fragment of real anastomosing pattern (b) (Narew River from Figure 2.1) [87]

2.3. Multi-scale phenomenon of anastomosing river

The greatest challenge in modeling an anastomosing river is the necessity of joining the single methodology processes that occur in different spatio-temporal scales. The river's speed of flow is several orders of magnitude greater than the environmental changes, such as peat bog growth and sedimentation (years to centuries). Moreover, seasonal events like spring floods that often significantly remodel the network of channels also occur very fast (hours to days).

All the processes and its location in spatio-temporal scales can be visualized in the form of scale map (see Fig. 2.5). Dependencies and flow of information between the components are represented by arrows.

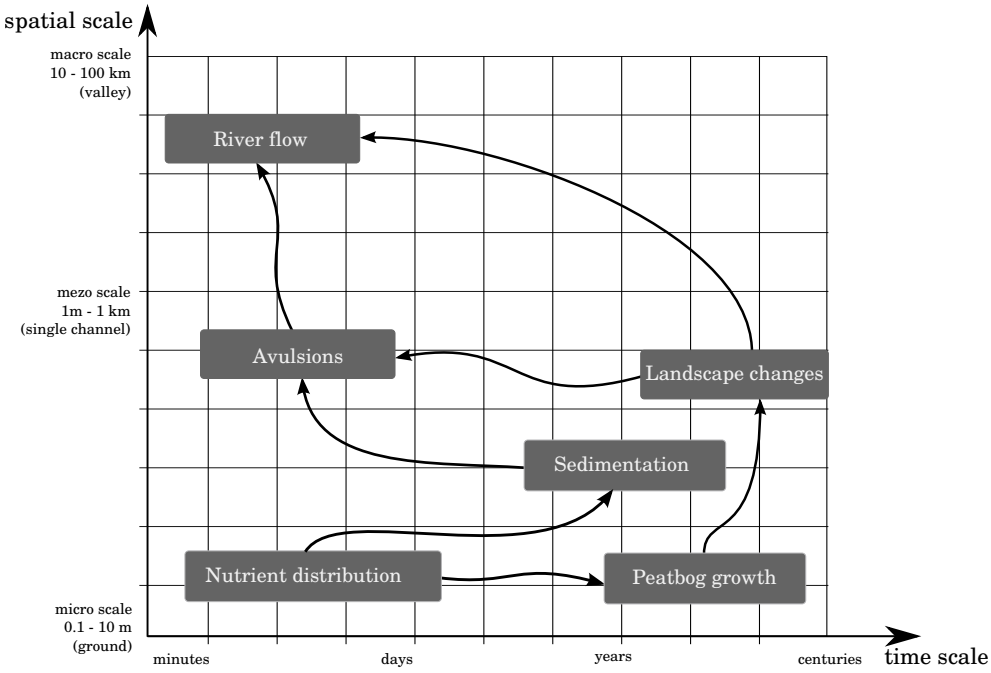

Figure 2.5. Scale map of components of anastomosing river system[2]

The timestep of the model must be adjusted to the fastest processes; i.e., it should last only seconds or minutes. In turn, visible changes in the landscape of a peat bog might appear only after dozens of years. Additionally, the terrain must be represented by using high-resolution grids in order to precisely reflect its shape and the pattern of the river channels. The very slow propagation of changes through such a dense and large mesh is another problem. As a result, the model requires an increased number of simulation steps to produce valuable patterns.

One solution to this problem might be employing a high-performance computing system. In the case of this model, two parallel implementations were proposed: cluster [179] and GPU computing [181]. The other method is to modify the modeling tool. This approach was applied with success and resulted in the definition of a new extension to the Cellular Automata paradigm.

[2] Avulsion is the rapid abandonment of a river channel and the formation of a new river channel

3. High performance simulations with Cellular Automata approach

As stated at the end of the previous section, when Cellular Automata are applied to model phenomena in which the spatio-temporal scale spans over several orders of magnitude, its granularity must be fitted to the smallest and slowest components. As a result, the model becomes quite computationally demanding. Thus, the only solution is to parallelize the computations.

Cellular Automata have several features that make parallel computing mandatory in the models that use this methodology. First of all, Cellular Automata are inherently parallel because the original definition says that all of the cells should be updated at the same time using the same set of rules. The cells are updated using only the information gathered from their closest neighbors.

Before the age of GPUs with programmable processing units, the most popular platform for parallel computing was a system with distributed memory; i.e., clusters. In the case of this architecture, the grid of the cells was simply partitioned among the available computational nodes. The computational nodes only had to synchronize and exchange data for marginal cells.

This chapter starts with a brief presentation of how the model of an anastomosing river was implemented on a cluster using the message passing communication library (MPI — *Message Passing Interface*). This implementation was simple and, in this case, optimal. Much more attention is devoted to the use of a GPU platform, since a single graphic processor is able to provide computing power that surpasses large clusters in the case of some problems. There are potential benefits from implementing Cellular Automata models on a GPU since their data structure and organization of computing are similar to rendering graphic scenes.

The new algorithms and organizations of processing are presented in the cases of two models:

1) surface water flow,
2) pedestrian dynamics.

3.1. Cellular Automata models on cluster platforms

From the beginning, there have been many attempts to find the best parallel platform for CA computing. For several years, Toffoli and Margoulus developed Cellular Automata Machines [119, 182] that were specialized devices with memory and processors optimized for Cellular Automata algorithms. However, the universality of cluster platforms prevails over unusual solutions. These platforms were used together with popular libraries supporting the message passing paradigm such as MPI and PVM (*Parallel Virtual Machines*). There are hundreds of implementations of particular Cellular Automata models for these types of parallel architectures. Also, there were many projects devoted to the design and implementation of universal environments for Cellular Automata simulations, like CAMEL [183] or CARPET [184]. These works were mainly focused on the flexibility and convenience of the proposed tools, as the optimal processing architecture is already included in Cellular Automata.

The nodes in a cluster might be organized into a one-dimensional array: $P_1, \ldots, P_N$, where N stands for the number of nodes participating in a computation. Processor P_1 processes columns from 1 to m, P_2 — from $m+1$ to $2m$, and so on ($m = N/M$, where M stands for the total number of columns in the grid of the Cellular Automata). Each computational node stores two additional columns (which are not processed by this node). They contain a copy of the border cells from the adjacent processors. Such a copy is necessary on the P_i node to update its own border cells. At the end of each timestep, nodes P_i exchange the border columns to its neighbors: P_{i-1} and P_{i+1}.

In the parallel implementation of the Cellular Automata model of an anastomosing river, the grid of the cells was simply divided into stripes and distributed to the computational nodes. The striped partitioning is preferred due to its simplified communication and load-balancing implementation. Such a scheme applies only to the calculation of the water and nutrient distribution. The thickness of the peat layer can be updated without any communication because the appropriate rule uses only the internal states of the cells.

Load balancing can be implemented to this schema in a relatively easy way. In the case of the model of an anastomosing river, many cells are skipped by the algorithm because they do not contain water nor nutrients. Moreover, the active cells are usually grouped into regions — the areas covered by water as well as the surroundings with nutrients. Thus, the load-balancing algorithm divides and distributes the active areas among the available computational nodes (see Fig. 3.1).

In this particular model, the efficiency of the load-balancing algorithms depends on the simulation setup and its further progress. For example, the pattern of flowing water demonstrated in Figure 3.2 almost uniformly covers the whole domain. In such

a case, the method of load balancing described above as well as any other method might only introduce unnecessary additional overhead.

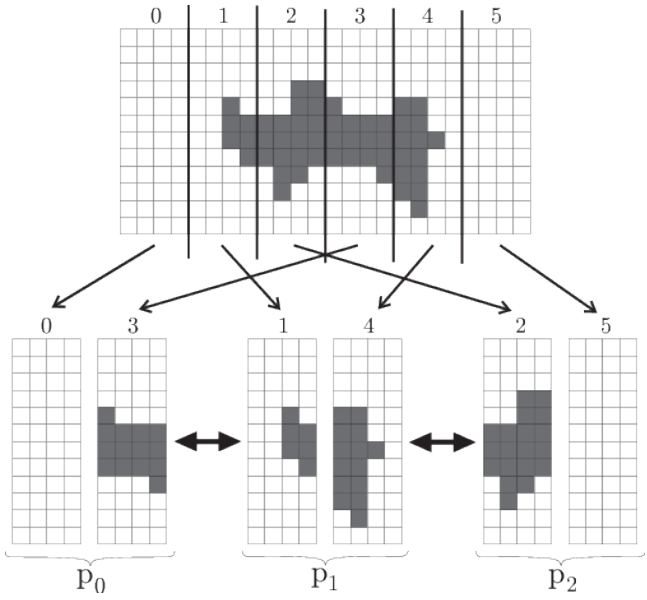

Figure 3.1. Load-balancing method for fine-grained Cellular Automata model of anastomosing river [185]

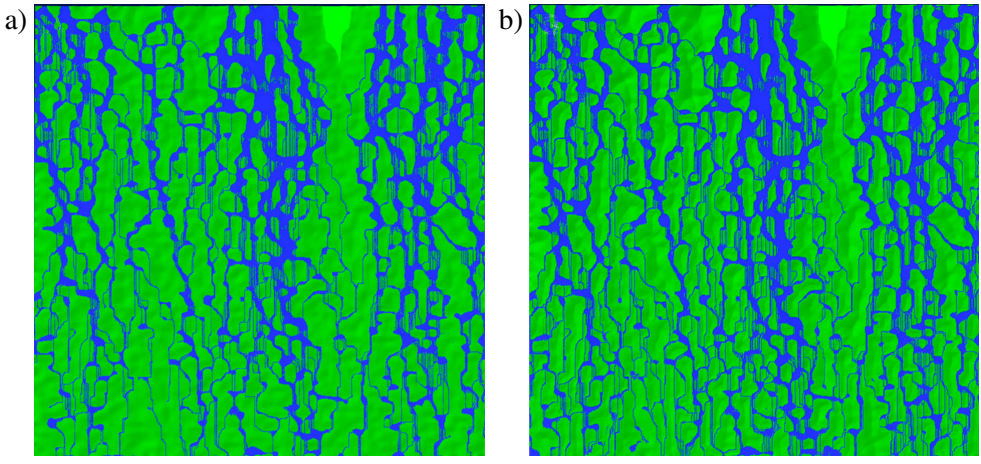

Figure 3.2. Dense river pattern generated using fine-grained Cellular Automata model. Computations were performed on grid with 800 cells. Snapshots were taken after 10^3 (a) and 2×10^4 (b) time-steps [180]

Figure 3.3 presents the results of the performance tests made for the parallel model of anastomosing. It was implemented using the schema presented above. The tests were performed with a cluster of eight nodes with AMD Athlon 2400+ processors connected with a fast communication the interface-based SCI (Scalable Coherent Interface) standard. The measured speedup was close to linear; however, a relatively small number of computations decreased it for smaller grids (left diagram). Also, the load-balancing method mentioned above provided few benefits (right diagram).

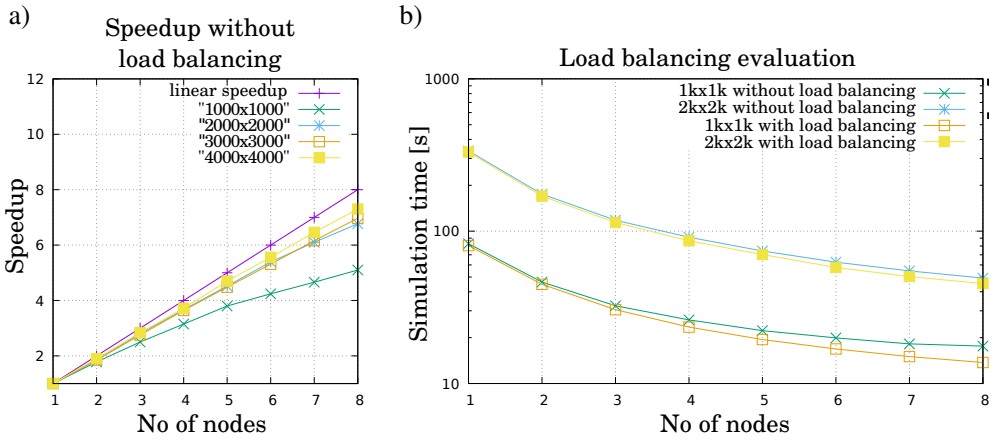

Figure 3.3. Speedups gained in simulations without load balancing involving various sizes of grids (a). Comparison of implementation with load balancing and implementation without this feature (b) [185]

3.2. Accelerating Cellular Automata models with GPU

Graphics Processing Units (GPU) are designed to speed up computations related to rendering computer graphics. Their architecture is optimized for processing a large amount of data with a regular structure in parallel; i.e., a set of vertices that constitute a graphic scene. Cellular Automata have similar features; e.g., a large number of cells that are organized in a regular grid with the same rules applied to each cell. Moreover, all cells should be updated at the same time.

The simple and static schema of data dependency in Cellular Automata is another feature that is useful from the point of view of GPU computing. This fits to the philosophy of using various types of memory in a GPU [177]. The data of neighboring cells can be merged to achieve a high throughput when it is transferred to faster specialized types of memory located close to the processing unit (shared/local memory, registers). All of these features result in the fact that GPU implementation

is potentially very profitable for a Cellular Automata model. The use of graphics processors for simulation with the use of Cellular Automata has been presented and discussed in many papers: [186, 187, 188, 189].

GPU for General-Purpose Computations

The first GPU with programmable shader units was introduced in 2001. It was quickly noticed that graphics processors could be used to solve problems belonging to a more general class. Initially, the only way to perform computations other than rendering graphics was to code an algorithm using an available shader language (e.g., GLSL, Cg). There are several publications that present how various known algorithms can be implemented using shader languages. Basic algorithms of linear algebra were some of the first to be utilized in this fashion [190, 191]. A valuable introduction to mapping general computation problem to a GPU architecture can be found in books "GPU Gems" [192].

Shading languages are optimized for writing code that renders graphics data (vertices, pixels). For obvious reasons, these are not convenient when used for writing code of other types of algorithms. A real breakthrough in using a GPU for general-purpose computing (GPGPU — General-Purpose computation on Graphics Processing Units) was the introduction of CUDA (Common Unified Device Architecture) by Nvidia in 2007. Equipped with programming tools (a compiler, debugger, and profiler), this platform allows coders to write code for Nvidia family processors by using a slightly modified C programming language ("C for CUDA"). For the same purpose, the Khronos Group presented OpenCL (*Open Computing Language* an open programming environment) in 2008. The goal of this project was to provide a unified programming environment for creating programs that used various computational resources available in modern computers: CPU units as well as GPU units.

GPUs are designed and optimized for processing a large number of streams of identical instructions in parallel. Such a processing schema is common in computer graphics when the vertices and fragments are processed by the code of shaders. When applied to general problems, GPUs show their power when the computation task is similar to the rendering task. GPUs can also be used in combination with a traditional CPU platform (see examples in [193, 194]). In this case, the part of the computation that best fits the streaming processing philosophy has to be extracted and delegated for execution on the GPU.

GPU programming uses the concept of stream processing where a sequence of instructions is executed concurrently over all of the items of a dataset. In the CUDA platform, this sequence of instructions forms the so-called kernel, and its execution is named as a thread. Single threads are executed by a streaming processor. Streaming processors are grouped into larger sets (e.g., 8-32 units) and constitute a streaming

multi-processor (SM). Each streaming multiprocessor is equipped with a scheduler that dispatches instructions from the group of threads to the concurrent execution.

A set of threads are logically arranged into a one-, two-, or three-dimensional grid; such a group is called a block. A group of threads that are scheduled together are called a warp. In order to achieve a fully parallel execution, all of the threads in the warp need to do exactly the same operations; otherwise, the threads will be scheduled to be executed one by one. Thus, the full control of the execution paths in a program is a crucial approach for gaining a high efficiency in GPU computing. The simplest method is to assure that a kernel only contains one path of an execution. This can be difficult in the case of more-complex algorithms. The solution in this case is to organize the processing in such a way as to be sure that all threads within a single warp execute the same path.

The GPU memory model includes various types in terms of their speed, latency, and sharing. Registers are the fastest memory, and they are private for threads that are currently processed by the SP (Streaming Processor). The shared memory is almost as fast as the registers, and it is shared by all threads within the block. Global memory is a huge area located off-chip and having the highest latency and lowest throughput. This memory is available for all threads invoked during the execution of a program. In order to balance the deficiency of its performance, a Level 2 cache memory is used to mediate the transfers from the global to the shared memory. The Level 1 cache is also available; each streaming multiprocessor has its own piece of this type of memory.

The term local memory is a bit confusing, especially when CUDA and OpenCL are compared. In the case of the Nvidia proprietary framework, it refers to the part of the global memory that is used to register the spilling; i.e., temporarily storing the registers in the memory. In world of OpenCL, local memory is equivalent to CUDA shared memory.

These various types of memory have to be used correctly. It is necessary to organize the data transfers to fill all of the available bandwidth. This can be made by coalescing a series of individual accesses to the global memory into larger blocks of data (which are treat as single transactions). Coalescing is performed by the hardware, but programmers must obey some rules; i.e., if the threads in a block access consecutive global memory locations, then all of the accesses are combined into a single larger transaction.

GPUs are developing very fast; almost every year, a new generation of processors is proposed by competing vendors (Nvidia and AMD). Apart from the multiplying processing units, these companies constantly introduce improvements into the programming environments. In particular, Nvidia publishes a list of supported features in a document called Compute Capability — the current version is 7.5 (2018). GPU has become more friendly to programmers, and new capabilities have allowed us

to avoid manual optimization. Nevertheless, algorithms still need to be adapted for streaming multiprocessors in order to achieve high performance.

Cellular Automata model of water flow optimized for GPU

When the model of an anastomosing river was designed and developed, GPUs with programmable processing units were not available. When these processors became more popular (and available) and appropriate programming interfaces like CUDA and OpenCL were brought to programmers, the core parts of this model were revised and optimized for the new architecture. The heart of this model (and, at the same time, the most computationally demanding procedure) was the algorithm of water flow. In [181], a new optimized algorithm was proposed. Below, this adaptation is briefly presented to illustrate the susceptibility of Cellular Automata to high optimization on the GPU architecture. First, it was modified to have a single path of execution.

In the original algorithm (see Subsection 2.2 and [165, 181]), neighboring cells are iteratively selected and included into the further calculation until the expected value of the average water level is reached. The new algorithm (see Listing 3.1) starts by sorting neighboring cells according to the level of water (Line 9).

```
1  float min_levels(float w, float* q, float* f){
2  /* ****************************************
3  * w - water in central cell,                *
4  * q[0] - terrain altitude in central cell   *
5  * q[1] ... q[m] terrain altitude and water  *
6  *              in neighboring cells         *
7  * f - flows from central cell to neighbours *
8  ****************************************/
9
10         float* qSorted = sortAscending(q);
11         int i=0;
12         float qSum=0;
13
14         while((i < m) && (w >= i*qSorted[i]-qSum) )
15         {
16                 qSum=qSum+qSorted[i];
17                 i++;
18         }
19         average = (qSum+w)/i;
20         for(int j = 0; i < m-1; j++)
21                 f[j] = max(0, average-q[j]);
22         return average;
23 }
```

Listing 3.1. "Reversed" algorithm for calculating new average level of water

This simple procedure is fundamental for the further organization of the processing and allows us to eliminate the conditional instructions that are present in the original algorithm. For this purpose, a static sorting network is used (which also does not need any conditional instructions). Next, the algorithm iteratively tests which of the neighboring cells can receive water from a central cell (Lines 4-6). The loop stops when the level of water in currently tested cell $gSorted[i]$ is equal to or greater than the average level of the water in the central cell and the previously included cells (condition in Line 4). At the end, the algorithm calculates an amount of water that must be exchanged between cells and stores it in a temporary array.

The algorithm in this form has been implemented for execution on a normal CPU in order to compare it to the GPU version. For running this algorithm on a GPU, it has been split into two kernels:

1) `gpu_best_levels`, (see Listing 3.2), calculates expected average level in central cell and its neighbors and stores in `best_levels[]` array.
2) `gpu_distribute`, (see Listing 3.3), update water amounts in cells.

The grid was partitioned among the blocks of threads. Each thread processed only one cell. The first implementation only used the global memory. The data was stored in arrays, and the memory coalescing for the store and read operations is achieved automatically. Only the cells located at the edges of blocks cannot be coalesced.

The calculations are performed without branch instructions; they is emulated with built-in OpenCL functions `step` (`step`*(x, y)* return 0.0 if $x < y$, else 1.0) and `max`. Also, a short loop `while` is simply replaced by the sequence of instructions. As a result, we are sure that each of the threads will execute exactly the same sequence of instructions.

```
1  __kernel void gpu_best_levels(__global float level[],
2                                __global float terrain[],
3                                __global float best_levels[]) {
4
5          int x = get_global_id(0), y = get_global_id(1);
6          int w = get_global_size(0), h = get_global_size(1);
7
8          float q[5];
9          q[0] = terrain[x + y*w];
10         q[1] = level[x + (y - 1)*w];
11         q[2] = level[(x - 1) + y*w];
12         q[3] = level[(x + 1) + y*w];
13         q[4] = level[x + (y + 1)*w];
14         best_levels[x + y*w] = best_level(level[x + y*w], q);
15 }
16
```

```
17 float best_level(float w, float q[]) {
18         float qSum = w - q[0];
19         sort5(q);
20         qSum += q[0];
21         int i=1;
22         t = step(q[1], qSum);      qSum += t*q[1]; i += t;
23         t = step(q[2] * 2, qSum); qSum += t*q[2]; i += t;
24         t = step(q[3] * 3, qSum); qSum += t*q[3]; i += t;
25         t = step(q[4] * 4, qSum); qSum += t*q[4]; i += t;
26         return qSum/i;
27 }
28 void sort5(float a[]) {
29         #define SWAP(i, j) {float t=min(a[i], a[j]);
30         a[j]=max(a[i], a[j]); a[i]=t;}
31         SWAP(0, 1);  SWAP(3, 4);  SWAP(2, 4);
32         SWAP(2, 3);  SWAP(0, 3);  SWAP(1, 4);
33         SWAP(0, 2);  SWAP(1, 3);  SWAP(1, 2);
34         #undef SWAP
35 }
```

Listing 3.2. Source code of the **gpu__best_levels(...)** kernel functions (OpenCL syntax)

```
1  __kernel void gpu_distribute(__global float level[],
2                               __global float terrain[],
3                               __global float sources[],
4                               __global float best_levels[]) {
5          int x = get_global_id(0), y = get_global_id(1);
6          int w = get_global_size(0), h = get_global_size(1);
7          float a_level = level[x + y*w];
8          float a_terrain = terrain[x + y*w];
9          float change = sources[x + y*w];
10         change += max(0, best_levels[x + (y - 1)*w] - a_level);
11         change += max(0, best_levels[(x - 1) + y*w] - a_level);
12         change += max(0, best_levels[(x + 1) + y*w] - a_level);
13         change += max(0, best_levels[x + (y + 1)*w] - a_level);
14         change += max(a_terrain, best_levels[x + y*w) - a_level;
15         level[x + y*w] = max(change + a_level, a_terrain);
16 }
```

Listing 3.3. Source code of **gpu_distribute(...)** kernel function (OpenCL syntax)

Performance evaluation

The algorithm was tested for its efficiency on several graphics processors [195, 177]. Nvidia processors were used, as they have much better support for the Linux platform. The tests included early processors with Compute Capability 1.1 (Nvidia

Quadro NVS 140, GeForce 8800 GT G92) and more-advanced models (from 2011) with CC 2.0 (Nvidia GTX460 Fermi). The tests included scenerios of water flowing through the terrain. Figure 3.4 demonstrates the sample results of simulations for two different terrain maps: a) a part of a river embankment, and b) an undulating terrain ("land of lakes").

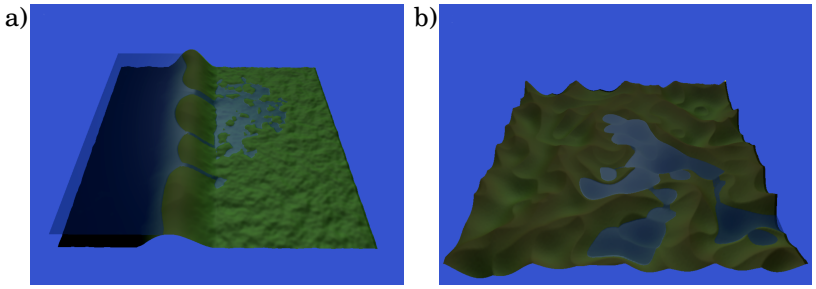

Figure 3.4. Visualization of sample results from simulations: a) pouring water through embankment; b) "land of lakes" [195]

The tests showed that, for this algorithm, GPUs are able to outperform CPUs by 6 to 500 times (see charts in Figure 3.5; a more detailed presentation is available in [195]).

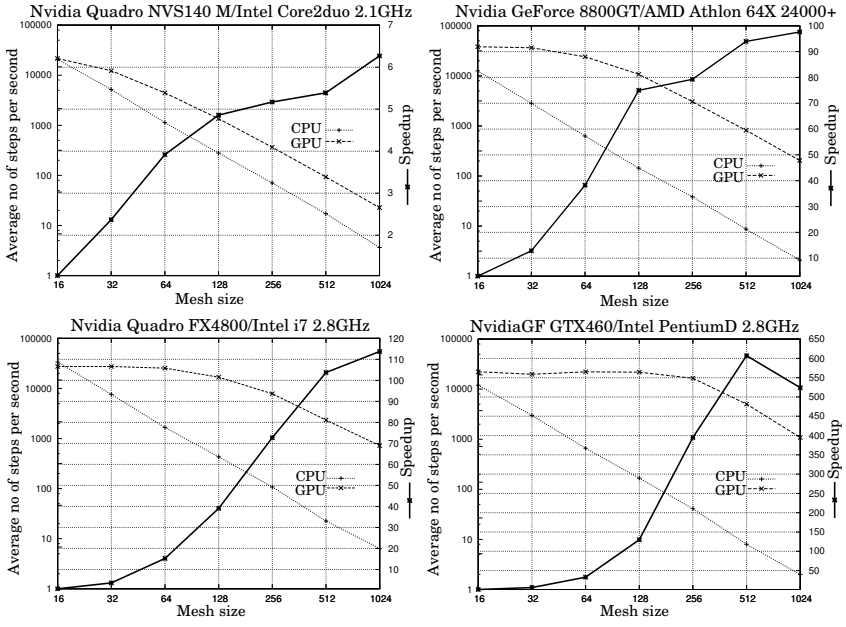

Figure 3.5. Performance of algorithm of water flow for various GPUs [195]

The weakest processor (the Nvidia Quadro NVS 140M with Compute Capability 1.1) was installed in the laptops and had the slowest memory (which is shared with the main memory of the computer). This processor was still better than the Intel Core2, which possesses relatively high performance. The most efficient GPU in these tests was the Nvidia GTX460 (Fermi generation with CC 2.0); it was up to 600 times faster that relatively old Intel Pentium D. However, this CPU was only twice as slow as the most advanced Intel i7. It is also worth emphasizing that, for small-sized grids, the most powerful GPUs are not fully utilized — the average number of steps remains the same for grids with sizes up to 256×256.

Profits from using shared (local) memory

Solving general problems with GPUs requires low-level programming. A programmer must be aware of the GPU's design and how it handles threads and memory transactions. The program must "fit" this architecture. One of the most useful tools for GPU programmers is a profiler. The Nvidia CUDA Toolkit provides profiling tools that give us insight into which way the memory transactions are performed. In particular, older GPUs with Computing Capabilities below 2.0 are vulnerable to incorrectly organized data transfers.

The profiling of the algorithm of water flow is presented in [177]. Briefly, the store operations in this algorithm are performed in an optimal way. This comes from the fact that each thread only saves the calculated data once in the cell that is directly assigned to this cell (see Listing 3.2, Line 13, and Listing 3.3, Line 16). The data is transferred at $64B = 16 \times 4B$ (the workgroup size is 16, and the float data type is used); only this size of transaction is observed in the profiler. Also, their numbers agree with the theoretical calculations; e.g., for a grid size of 512×512, the expected number of transactions is $\frac{512 \times 512}{16} = 16,384$.

The "read" operations require more attention (see Fig. 3.6). Each thread reads data from its own cell (transaction of type 1) and from the four neighboring cells (transactions of types 2-5). Transactions from the central cell and its upper and lower neighbors (Types 1, 2, and 3) are coalesced as a 64B transaction. When the threads read data from the left neighbor (Type 4), the transactions are not aligned to the boundary of 64B, and they cannot be coalesced into a 128B block. As a result, they are realized as one 32B transaction and one 64B transaction. It is marked in diagram that these transactions only partially contain valuable data. A similar situation is observed in the case of Type 5 transactions (right neighbor). They are also not aligned to the boundary of 64B, but they can be realized by 128B transactions (only part of them carry the necessary data).

An analysis with a profiler is able to show another source of non-optimal memory transactions – kernel `gpu_best_levels` read data from two tables (`level`

and `terrain`; see Listing 3.2) and kernel `gpu_distribute` read data from three tables (`level`, `terrain`, and `sources`; see Listing 3.3). A theoretical non-redundant number of 64B reads should be $2 \times 16 \times 1024 = 32,768$ (32 per work-group) for `gpu_best_levels` and $3 \times 16 \times 1024 = 49,152$ (48 per workgroup) for `gpu_distribute`. Respectively, 29% and 33% of the total number of transactions were discovered during the profiling of the real implementation of the algorithm.

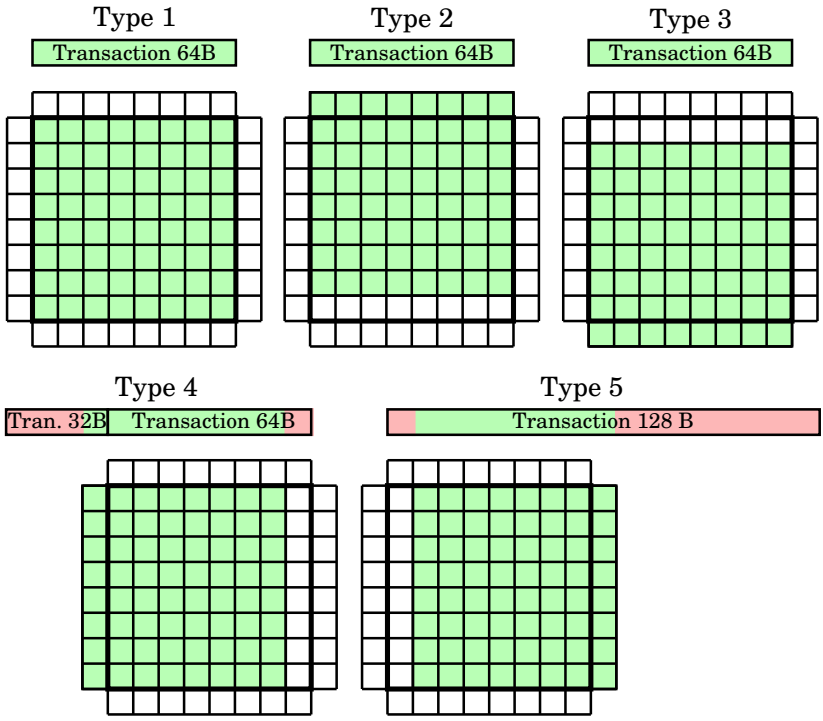

Figure 3.6. Five types of read transactions performed by GPU-optimized algorithm of level minimization. Before calculations, algorithms read data for central cell (Type 1) and its four neighbors (Types 2 to 5). Green colors in transactions indicate data used by threads [177]

The redundancy can be eliminated by manually implementing the managed cache in the local (shared) memory. This type of memory is much faster than global memory, and it features lower latency. In the programming model, it is shared for all threads within the same block. Thus, before the calculations, all threads should read the necessary data into the buffer in the shared memory. Additionally, some threads must retrieve additional data that is necessary for processing cells located at the edge of the block (see Fig. 3.7). The general form of the algorithm remains the same, and the modifications are related to the procedure of loading the data.

Listing 3.4 presents the modified code. Initially, kernel `best_level()` allocates an array in the local memory to store the data (Line 19). The width of the buffer (`BUFFER_WIDTH`) is chosen to prevent bank conflicts when half-warp accesses to the local memory. The cells in one row can be accessed in parallel, as they are located in different banks.

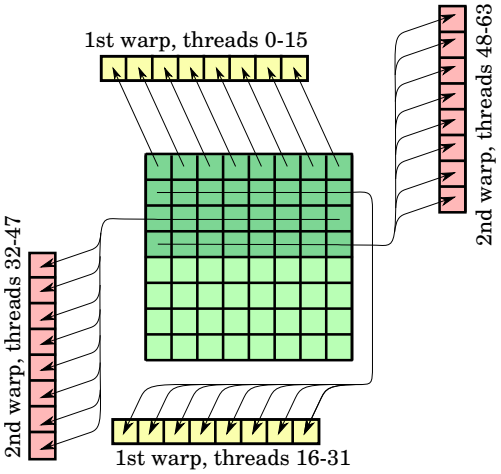

Figure 3.7. Threads located at edge of workgroup must read additional data from area assigned to other workgroups (for clarity, workgroups have size limit of 8×8)

The code in Lines 15-19 implements the loading data for the cells that are assigned to this block of threads (green area in Figure 3.7). Next, some threads must execute an additional code to read the data for the cells from the neighboring blocks (pink and yellow cells). As emphasized earlier, the branch instructions in the algorithm executed on the GPU might disturb the perfect parallel execution. Here, the threads that will execute these additional read transactions are precisely selected to be scheduled together for execution. Thus, it is assured that all of these threads execute the same stream of instructions. This part of the code is located in Lines 21-31. It must also be ensured that no part of the working group tries to read the data that has yet to be stored in the local memory (function `barrier(CLK_LOCAL_MEM_FENCE)`).

Once again, the results of this optimization were verified using the profiler. In theory, the following number of transactions should now be observed (see Fig. 3.7):

- $16 \times$ 64B transactions for cells assigned to workgroup (green area),
- $2 \times$ 64B transactions for above and below the workgroup area (yellow area),
- $32 \times$ 32B transactions for left and right edge of the workgroup area (red area),
- $16 \times$ 64B transactions that read `terrain` table.

The profiling results acknowledged these predictions (see [177] for a full report). This implementation needs only 66 transactions, which is 40% better when compared to the implementation that only uses global memory (16+5× 16 + 16 = 112 transactions).

```
#define BUFF_WIDTH 19
#define BUFF_HEIGHT 18

__kernel void wfca_best_level(__global float* terrain,
                              __global float* level,
                              __global float* best_level) {

int x = get_global_id(0);
int y = get_global_id(1);
int sX = 16, sY = get_global_size(1);
int pX=1, pY = get_global_size(0);

__local float buffer[BUFF_HEIGHT*BUFF_WIDTH];

int pos = x + y * pY;
int lx = get_local_id(0);
int ly = get_local_id(1);
int lpos = (lx+1) + (ly+1) * BUFF_WIDTH;
buffer[lpos] = level[pos];

if ( ly == 0 ) {
  buffer[lx+1]
   = level[x + max(0, (y - ly - 1) * pY)];
  buffer[lx+1+(sX+1)*BUFF_WIDTH]
   = level[x + min(y - ly + sx, sY - 1) * pY];
} else if ( ly == 8 ) {
  buffer[(lx+1)*BUFF_WIDTH+sx+1]
   = level[min(pY - 1, x - lx + sx) + (y - ly + lx) * pY];
  buffer[(lx+1)*BUFF_WIDTH]
   = level[max(0, x - lx - 1) + (y - ly + lx) * pY];
}

barrier(CLK_LOCAL_MEM_FENCE);

/*********************************
 * Here, the code for distributing *
 * water from central cell        *
 * to neighbors                  *
 *********************************/

barrier(CLK_LOCAL_MEM_FENCE);
}
```

Listing 3.4. Kernel `best_level()` with modifications allowing local memory usage

Manually managed cache and hardware cache

As mentioned earlier, the architecture and capabilities of GPUs have developed extremely fast. New features help simplify GPU programming, hiding their unique architecture from programmers. The solution and improvements presented above were efficient for the earliest generations of the Nvidia processors that have Compute Capability (CC) below 2.0. Processors with CC 2.0 and higher have a hardware cache that improves the transactions to and from the global memory. In [196], the efficiency of the implementation presented above with manually managed cache was evaluated on processors that have different compute capabilities.

The Nvidia Geforce 9800 GT (G86 processor) supports Compute Capability 1.0 and possesses 112 unified processing units with GDDR3 memory. The tests showed that this chip runs the water flow model four times faster when the local memory is used as a cache (see Fig. 3.8). The same tests applied on the Nvidia Geforce GTX 480 (Compute Capability 2.0) revealed no differences between the implementation with global memory only and with local memory.

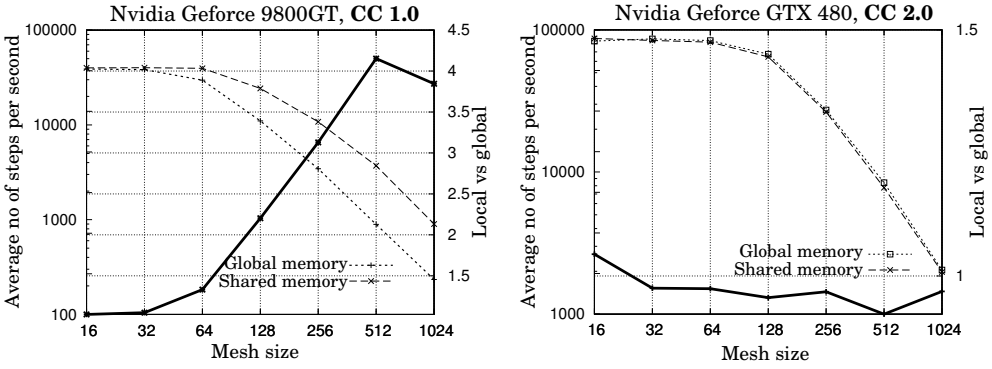

Figure 3.8. Comparison of implementations with global memory only and with local memory (manually managed cache) for two GPUs with different Compute Capabilities

As emphasized earlier, GPU programming is a low-level coding. When a programmer has a deep knowledge of the architecture and its capabilities, he is able to construct code that will be fully optimal. However, such an approach can be very time-consuming, and the resulting code might not be portable to other GPU platforms or when new generations of processors appear. The optimization presented above perfectly illustrates this issue. The method of using the local/shared memory in the algorithm of water flow is smart and efficient, but it is unnecessary when more-modern chips are used. GPU programming is becoming a more high-level programming than in the past. Nonetheless, the philosophy of stream processing still applies. The most efficient method for constructing an optimal algorithm for the GPU

platform is ensuring that several streams of instructions can be executed perfectly in parallel. The effectiveness of this approach was tested on a model of pedestrian dynamics (the results of these tests are presented in the next chapters).

3.3. Large-scale simulations with Cellular Automata models on GPU

The typical approach used in Cellular Automata models of pedestrian dynamics assumes that the domain in which the pedestrians are moving is represented by a regular grid [197, 198]. The cells of this grid can be occupied by an individual, occupied by an obstacle, or empty. Additionally, some cells are defined as points of interests (POIs); e.g., points or areas that are a goal (or goals) for the modeled pedestrians.

The main problem that must be solved in the model of pedestrian dynamics is navigation. In order to precisely model how people navigate in buildings or in open areas, a lot of additional processing should be included; e.g., spotting and recognizing beacons, using memory, and knowledge about the area. One of the most often used methods for approximating all of these decisions is the approach of floor fields (also called potential fields) [58, 199]. This method is based on the assumption that points of interest produce a "virtual substance" that diffuses through the modeled domain. Pedestrians use the level of this substance in the occupied cells and in neighboring cells to decide which direction to go.

There are several extensions to the idea of floor fields. In the most basic configuration, a single static floor field is used. Such a field is generated only once before the simulation starts. A more complex configuration might be used (more than one potential field) in order to reflect a situation when a pedestrian is affected by many POIs. The potential field might also be generated dynamically during the simulation. It can be used to present a changing situation inside a building (e.g., opening new routes for evacuation). It can be also used to reflect some aspects of people's behavior, like the tendency to maintain a certain distance between pedestrians [200, 201]. Other applications of floor fields are presented and discussed in several publications in this area [202]. All of these methods are computationally demanding, but their algorithms are relatively simple and easy to parallelize.

One of the main branches of investigating pedestrian dynamics is the modeling of evacuations in emergency situations [203]. These works have an important practical meaning; the results are used to test whether buildings are safe or to design buildings that facilitate easy evacuation. These models can also be used to test various scenarios of evacuation in the case of huge gatherings and events involving large numbers of participants. This recent application requires the models to be capable of conducting simulations in real time or faster. This means that the model is able to present how

a crowd might behave faster than it actually happens. This is the reason why Cellular Automata models are widely applied in this area (despite the fact that some aspects of pedestrian behavior are not easy to express using this approach).

Social Distance model of pedestrian dynamic accelerated by GPU

The Social Distance model [134] is founded on investigations made by Dirk Helbing and his concept of social force [203]. In general, this model assumes that people protect the area around their body (private sphere) from other people. Thus, each pedestrian can be represented by a particle with a repulsive force. Unlike the original model of Helbing, Wąs uses Cellular Automata. The pedestrians in his model travel through a regular grid with a cell size chosen accordingly to the average area occupied by a standing man (a square with a size of 25cm $\times$ 25cm). The migration of this model onto the GPU platform is presented in a series of papers [204, 205, 61].

A unique solution that was proposed in the Social Distance model was the idea of allowed and forbidden positions. A person can move from one cell to another only if the reciprocal orientation of two people standing in neighboring cells (after the movement) do not break some rules. The authors assumed that a pedestrian body is represented by an ellipse and distinguished 14 combinations of two people standing side by side (see Fig. 3.9). Their mutual influence is quantified (compressibility factor) and used in a rule that governs pedestrian movement. The model uses static floor fields for navigating pedestrians towards the POIs. Dynamic floor fields can also be used to improve the behavior of the crowd.

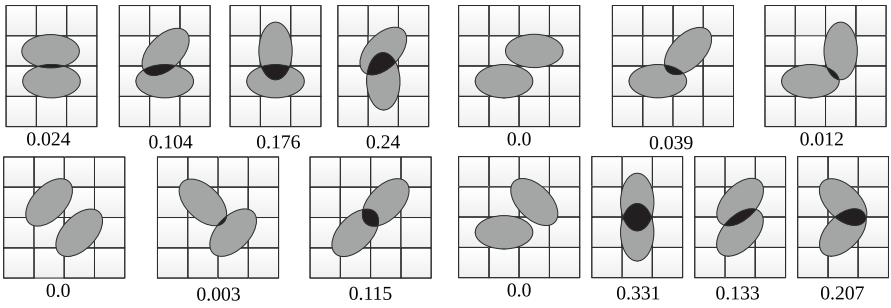

Figure 3.9. Reciprocal orientation of two pedestrians in neighboring cells. Each configuration has weight value (called compressibility factor) assigned, which represent rate of mutual influence [61, 134]

The Social Distance model was extended by various features, which made it a reliable tool for testing evacuation scenarios (i.e., the evacuations of Allianz Arena in Munich, Wisła Stadium in Krakow, and GKS Tychy Stadium) [206, 201]. Its authors

devoted much attention to validation and verification of simulation results [207, 208], therefore this topic is not covered in this monograph. Its main potential area of application requires the ability of simulating large gatherings in real time or faster. The implementation optimized for GPUs is the most obvious choice nowadays.

The original algorithm was never optimized for stream processing and GPU. A first attempt to move this model into the GPU platform was presented in [205]. The algorithms were simply rewritten using C for the CUDA language and prepared for running on Nvidia processors. Even this naive implementation was up to two times faster than the CPU version.

In [60, 61], the Social Distance model was optimized for GPUs. The optimization was profound; their goal was to eliminate multiple paths of execution (i.e. by using lookup tables). Since a modeled pedestrian must make a decision about the direction of his movement based on several factors, the original algorithm generates many divergent execution paths. After an analysis, the three parts of the algorithm were identified as prone for optimization:

1) finding visible areas for pedestrian,
2) calculating the compressibility factor for two pedestrians standing side by side,
3) calculating the cost of pedestrian movement.

The visibility area of a pedestrian means the cells that are available for movement. The shape of this area depends on the current orientation of the pedestrian. A lookup table contains information on the orientations of the pedestrian after performing a movement in any possible direction. The direction is calculated by using the cost function — if the cost of the movement is higher than the assumed threshold, the pedestrian must remain in place.

The compressibility factor is used to resolve situations when two pedestrians are standing side by side after moving. Here, these values are simply stored in a four-dimensional array that is indexed by values: potential direction of movement, orientation of pedestrian, and orientation of his neighbor. When the compressibility factor for a potential location is higher than the defined threshold value, a pedestrian cannot move to a new location.

The cost function is normally calculated by using the following formula:

$$cost(f_{ij}) = S_{ij} + (dens(f_{ij}) + \alpha dist(s, f_{ij})) \cdot W \cdot I \tag{3.1}$$

where, $S_{i,j}$ and $D_{i,j}$ are the values from the static and dynamic floor fields, respectively. $dens(f_{ij}) = e^{\theta D_{i,j}}$ represents the density of the cell neighborhood. $dist(s, f_{ij})$ provides the distance to the nearest POI. Parameters α and θ are weight factors; they were empirically chosen by the authors. Parameters W and I are arbitrarily chosen parameters that control pedestrian behavior near obstacles. The optimization of this stage is achieved by using pre-calculated data. The cost function

is used to determine direction of movement, which is used to move a pedestrian to a new position with a new orientation.

In a single step of the simulation, each pedestrian is processed by a single thread. In order to avoid the necessity of synchronization between threads, the whole population is partitioned into nine groups according to their locations (see Fig. 3.10). This procedure allows us to avoid the situation when two pedestrians want to enter the same cell — cells processed at the same time are at a distance that prevents conflicts.

0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8
0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8
0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8

Figure 3.10. Grid of Cellular Automata is partitioned into 9 element regions; each cell in a region is numbered 0-8. Cells with same index are processed together [61]

Tests of the basic model (without dynamic floor field) showed that implementations for GPU outperform the version for normal processor (see Fig. 3.11). The GPU versions better handles situations in which pedestrians are packed more densely — the ratio between communication effort and calculation fraction is more beneficial.

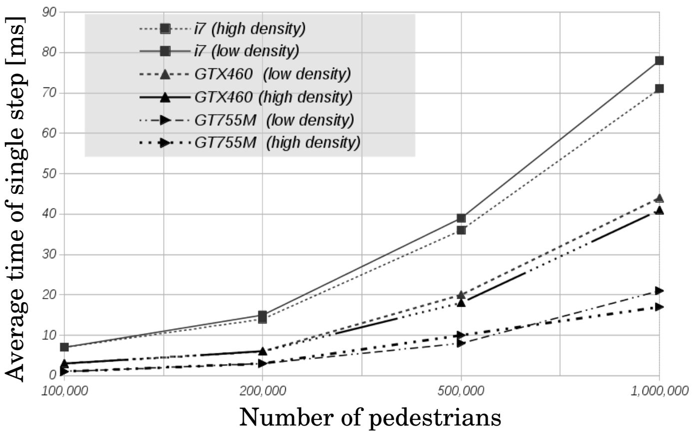

Figure 3.11. Performance evaluation of two Nvidia graphic cards (GTX460 Fermi and GT755M Kepler) and Intel i7 processor. The tests were preformed for two setups with different initial distributions (low and high density) of pedestrians

Another component of the model that can be optimized for GPUs is the calculation of floor fields: static and dynamic. A static floor field is calculated by using an algorithm that resembles diffusion from the POIs. The naive algorithm is simple but laborious — the grid is iteratively updated until all cells receive a proper value. The implementation for GPUs is not straightforward. The algorithm sweeps the grid processing row (or column) in parallel. It interchangeably sweeps the grid in four directions (up to down, left to right, down to up, and right to left). This method results in the fact that threads are invoked for cells that are at the front of the ”wave” of diffusion. Additionally, the algorithm uses the shared memory of the GPU to cache the currently processed rows or columns of cells. The tests (see [61]) showed that the algorithm for GPUs is up to 1000 times faster than the basic method. Normally, the static floor (or floors) are generated once at the beginning of the simulation. The fast method for its updating allows for changing this floor field more frequently as a way of representing the changing locations of POIs.

The dynamic floors that reflect tendency of pedestrians to keep distance to other people change along with the movement of pedestrians. Thus, each cell is simply updated just after calculating the new positions of the pedestrians. These calculations add only a constant factor to whole effort of computation for single step of simulation.

Computations with multiple GPUs

The CUDA platform allows us to distribute the computations over several graphic processors. The Social Distance model was also implemented for such a platform [61]. The most basic approach assumes simple domain partitioning, which means that the grid is distributed among the available processors (see Fig. 3.12).

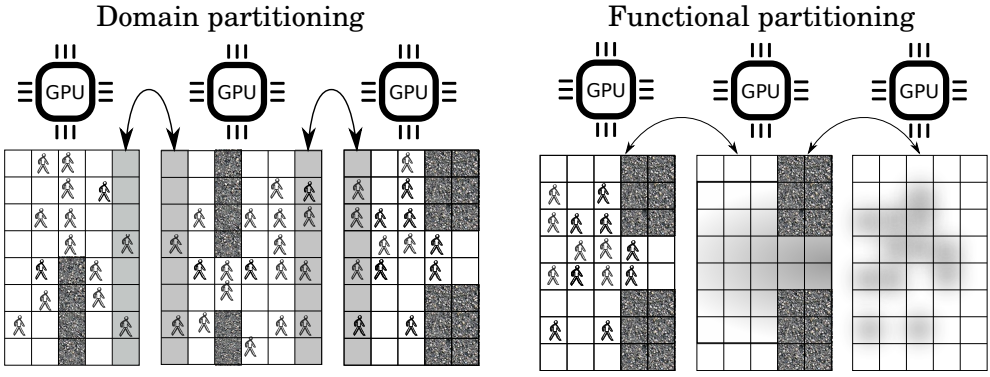

Figure 3.12. Two approaches to using platforms with multiple-GPU for processing models of pedestrian dynamics. In domain partitioning grid is statically or dynamically distributed between available GPUS. Functional partitioning allows to distribute different type of calculations present in the model.

Figure 3.13 presents benefits from using multiple-GPU configuration when domain is simple partitioned among the nodes. The achieved speedup is linear of better depending on numbers of pedestrians involved in simulation.

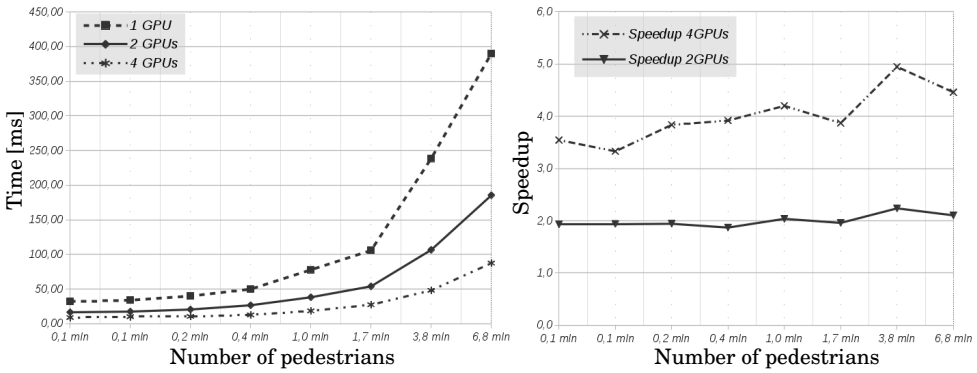

Figure 3.13. Execution times and speedup evaluated using computational platforms with 1, 2 and 4 GPUs (Nvidia GTX580 graphics cards of Fermi architecture, 512 CUDA cores and 1.5 GB memory). Simulation domains were partitioned among the available processors

In the case of the Social Distance model, a single GPU is already powerful enough to handle really large crowds with a high speed — as the testing scenario the meeting of Pope Francis with pilgrims at Krakow's Błonia Park was used (see Fig. 3.14). Compared to a single GPU platform, the use of multiple GPUs mainly allows us to simulate larger gatherings of pedestrians. The implementation with four GPUs is able to simulate up to seven million pedestrians at the same time. However, simulations involving such large gatherings do not have practical values.

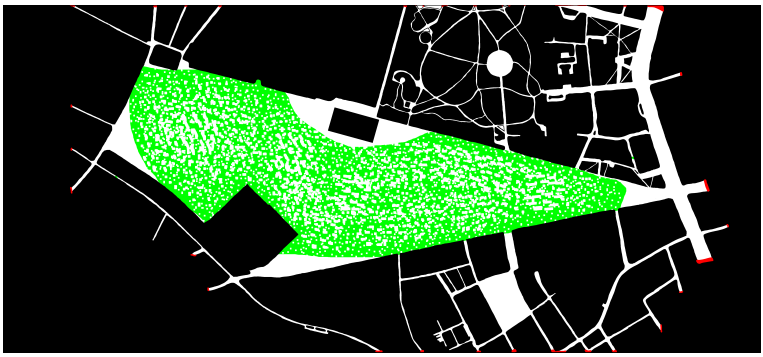

Figure 3.14. Binary map of Krakow's Błonia Park with initial distribution of pedestrians. The park is able to accommodate approximately 2 million people [60]

An alternative method is to employ additional graphic processors to calculate the supplementary data; e.g., additional static or dynamics floor fields gathering statistics (see Figure 3.12) — this approach can be described as functional partitioning. In the case of using the Social Distance model for testing evacuation scenarios, platforms with multiple GPUs can be used to test different variants at the same time. The high-performance computation in this model can be used in systems dedicated to managing large gatherings. An example of such a system is the Khumb Mela Experiment[1]. Khumb Mela is the largest religious festival in the world, which involves the pilgrimage of 100 million Hindus to a sacred river. Unfortunately, such events are often accompanied by accidents and deaths. The Khumb Mela Experiment addresses three research areas in order to understand and manage massive human crowds:

1) data collection (various types of sensors),
2) data analysis,
3) modeling and prediction.

In the third area, the high-performance model is able to provide the simulation results fast enough to properly react in an emergency situation.

3.4. Summary on applying GPU computations for Cellular Automata models

The GPU platform seems to be the best choice for Cellular Automata computations. This type of computation naturally supports fine-grained processing where short functions (kernels) are invoked by a group of threads working in parallel. This also comes from the fact that the GPU was originally designed to process a set of vertices and pixels. Cellular Automata has a similar organization in its data and processing. The method of computation is pretty straightforward — each cell is processed by a single thread. The locality of the interactions also provides benefits in its almost optimal organization of transactions to and from the memory (see [181]). The progress in GPU platforms as well as the accompanying tools also makes programming for this platform the easiest; i.e., the hardware cache provides the same level of efficiency as a fully optimized algorithm that uses shared memory explicitly [177]. Concluding, the implementations presented and discussed above clearly show that the most promising way of optimizing algorithms for Cellular Automata models is to optimize algorithms for stream processing by eliminating multiple paths of execution or by keeping full control on these paths.

[1] http://www.the-kumbh-mela-experiment.com

4. Cellular Automata applied for multiscale phenomena

The Cellular Automata approach is suitable for modeling phenomena that occur on a single spatio-temporal scale. The multicscale systems can be modeled using this approach only when they are combined with other methodology or when some extensions are introduced.

The Cellular Automata approach is believed to better represent microscale interaction between basic components of a systems. In [209] they are used to simulate recrystallization of ferrite grains in the process of rolling steel. On macroscale level the Finite Element Method were employed to model deformation of the material. Information from macroscale level is transferred to the Cellular Automata using interpolation based on SPH (*Smoothed Particle Hydrodynamics*) method. Another example of such the combination is presented in [210] where the Cellular Automata is combined with Lattice Boltzmann Gas to model avascular tumor growth. The Lattice Boltzmann Gas is used to model nutrients diffusion in tissue. The Cellular Automata model tumor development and response of immune system. In both of these methods transfer of information occurs through the parameters that are output for one submodel and input to another.

The multiscale model can be also composed of a few Cellular Automata, each working in different scale. This approach, called a Complex Automata (*CxA*) was proposed in [211, 212]. In fact, their framework is defined on high level of abstraction and they also proposed to extend this approach for agent-based systems and lattice gases. The most challenging issue in case of this framework is definition of mutual dependencies and flow of information between submodels. The Authors used for this purpose scale maps similar to those presented in Chapter 2 in Figure 2.5.

This problem of multiscality is also clearly visible in the case of modeling an anastomosing river. A slowly but constantly developing peat bog (environment) is coupled with a river system that changes in rapid and short events (avulsions). If we want to keep the traditional Cellular Automata framework, all of the processes must be clocked as fast as the slowest one. As it was emphasized in Chapter 2, it requires

some trade-offs between performance and the model's accuracy. An alternative solution is to partially modify the assumptions of the Cellular Automata paradigm. An anastomosing river system consists of two main components. The peat bog (environment) is spatially and temporally homogeneous. At every part of this area, the same rules are used to model the accumulation of nutrients, and the same parameters describe the local conditions (thickness of the peat layer). Unlike a peat bog, the river network is irregular. The channels are randomly distributed and connected. Their parameters (length, capacity) are also diverse. Changes in the network structure occur at random moments and places. In order to precisely model such a system, something having similar structures should be employed. The obvious choice seems to be a graph. In this particular case, an Euclidean graph is the appropriate choice because the nodes have specific positions in an Euclidean plane and the edges are described by Euclidean distances between nodes.

The application of graph theory for defining Cellular Automata models has previously been considered in some theoretical investigations; i.e., in [213], where Caylay graphs were proposed to formally define the Cellular Automata models. This approach is usually proposed only in specific cases (i.e., urban process [214]) because the complexity of algorithms that deal with graphs is higher than algorithms that only use a regular grid.

A graph and the Cellular Automata should be coupled now to establish links that model the interactions between the peat bog and the river systems. In [178, 165, 180, 129] and such, a modeling tool was proposed. This methodology assumes that the graph is constructed over a regular mesh of the Cellular Automata (see Fig. 4.1). Some cells of the Cellular Automata are selected and treated as nodes of the graph. These nodes have positions strictly defined by the lattice; thus, the distance property assigned to the edges is, in fact, redundant information here.

The graph can be predefined, or it can be constructed during the simulation by using additional procedures. It selects cells and includes them in the graph using rules or algorithms that mimic the respective processes in the modeled phenomena. The edges represent additional relationships of the neighborhoods between selected cells (nodes) in the graph of the Cellular Automata.

The cells have two sets of state parameters. The first set is used when Cellular Automata rules are applied. The second set of parameters is active only if the cells belong to the graph, and they are used when the graph is processed. A set of the states can be treated as a channel of communication between the subsystems represented by the lattice and by the graph. These two structures can be processed separately, but a state set in one subsystem can be an input for a second subsystem.

In a further part of this chapter, three examples of using a graph of the Cellular Automata is presented and discussed in detail. The result of these works shows that

the graph of the Cellular Automata can be applied to some more general class of multiscale phenomena in which a transportation network coexists with the environment that produces or consumes the substances transferred by the network. This idea was developed in subsequent applications to became a mature methodology of the modeling of this class of phenomena.

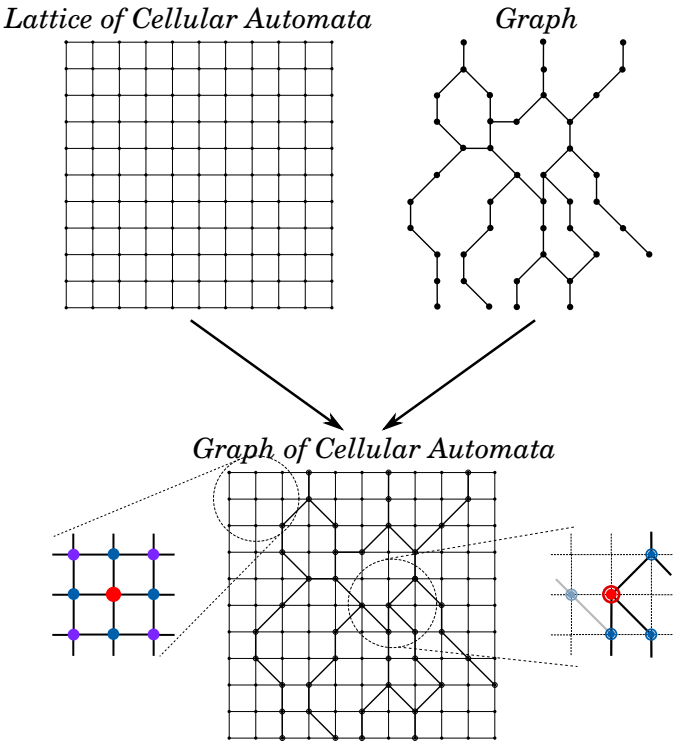

Figure 4.1. Graph of Cellular Automata is combination of regular lattice of automata with graph. There are two types of neighborhoods: classic (von Neumann or More) and another defined by graph structure

4.1. Model of Anastomosing river with Graph of Cellular Automata

The idea presented above was invented and applied for the first time in the model of an anastomosing river [129, 165, 178]. In general, the main assumption that was made in this model was to present the environment (the area of a valley filled with a growing layer of peat bog) using the classic Cellular Automata approach while the river network was represented by the graph.

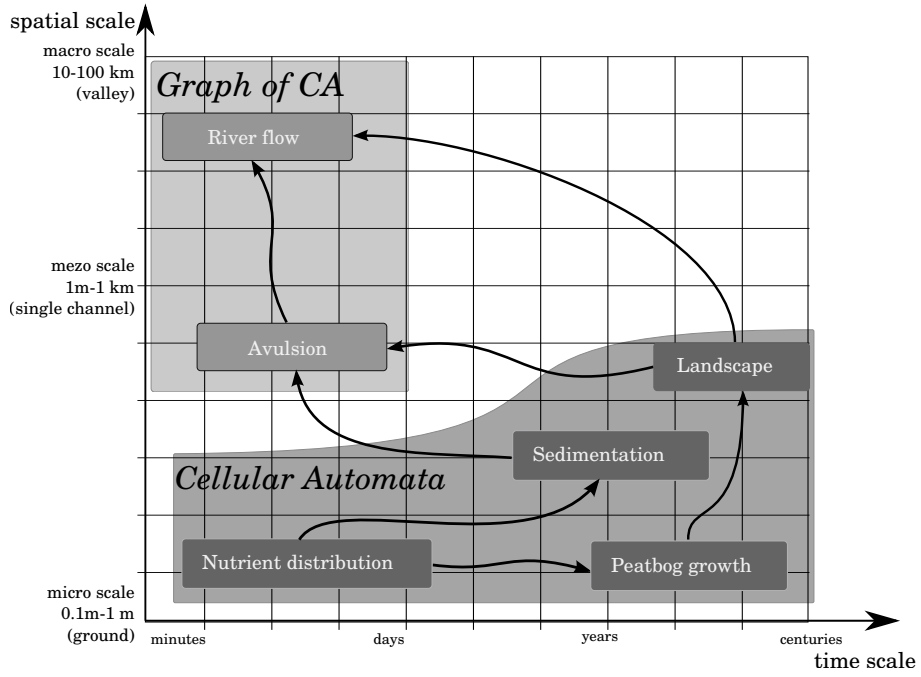

Figure 4.2. Scale map of processes included in model of anastomosing river. Slow and local process are modeled using classic CA approach, and faster global processes are represented using Graph of Cellular Automata

Figure 4.2 presents the components of the model and spatio-temporal scales of the processes modeled by these components. The diagram also presents the dependencies between these components. The slow and fine-grained processes of the nutrient distribution and peat bog growth are simulated by the Cellular Automata methodology. The processes that are faster and act like larger structures (like river channels) are represented using the Graph of Cellular Automata.

This model of an anastomosing river can be formally defined as follows:

$$AN_{GCA} = \langle Z^n, G_{CA}, X_K, S, \delta \rangle \tag{4.1}$$

- Z^n — two-dimensional grid of the Cellular Automata;
- $X_K(i)$ — neighborhood of the i cells in Cellular Automata, usually a Moore or van Neumann neighborhood,
- $G_{CA} = (V_{CA}, E_{CA})$ and:
  - $V_{CA} \subset Z^n$ is the finished set of nodes (which are also the cells in the Cellular Automata grid.)

  - $E_{CA} \subset Z^n \times Z^n$ is the finished set of edges established between two neighboring cells; the edges are treated as an additional relationship of a neighborhood;
- $S = S_{peat} \times S_{river}$ — set of states of each cells;
- S_{peat} — state-related tissue:
  - t_p — altitude parameter,
  - n_p — amounts of nutrients,
  - p_p — height of peat layer;
- S_{river} — state related to vascular network:
  - s_r — size of channel,
  - f_r — flow value in channel;
- $\delta : S^t \rightarrow S^{t+1}$ — set of rules applied to model in each time step:
  - rule for nutrient distribution,
  - rule for peat bog growth,
  - rule for flow updating in river network (applied asynchronously),
  - rule for channel aggregation.

The evolution of the peat bog is driven by the same rule that is used in the case of a pure Cellular Automata model. Each cell was updated using the following two rules:

- nutrients diffuse from cells marked as source of nutrients,
- peat bog layer grows proportionally to nutrient level.

The simulation algorithm of the models consists of four main steps that are executed at each timestep:

1. Update throughput (graph).
2. Update flow (flow).
3. Diffuse nutrient (lattice).
4. Update peat bog thickness (lattice).

The source of nutrients is each cell that belongs to the graph. Using the classical Cellular Automata diffusion rule, the nutrients are disseminated over the lattice. During the simulation, the graph develops by creating new branches. A branch is constructed by an algorithm that selects consecutive neighboring cells that form a path. In the case of this model, such a path usually ends up when it joins up with any existing path. This procedure is repeated many times in different parts of the networks, leading to the creation of complex hierarchical networks similar to those created by anastomosing rivers.

Branch creation is initiated when a channel is fully or partially blocked. It occurs when the value of the throughput drops below the flow value or when the flow value exceeds the throughput. According to the hypothesis stated by Gradziński et al. [173], the river channels are slowly blocked by organic and inorganic materials sedimented in the riverbed. When the throughput in any cell belonging to the graph decreases below the flow rate, a bypassing channel is created. The same situation might appear when the flow of water must be updated due to changes in another part of the network; in the new configuration, a more intense flow is directed to a narrow channel.

The blockade in a cell triggers the procedure of creating a new branch. The algorithm that traces a new branch across the lattice uses the steepest descent path method with an extension that is used when a branch meets a local depression. In reality, when a newly created channel meets a local depression, water fills the depression and pours over its banks. The channel develops downwards until it meets another channel. In the model, this situation is approximated due to the modeling framework [129].

Single changes in the local network configuration might trigger a cascade of consecutive events that will induce the creation of several new branches. The network will modify this structure until it finds a new balance in which the flowing water is not blocked in any part of the network. After that, the system of the anastomosing river develops slowly until a new blockade appears. This behavior partially displays the property of Self-Organized Criticality [215]. The system evolves towards the critical state in which even a small change triggers a cascade of events that significantly modify the system and pushes it back into a more stable configuration.

Figure 4.3 presents samples of anastomosing networks generated using that model. More simulation results (with discussions) can be found in [129, 216].

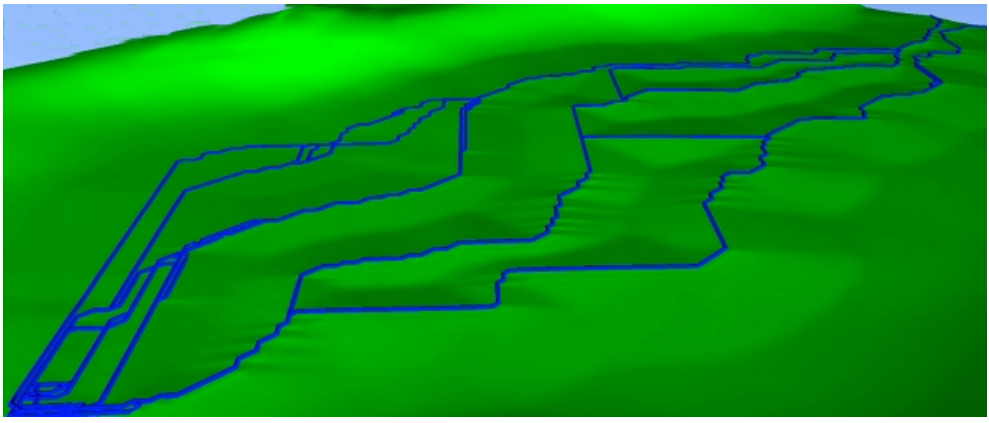

Figure 4.3. Sample pattern of anastomosing river obtained through simulation with Graph of Cellular Automata. Elevation of terrain near river channel due to peat bog growth is clearly visible

Unlike the classic Cellular Automata, the graph of cells is processed asynchronously; i.e., the cells are updated in the order defined by the graph. However, rules that modify the state of a cell without using information from its neighborhood (e.g., aggregation rule) can also be applied synchronously, which benefits the performance.

The use of graphs simplifies the simulation of water flow through the river network. In the simplest model, the flow rate that is set at the starting cell is propagated to the subsequent cells in the graph. At the point of bifurcation, the flow rate is divided proportionally to the throughputs in the branches.

Analysis of anastomosing rivers using graph descriptors

A graph is a quite natural method for representing networks. It facilities network analysis with several types of graph descriptors that are created to evaluate the graph and networks [217, 218]. In the particular case of an anastomosing river, the typical graph descriptors are not very valuable. The network of an anastomosing river does not have interesting properties like scale-free or small-world networks. Only the hierarchical structure of the anastomoses shows some properties of fractality and self-similarity.

Another problem is that there are no investigations that examine the properties displayed by networks of anastomosing rivers. This is mainly due to the fact that this type of river is relatively rarely observed. Moreover, for a long period of time, such rivers were classified together with braided rivers [219] that have a partially similar pattern but are created as a result of completely different factors.

In fact, analyses of the structure, fractality, and scaling properties of river networks has been discussed several times in publications from areas of geology and hydrology [220, 221, 222, 223]. Once again, they are focused on river networks that have branching structures. The results of investigations involving descriptors that were originally developed for rivers with branching patterns were presented in [129]. There, two methods were used: Tokunaga and Horton-Strahler statistics [224]. Both methods define a set of parameters that are intended to capture the properties of river networks.

In [129], these two methods were ad-hoc adopted to examine the properties of anastomosing networks. An anastomosing network was initially converted into branching networks by pruning the joining channels. The statistics for the real patterns of an anastomosing river (the Narew River) were calculated manually.

The ability of analysis networks represented by the Graph of Cellular Automata were investigated with more details when this methodology was applied to model vascular networks.

Modeling transportation networks in consuming or producing environment

An anastomosing river is a system composed of two components having completely different natures of growth but coupled together with various interactions. The river acts as a transportation network that supplies resources (nutrients) to the surrounding environment (peat bog). The resources disseminate from the water to the soil of the nearby channel and change into a vertical growth of a peat bog layer. The peat bog might be treated as a consuming environment that changes the nutrients into its own progress.

A river network occasionally changes its structure due to blockades caused by growth in the environment. The routes of the new channels are determined by the landform of the peat bog. Local depressions that are the result of a poor nutrient supply attracts the formation of channels. Next, they are filled by the peat created by the more intensive vegetation in this area.

River networks change their patterns over time in such a way that the whole area of its environment develops more or less equally. The local progress of the environment might be faster due to better nutrient delivery. However, a river channel running through this area is also blocked faster. As a result, peat bog progress will be inhibited there. Finally, the river creates an anastomoses that starts to supply other less developed areas. Globally, a river network is a dynamic structure that changes its pattern to supply the whole environment equally. These changes are driven by the environment that inhibits and accelerates its progress where and when necessary.

Based on these observations, the general class of phenomena can be defined. It consists of a transportation network immersed in consuming (or producing) an environment. Several examples of such systems can be observed in nature and engineering. Some of them fit this definition almost perfectly, while others show only a partial similarity. These phenomena might be modeled and analyzed by using the same methodology (possibly with some necessary extensions).

The Graph of Cellular Automata can be used as a simulation framework for modeling the phenomena of a transportation network immersed in consuming (or producing) an environment. In a further part of this chapter, two applications of this framework are presented: a vascular system created due to carcinogenesis, and the formation of mycelium inside infected cereal grains.

4.2. Model of tumor-induced angiogenesis

The vascular system provides a distribution path for blood, which carries oxygen, carbon dioxide, nutrients, and metabolic products to the cell of the body. It has

a hierarchical structure, starting from large vessels leaving the heart through arteries and capillaries and returning through the veins. Normally, it is mainly formed during embryogenesis and later grows and matures during childhood. In adult organisms, the process of vasculogenesis is quiescent as a result of precise balance of several stimulators and inhibitors. Vasculogenesis intensifies during wound healing, placenta forming, and (unfortunately) carcinogenesis [225, 226].

The vascular system and its surrounding tissue can be treated as a transportation network immersed in a consuming and producing environment. In this phenomena, the blood carries oxygen and nutrients that diffuse from the vessels to the surrounding cells. The cells use these resources to develop or merely maintain their vital functions. The cells also produce metabolic products and carbon dioxide, which are removed and transported by the blood to the lungs and kidneys. Unlike an anastomosing river, the vascular system is not a result of long-term interaction of an environment and network. It is formed during embryogenesis and later in childhood governed by physiological processes to became fully functional network supplying all cell in body. Its similarity to the dynamics of an anastomosing river system can only be observed when such the network is created *de novo*, i.e., during tumor-induced angiogenesis.

Rapidly proliferating cancer cells might be supplied by their existing neighboring cells for some time. The further growth of a tumor pushes on the surrounding tissue, and the central part of the tumor might be influenced by hypoxia and a lack of nutrients (see Fig. 4.4). The starving cells produce various substances that activate angiogenesis in the neighboring vessels. The endothelial cells that form the blood vessels start to proliferate and migrate due to chemotaxis towards the highest concentration of TAFs. Finally, they form a new vessel that is closer to the tumor, carrying the necessary resources to the cancer cells.

Intensive investigations conducted over many decades have revealed several factors that influence this process. Some of them stimulate this process, while others inhibit it. These substances acts on various types of cells in order to produce a fully functional network of vessels. Unfortunately, in the case of a tumor, this process goes out of control. The imbalance of these substances results in a pathologically formed network. It is still unable to fulfill its functions, which gives positive feedback to the tumor cells. Finally, the tumor constantly invades the surrounding tissue while the core became necrotic.

There are several anticancer therapies that target tumor-induced angiogenesis. Some of them try to inhibit the process of angiogenesis itself. The goal is to starve the tumor cells until they die. Other therapies act in an opposite way – they try to normalize the vascular network inside the tumor. A functional and effective network facilitates chemotherapy — the drugs are able to penetrate the whole tumor. Unfortunately, both methods are only effective in certain types of cancer.

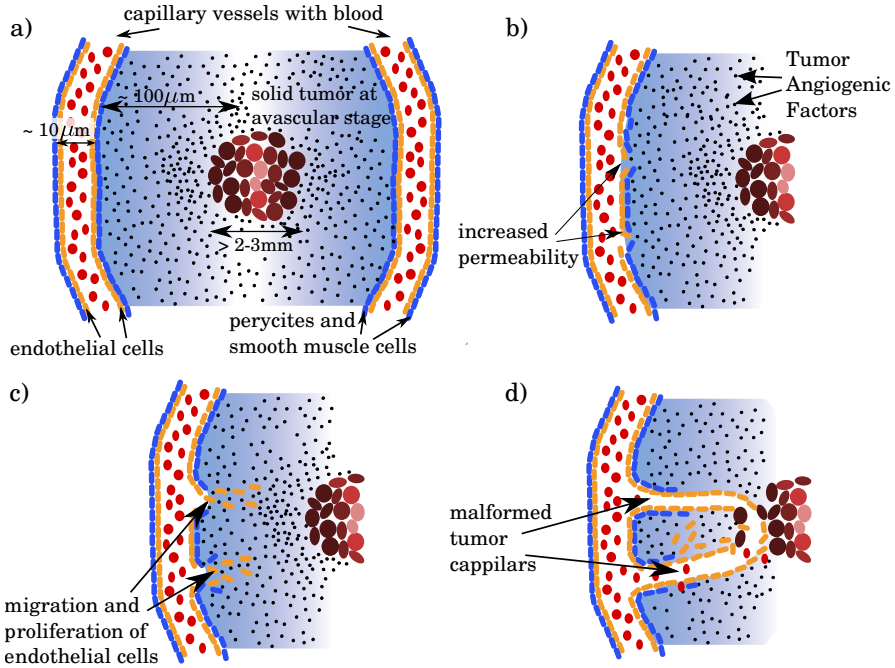

Figure 4.4. Stages of forming new vessels in tumor-induced angiogenesis: a) tumor is able to grow without vessels only to some extent; b) TAFs migrate around tumor an activate cells that form neighboring vessels; b) endothelial cells proliferate and migrate towards highest concentration of TAFs; d) endothelial cells form vessel that is partially able to supply nutrients to starving cancer cells

The purpose of the computer modeling of tumor-induced angiogenesis is twofold. The process of building a model of a particular phenomena allows us to better understand the processes that govern its evolution. The model is, in fact, a synthesis of our entire knowledge of these phenomena. Implementing and running the model verify whether the assumptions are correct, and all of the rules are consistent. Usually, this process opens new questions and problems which also need to be investigated. The second application of computer modeling in this area (especially when a high-quality and precise model is available) is to use it to conduct experiments "in computo" to test possible anticancer treatments.

Computer models of tumor-induced angiogenesis

The computer models of tumor-induced angiogenesis can be divided into two groups: continuous and discrete [227, 228, 229]. Continuous models simply use a set of differential equations to model the diffusion of various angiogenic factors and

cells that are used to form the vessels (endothelial cells, smooth muscle cells, pericytes) [230, 231]. The main advantage of these models is that they use input parameters that can be taken directly from the empirical investigations, like the diffusion rate of various substances in the tissue. On the other hand, these models are not able to represent the various complex interactions that occur between various cells. Also, the scope of the processes that can be included in a model are limited to only those that can be expressed in the form of differential equations.

A discrete representation of the cells constituting the vascular system and surrounding tissue allows us to construct models that are able to include more factors and processes. Cellular Automata with continuous states were the obvious choice for the basic models [227]. This approach has been exploited in several works:[232, 233, 234, 235]. The growing complexity of these models makes agent-based modeling a more appropriate framework in this case [236, 237].

Model of tumor-induced angiogenesis with Graph of Cellular Automata

The application of the Graph of Cellular Automata for modeling tumor-induced angiogenesis partially relies on the model proposed by Anderson and Chaplain [227]. The novelty is that the Graph of Cellular Automata encloses the forming vessels into a convenient spatial structure (graph), while the vessels were only a pattern of activated cells in the original models. The basics of this model were presented in [87]. The framework of the Graph of Cellular Automata can be applied to model tumor-induced angiogenesis with relatively minor modifications as compared to the model of anastomosing rivers. Briefly, the Cellular Automata grid is used to represent a particular type of tissue (see Fig. 4.5). Selected cells or groups of cells in the grid act as tumor cells — the development of tumors was not investigated, and all models discussed below only consider static solid tumors. The cells can also take on other roles; i.e., a source of inhibitors. The Graph of Cellular Automata represents the network of blood vessels. The cells that belong to this graph are the sources of oxygen and nutrients. The distribution of these substances to the surrounding tissue is modeled using the typical Cellular Automata rules for diffusion. The metabolic products are neglected in these models as having an unknown or small influence on inducing angiogenesis.

The tumor cells located too far from the vessel cells can be deprived of oxygen and nutrients. This stress causes them to start producing tumor angiogenic factors (TAF) — a group of various chemical substances that are recognized as having a significant influence on this process. The rules of their distribution produce a gradient of TAF concentration around the tumor cells. When the TAF concentration in a cell that already belongs to the graph exceeds a defined threshold, a new branch (sprout) of

the network is created starting from this cell. This sprout grows towards the highest concentration of TAFs. The branch is constructed by an iterative process that seeks the appropriate cell in the closest neighborhood of the recently added node (a “tip” node) and adds it to the graph. The procedure stops when a branch reaches the area with the highest TAF concentration or when it meets another branch.

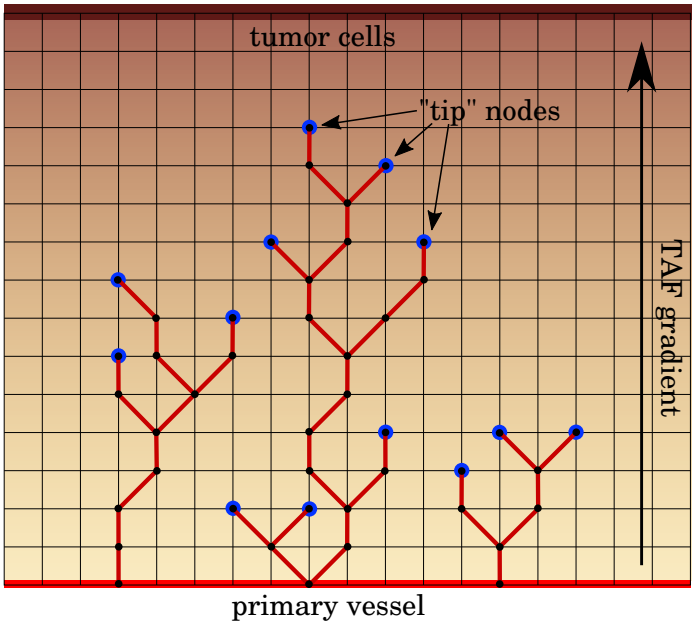

Figure 4.5. Graph of Cellular Automata applied for modeling tumor-induced angiogenesis (see text for detailed explanation)

The model of tumor-induced angiogenesis can be formally defined as the following tuple:

$$TIA_{GCA} = \langle Z^n, G_{CA}, X_K, S, \delta \rangle \tag{4.2}$$

- Z^n — n-dimensional grid of Cellular Automata ($n = 2, 3$);
- $X_K(i)$ — neighborhood of the i cells in the Cellular Automata (usually a Moore or van Neumann neighborhood);
- $G_{CA} = (V_{CA}, E_{CA})$ and:
  - $V_{CA} \subset Z^n$ is a finished set of nodes (which are also cells in a Cellular Automata grid),
  - $E_{CA} \subset Z^n \times Z^n$ is s finished set of edges established between two neighboring cells (the edges are treated as an additional relationship of the neighborhood);

- $S = S_{plant} \times S_{fungi}$ — set of states of each cells;
- S_{tissue} — state related to tissue:
  - t_t — type of tissue cell: “normal,” “tumor”,
  - n_t — amounts of nutrients/oxygen,
  - taf_t — concentration of TAFs (Tumor Angiogenic Factors);
- $S_{vascular}$ — state related to vascular network:
  - t_v — type of cell: “regular,” “tip”,
  - n_v — maturity level;
- $\delta : S^t \rightarrow S^{t+1}$ — set of rules applied to model in each time step:
  - rules for TAFs/oxygen/nutrients,
  - rules for vessel maturation.

The above list of the parameters of the cells and rules of updating might be expanded if the new factors and phenomena are recognized as being relevant to the conducted computational experiment.

The sample results generated by the basic model presented above are demonstrated in Figure 4.6.

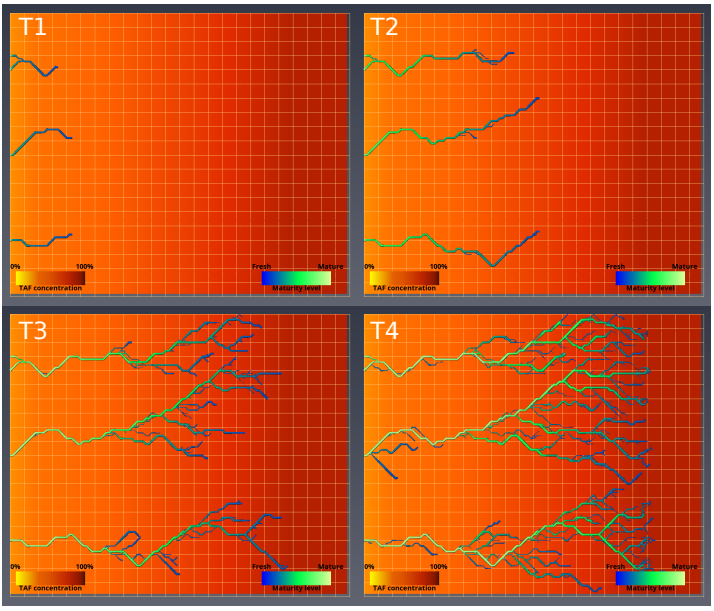

Figure 4.6. Sequence of snapshots of simulation results obtained from basic model (timesteps: T1 $\leq$ T2 $\leq$ T3 $\leq$ T4). TAF concentration and maturity level of vessel are marked in colors

The initial configuration has the tumor cells located on the right edge of the lattice and the primary vessel located on the opposite edge (neither are visualized for clarity). A gradient of the TAFs (the yellow to red colors) is established across the lattice; with a defined probability, they activate some nodes of the primary vessel to form new sprouts. They grow attracted by an increasing concentration of TAFs. Also, the probability of branching increases with the number of TAFs, which is clearly visible in the so-called brush effect; i.e., the rapid increase of vessel density within certain distances from the source of the TAFs.

This basic model can be easily extended by additional types of cells and new rules. For example, a reasonable assumption can be made that says that a branch must mature before it will be able to fulfill its function. This can be simply implemented by using a timer that count the steps until the newly added cells to the graph can reach the state of maturity. The more advanced variant assumes that additional types of cells (like pericytes and smooth muscle cells [238, 239]) participate in the formation and maturation of the vessels. The model assumes that these cells are distributed in the tissue and are attracted by the forming vessels (see Fig. 4.7).

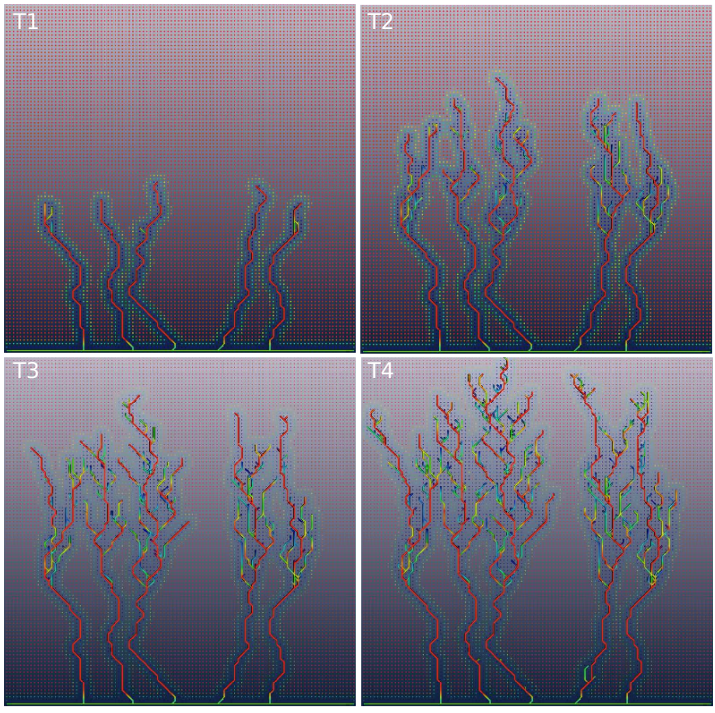

Figure 4.7. Sequence of snapshots from simulation, including existence of covering cells (pericytes). This type of cell is initially uniformly distributed in tissue and not replenished during simulation. Their influence is manifested by slow maturation of small vessels

Another extension that can be easily added to the model of tumor-induced angiogenesis is a submodel of the inhibiting processes. It assumes that some substances are able to counteract the processes of angiogenesis. They can naturally exist in the tissue or can be introduced from the outside (like drugs). These factors can inhibit angiogenesis in various ways; i.e., by inhibiting the proliferation of endothelial cells or by neutralizing the TAFs. Figure 4.8 presents the three-dimensional vascular network that develops in the area with two sources of angiogenic inhibitors.

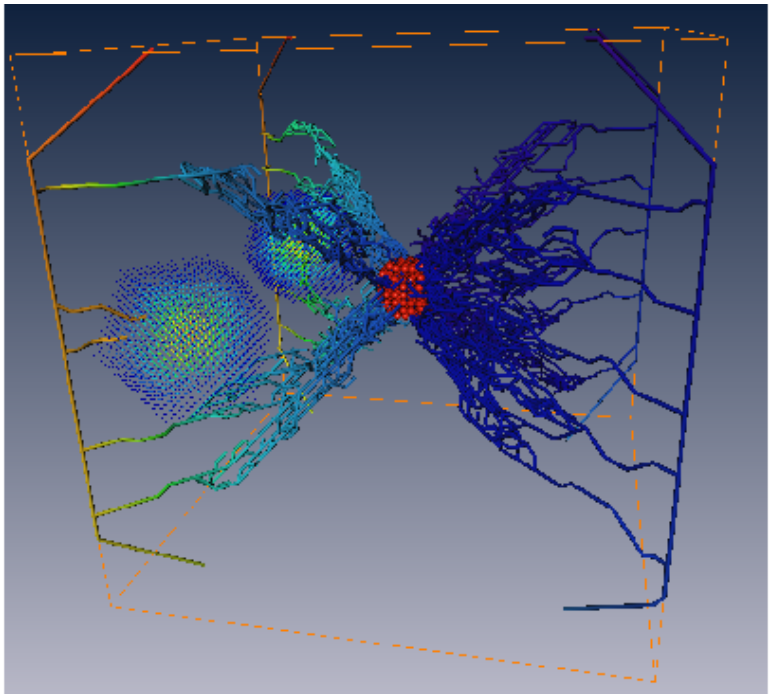

Figure 4.8. Simulation with inhibiting factors. Tumor cells are located at center of computational domain. Vessels grow from four primary vessels located at corners of domain. Two sources of inhibiting substances are located inside domain

Blood-flow calculations in modeled vasculature

The use of a graph for representing a vascular network allows us to implement a model of blood circulation. According to a recent investigation in medicine, this process has a significant meaning to maturing and strengthening newly formed vessels [240]. Vessels that do not transport blood are decomposed and removed. Additionally, the model of blood circulation allows us to precisely determine which areas of the newly formed vasculature are supplied with nutrients.

The other models of angiogenesis with blood circulation [241, 242] usually use a simplified representation of a vascular network; i.e., the network is represented by a regular grid of Cellular Automata. This approach is relatively easy to implement, but it is only an approximation of the chaotic and irregular structure of a vascular network formed during tumor-induced angiogenesis. The graph structure is more accurate in this application.

The flow of blood can be modeled using Poiseuille's Law. It provides information about the flow in a segment of a vessel (tube) formed by the edge between cells i and j:

$$Q_{ij} = \frac{\pi R^4 \Delta P_{ij}}{8 \mu L_{ij}} \tag{4.3}$$

where:

- μ — fluid viscosity,
- $\Delta P_{i,j}$ — pressure drop in segment,
- R_{ij} — radius of segment,
- L_{ij} — length of segment.

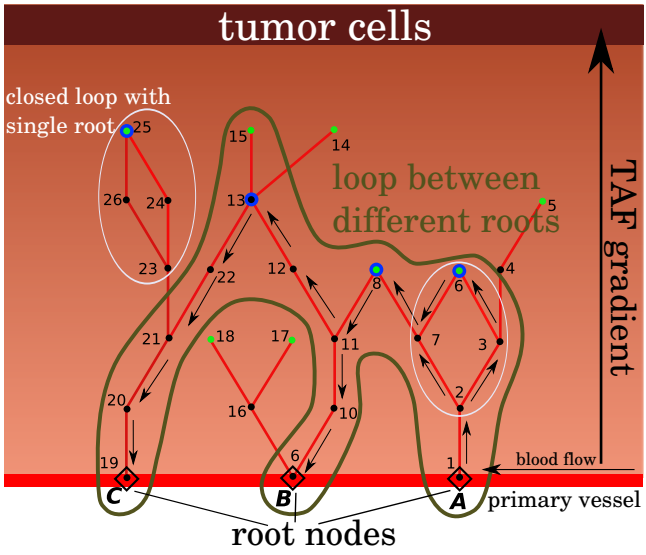

Figure 4.9. Calculating closed loops in vascular network. Green dots represent "tip" cells, and "closing" nodes are marked in blue

The algorithm assumes that blood flow is only possible in vessels that form a closed loop with a difference of pressure at both ends (see Fig. 4.9). Thus, before the computation, any dead-end branches are eliminated. Also, the loops that start and end at the same cell are removed from further computations.

The initial configuration of the vessels (i.e., the primary vessels) has a predefined flow. The cells that constitute these vessels have arbitrarily defined pressure values. Next, during the simulation, the whole network is analyzed each time a new branch is created, and the set of nodes that will be used for flow computation is calculated. In order to find the desired distribution of pressures and flows, a set of equations must be formed and solved.

Initially, closed loops are detected in the graph structure generated by the model. Nodes that belong to the primary vessel are marked as "root" nodes with arbitrarily assigned parameters of flow. The graph is visited starting from each "root" separately. Nodes that can be reached from different "root" cells are marked as "closing" nodes. The paths from the "roots" to "closing" nodes are merged and registered as closed loops The algorithm detects loops that cannot participate in the flow — a loop that starts and ends in the same node cannot transport blood.

For each node j of the graph (cell), the sum of the inflows and outflows must be equal to zero:

$$\sum_{j} Q_{ij} = 0 \tag{4.4}$$

When Q_{ij} is replaced by the Poiseuille's equation (4.3) the formula is obtained:

$$\sum_{j=1}^{j=n} \frac{\pi R_{ij}^4 \Delta P_{ij}}{8\mu L_{ij}} = 0 \tag{4.5}$$

Transforming further, we obtain the following equation for node i:

$$a_i P_i - a_{i1} P_1 - ... - a_{in} P_n = 0 \tag{4.6}$$

where:

- P_i — pressure in node i,
- $P_1, ..., P_n$ — pressures in neighboring nodes,
- $a_i = \sum_{j=1}^{j=n} \frac{R_{ij}^4}{L_{ij}}$,
- $a_{i1}, ..., a_{in}$ parameters calculated as $\frac{R_{ij}}{L_{ij}}$ for $j = 1, ..., n$.

As mentioned above, some of the nodes in the graph are marked as "root" nodes. We set an arbitrarily value of the pressure for that node, which forces the flow.

In order to calculate the pressures in all nodes participating in the flow, we construct a system of equations:

$$\begin{cases} a_{11} P_1 + a_{12} P_2 + ... + a_{1n} P_n = b_1 \\ a_{21} P_1 + a_{22} P_2 + ... + a_{2n} P_n = b_2 \\ ... \\ a_{mn} P_1 + a_{m2} P_2 + ... + a_{mn} P_n = b_m \end{cases} \tag{4.7}$$

The coefficients of the system are defined as follows:

$$a_{ij} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are not neighboring nodes } (i \neq j), \\ -\frac{R_{ij}^4}{L_{ij}} & \text{if } i \text{ i } j \text{ are neighboring nodes,} \\ \sum_{k=1}^{k=s} \frac{R_{ik}^4}{L_{ik}} & \text{if } i = j \text{ and } s \text{ is number of neighbors} \end{cases}$$

and the constant terms are:

$$b_i = \begin{cases} 0 & \text{if there is no roots in neighborhood of node } i, \\ \sum_{k=1}^{k=s} \frac{R_{ik}^4}{L_{ik}} & \text{otherwise and } s \text{ is a number of root neighbors.} \end{cases}$$

Solving the system of equations, we obtain the pressure distribution in all nodes that belong to any of the previously calculated closed loops. Finally, the flows in all of the cells of the vascular network are calculated by using the Poiseuille equation (4.3). Figure 4.10 presents sample simulation results with flow distribution marked on a three-dimensional graph of the cells.

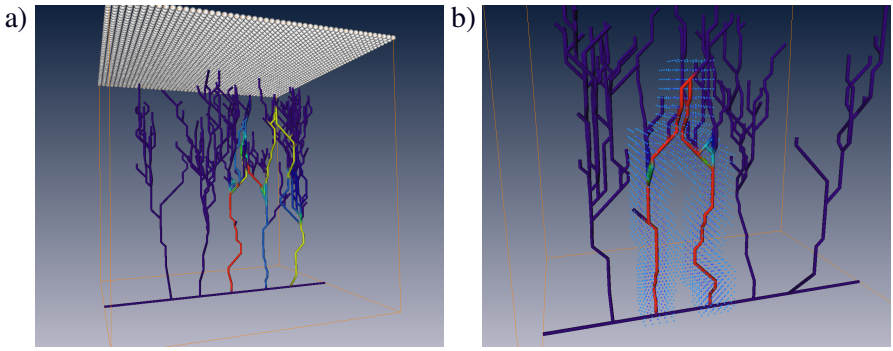

Figure 4.10. Three-dimensional vascular networks generated by model of tumor-induced angiogenesis with blood circulation (a). Flow distribution can be used to precisely model distribution of oxygen and nutrients (b)

Quantitative verification using graph descriptors

The vascular network represented by the graph structure can be quantitatively measured. However, the application of a single descriptor does not display a clear distinction between the structures generated with different sets of parameters. In [243], it was shown that a set of network descriptors can be efficiently applied for classifying vascular network structures.

Some of them were applied to find the features that can be used to distinguished between the networks that were generated by the use of different parameters. The most effective turned out to be:

1. Degree of vertex v [217] is is the number of edges incident with the node.

2. Clustering Coefficients [218] — measures neighborhood connectivity. For single vertex v, it is the ratio of the number of connections between the neighbors of vertex v ($|\{e_{ij}\}|$) to the number of links that could possibly exist between them ($k_v(k_v-1)$):
   $C(v) = \frac{2|\{e_{i,j}\}|}{k_v(k_v-1)}$. The clustering coefficient for the whole graph is calculated as follows: $C(G) = \frac{1}{n}\sum_{v\in V} C(v)$.

3. Graph Efficiency [217] — measures the traffic capacity of a network: $E(G) = \frac{1}{n(n-1)}\sum_{u,v\in V, u\neq v}\frac{1}{d(u,v)}$ ($d(u,v)$ is the distance between vertices u and v).

4. Vertex Efficiency (or Local Efficiency) [217] — $E_{loc}(v) = E(G_v)$, where G_v is a subgraph of neighbors of v and $E(\ldots)$ is graph efficiency.

Apart from the direct values of these descriptors, their statistical properties can also be used in the classification (such as their average or standard deviations [218]). In the Figure 4.11, a few examples of such an analysis are presented. In each case, 4 or 5 sets of parameters were used to conduct several simulations (30 for each set). The differences in the parameters correspond mainly to the various maturations, branchings, and growth ratios. The simulation results generated for these parameters were analyzed using the ready tools for calculating various network descriptors [218]. The results of this analysis form an n-dimensional feature vector (n is the number of applied classifiers). We applied the method of multi-variate analysis (PCA — Principal Component Analysis) to present the results.

In general, the vascular networks generated by the model presented above (as well as in other similar models) properly imitate real vascular networks. This also means that they produce networks that are relatively similar in structure: tree-shaped with a loss of anastomosis. Thus, the analysis with network descriptors does not provide so-spectacular results as in the cases of small world or scale free networks. For some combination of parameters, the networks can be classified using a small number of descriptors with good results (see Fig. 4.12).

A more detailed analysis with the above methodology applied for vascular networks generated by the models with the Graph of Cellular Automata was made by Czech et al. in [218]. Due to the large and complex structure of these networks, Czech developed a GPU-based implementation for these purposes [244].

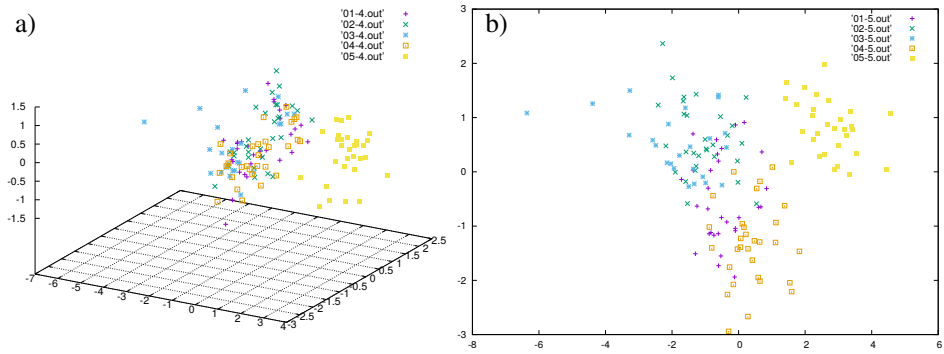

Figure 4.11. Vascular networks analyzed using ensemble of five network descriptors: efficiency, average of vertex efficiency, standard deviation of vertex efficiency, and vertex degree. Five-dimensional feature vectors was projected into three-dimensional space (a) and into two-dimensional space (b) using PCA method

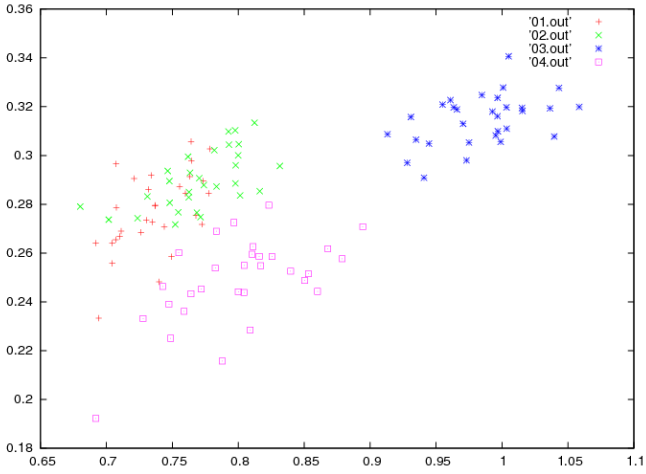

Figure 4.12. Analysis of networks generated using four sets of parameters. Clusterization was made using only two descriptors: standard deviation of Degree, standard deviation of Clustering Coefficient

4.3. Model of *Fusarium graminearum*

The Graph of Cellular automata was also applied to model other phenomena in which transportation networks are surrounded by a producing environment [165]. Fusarium head blight is a fungal disease [245, 246] that causes significant yield losses in wheat crops. Moreover, the mycotoxins produced by this fungus is dangerous to

many living organisms (including humans) when consumed. *Fusarium graminearum* is a species that is most often observed on wheat, corn, and grasses in the United States and Canada [247, 248]. Figure 4.13 presents a mycelium network cultivated in a Petri dish.

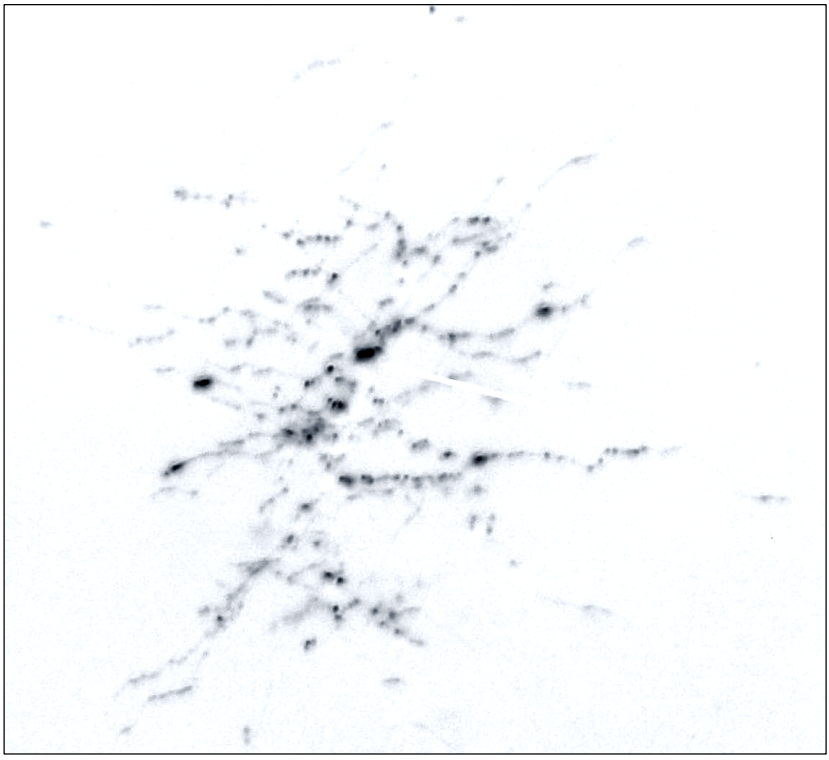

Figure 4.13. Real mycelium network after 22 hours of culture in Petri dish seen under optical microscope. Arrows points to site of inoculation. Food was uniformly distributed over whole dish area (courtesy of Dr. Shea Miller)

The fungus infects the host when its spores are deposited on or inside a spike tissue. Florescence is the period when wheat heads are most susceptible to infection. Certain levels of temperature and humidity raise the chances for infection. Initially, the fungus develops only on the external surfaces of the flower. When the flower opens, the fungus infects the internal tissues. In Figure 4.14, the fungus invading the interiors of the plant stem is visible under fluorescent lighting.

Inside the grain and other plant tissues, the mycelium forms complex anastomosing networks. It grows by extending its tips through the tissue towards the live regions of the plant. The main roles of the network are to gather nutrients (carbohydrates) from the host tissue and distribute them to all parts of the mycelium network.

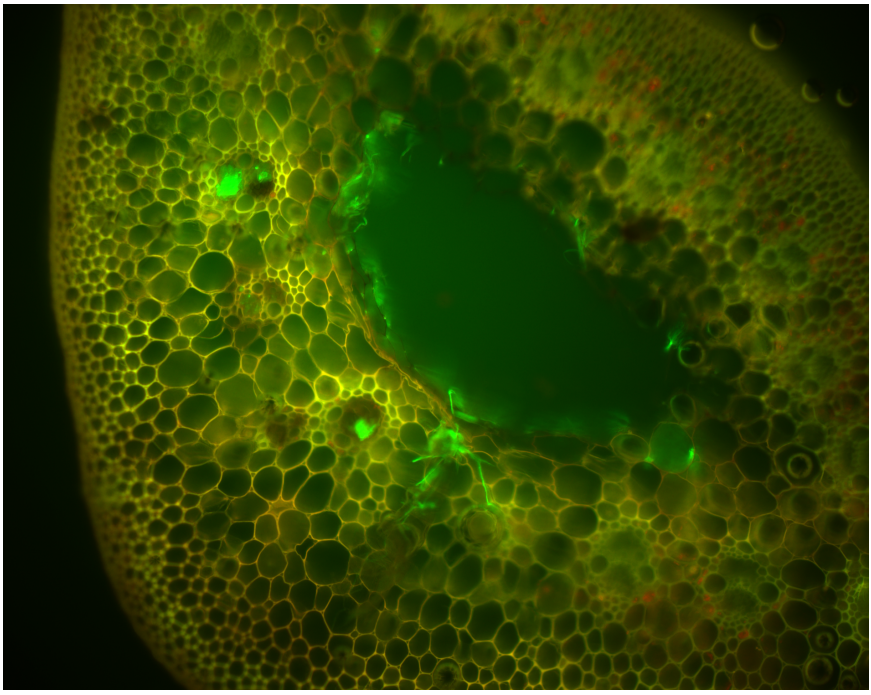

Figure 4.14. Real mycelium network inside plant (cross-section of stem) seen under optical microscope. Hyphae are green under fluorescent lighting. Mycelium grows attached to inner structures of plant

The fungus exploits the host tissue until the tissue dies. The parts of the mycelium that cannot receive nutrients in particular areas might be supplied from other parts of the network. In an extreme case, parts of the mycelium might go into hibernation or even decompose. In such a case, the mycelium network is partitioned into smaller separated pieces that independently develop further.

The general modeling approaches applied to simulating fungal development is slightly similar to those applied for modeling angiogenesis. The crucial component of these models is the rule governing the growth of the hyphal tip [249]. Also, there is a group of pure continuous models that use differential equations to represent the spreading of the fungus over the environment (i.e., soil). These model are only able to show the distribution of particular parameters (e.g., the concentration of mycelium cells or network tips) rather than on the position and shape of a single sprout (hypha) [250, 251].

Discrete models are able to represent single sprouts and model the process of their growth. However, they also use same components that are typical in continuous models. An example of such an approach is presented in [252].

The Cellular Automata approach is also applied in this area. It can be used in its more traditional form, which means a regular grid of cells with the states described by continuous values [253, 254, 255]. Such a model does not distinguish a single hypha but presents the density of the mycelium (like continuous models).

An approach similar to the Graph of Cellular Automata was proposed by Boswell et al. [256, 257]. Instead of an explicit graph structure, an additional triangular lattice only representing the hyphae is used. Additionally, a hexagonal lattice is used to implement a normal Cellular Automata model of the nutrients and product distribution.

Graph of Cellular Automata for modeling mycelium development

The development of mycelium inside plant tissue also might be included in the class of systems consisting of transportation networks immersed in a consuming/producing environment. The mycelium gathers and transports nutrients from the surrounding environment (plant tissue). In this particular case, the network uses these resources to maintain their functioning rather than transport them outside the system.

The following assumptions were made in this model:

- plant cells store a certain amount of nutrients — in this model, all of the processes related to building and maintaining the plant cells are neglected,
- mycelium cells need nutrients to survive,
- the existing mycelium network is immobile,
- mycelium networks create new branches at randomly selected moments,
- mycelium sprouts grow along a straight line with small variations of direction,
- when two mycelium sprouts meet, they join,
- the mycelium network produces enzymes that are disseminated to the surrounding cells,
- mycelium cells gather nutrients from the surrounding plant cells only if their walls are damaged.

A cell in the lattice represents plant tissue. It may be in one of the following states:

- “healthy” — cell represents normal non-infected tissue
- “infected” — cell walls have been damaged by fungus enzymes, and nutrients are retrieved
- “dry” — cell is deprived of any nutrients and is dead.

Figure 4.15 presents how the plant tissue and mycelium network are represented using Graph of Cellular Automata methodology. The cells belonging to the graph can be in one of three states: “tip,” “active,” and “dead.” “Tip” means that these cells represent the end of a sprout that might grow in the next step of the simulation. The

sprout grows by adding cells that are adjacent to the "tip" cell. Unlike the angiogenesis model (where growth is governed by the chemotaxis), the direction of growth is selected randomly here. A new branch can be randomly initiated at any cell that is in the "active" state.

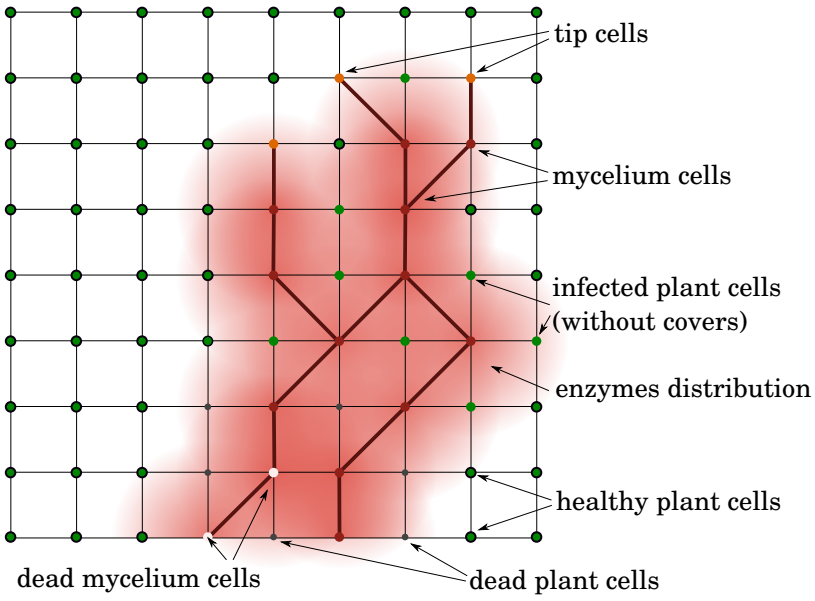

Figure 4.15. Mycelium modeled using Graph of Cellular Automata. Regular lattice of automata represents plant tissue — green/light-green dots represent healthy plant tissue. Graph of Cellular Automata represents mycelium network: brown dots represent active mycelium cells. Tips of sprouts are represented by orange dots at ends of branches. Active mycelium cells produce enzymes that degrade covers of plant cells. Naked cell (green dots) are deprived of nutrients until they became dry (gray dots). Mycelium cells that cannot get nutrients also might die (white dots)

"Active" cells belonging to the graph produce enzymes that are distributed from these cells to the neighborhood. In cells representing plant tissue, a gradient of enzyme concentration appears. When this concentration is higher than a given threshold in a particular cell, its covers start to dissolve. After a certain amount of time (measured in timesteps of the simulation), the covers are removed, and the cell changes its state to "infected" (proportional to its enzyme concentration). Now, the nutrients stored in this cell are transferred to the nearest cell representing the mycelium (belonging to the Graph of Cellular Automata).

Mycelium cells need nutrients to maintain their vital function. When they are surrounded by infected plant tissue (not dry), they might get these nutrients from

the plant. When the nearby plant cells became dry, nutrients can then be transported from other parts of the mycelium network. If the source of nutrients is too far from a particular cell, it will die after a certain period of time (defined as a parameter).

Some investigations suggest that there are two mechanisms of nutrient transport inside the mycelium. Nutrients can diffuse along the network paths; this type is known as "passive." The second mechanism is used in the parts of the network where the growing processes are most intense (tip cells). Nutrients are actively transported towards the tip cells.

Using the formality as in the case of the previous models, the model for fungus development can be defined as follows:

$$FUNGI_{GCA} = \langle Z^n, G_{CA}, X_K, S, \delta \rangle \tag{4.8}$$

- Z^n — n-dimensional rectangular grid of Cellular Automata;
- $X_K(i)$ — Moore neighborhood of i cells in Cellular Automata;
- $G_{CA} = (V_{CA}, E_{CA})$ and:
  - $V_{CA} \in Z^n$ is finished set of nodes (also cells in grid),
  - $E_{CA} \in Z^n \times Z^n$ is finished set of edges established between two neighboring cells; edges are treated as additional relationship of neighborhood;
- $S = S_{plant} \times S_{fungi}$ — set of states of each cell;
- S_{plant} — state related to plant cells (regular grid):
  - n_p — amounts of nutrients stored in plant cell,
  - c_p — condition of plant cell: "healthy," "infected," or "dry,"
  - e_p — amount of mycelium enzymes in plant cell;
- S_{fungi} — state related to fungus cells (graph):
  - n_f — amount of nutrients currently available in fungus cell,
  - c_f — condition of cell: "tip," "active," "inactive," or "dead;"
- $\delta : S^t \rightarrow S^{t+1}$ — set of rules applied to model in each time step.

Figure 4.16a presents the mycelium network generated by the model. It grows in a two-dimensional environment (like a Petri dish) with uniformly distributed nutrients. The growing mycelium absorbs nutrients — the blue regions have low levels of nutrients. During the simulation, parts of the mycelium can be separated from the primary network due to the death of the hyphae — such small parts are clearly visible around the main network. Figure 4.16b presents real mycelium cultivated in a Petri dish (courtesy of Dr. Shea Miller).

Figure 4.17 presents a simulation of a forming mycelium network in a two-dimensional environment. In this case, the initial configuration of the virtual experiment include uniformly distributed nutrient and the mycelium network inoculated to nine cells (a central cell connected to its eight neighbors).

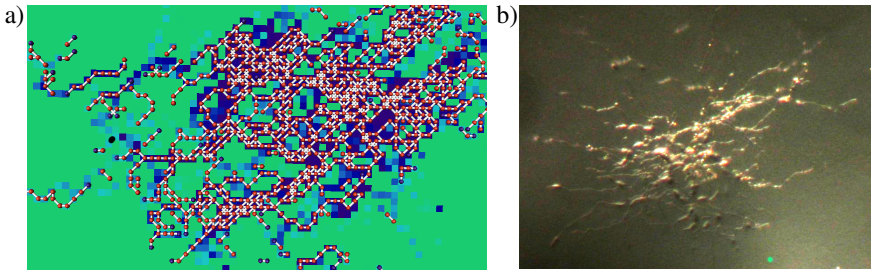

Figure 4.16. Mycelium network: (a) generated by model compared with real mycelium; (b) seen under optical microscope [195]

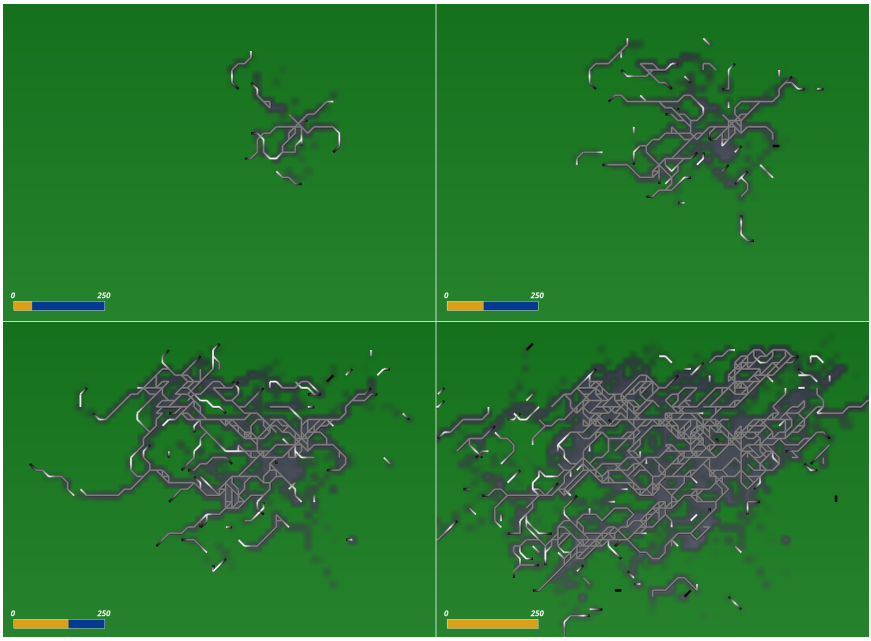

Figure 4.17. Sample results from most-basic simulation. Environment contains uniformly distributed nutrients (green color). Mycelium is inoculated in nine cell (central cell plus eight neighboring cells). Figure presents four moments of simulation (at the 50th, 100th, 150th, and 250th timestep). Dark-gray areas in environment mean dry plant tissue. Black cells in Graph of Cellular Automata represent actively growing sprouts–tips. White parts of graph represent dead segments of mycelium

As expected, the mycelium invades the growing areas of the environment, depriving them of nutrients. The network efficiently transports nutrients from the newly invaded area into the central regions of the network. Surprisingly, the dead fragments

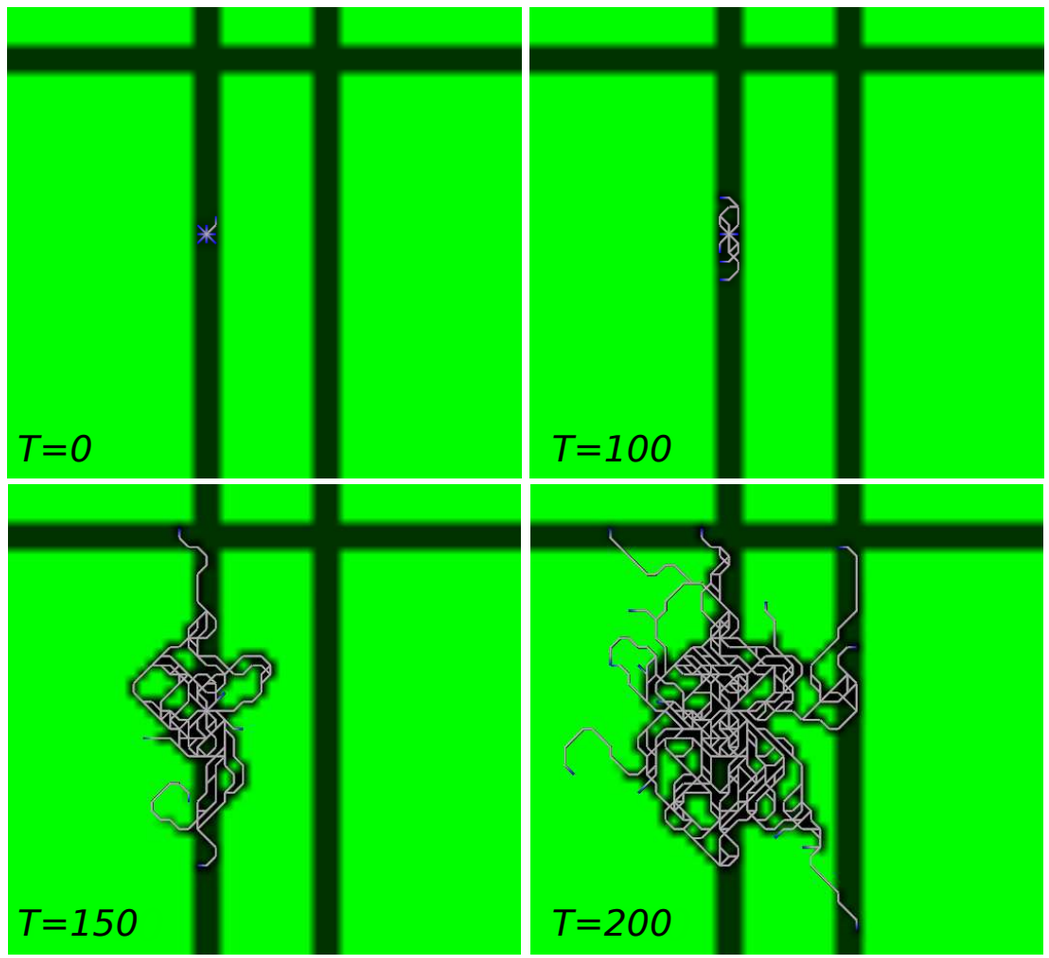

Figure 4.18. Simulation results obtained in experiments with heterogeneous environment. Black areas are almost deprived of nutrients and represent inner part of stem. Green areas represent parts of plant that are rich in nutrients but protected by hard walls. Fungus is initially inoculated inside stem and gradually invades plant after dissolving walls

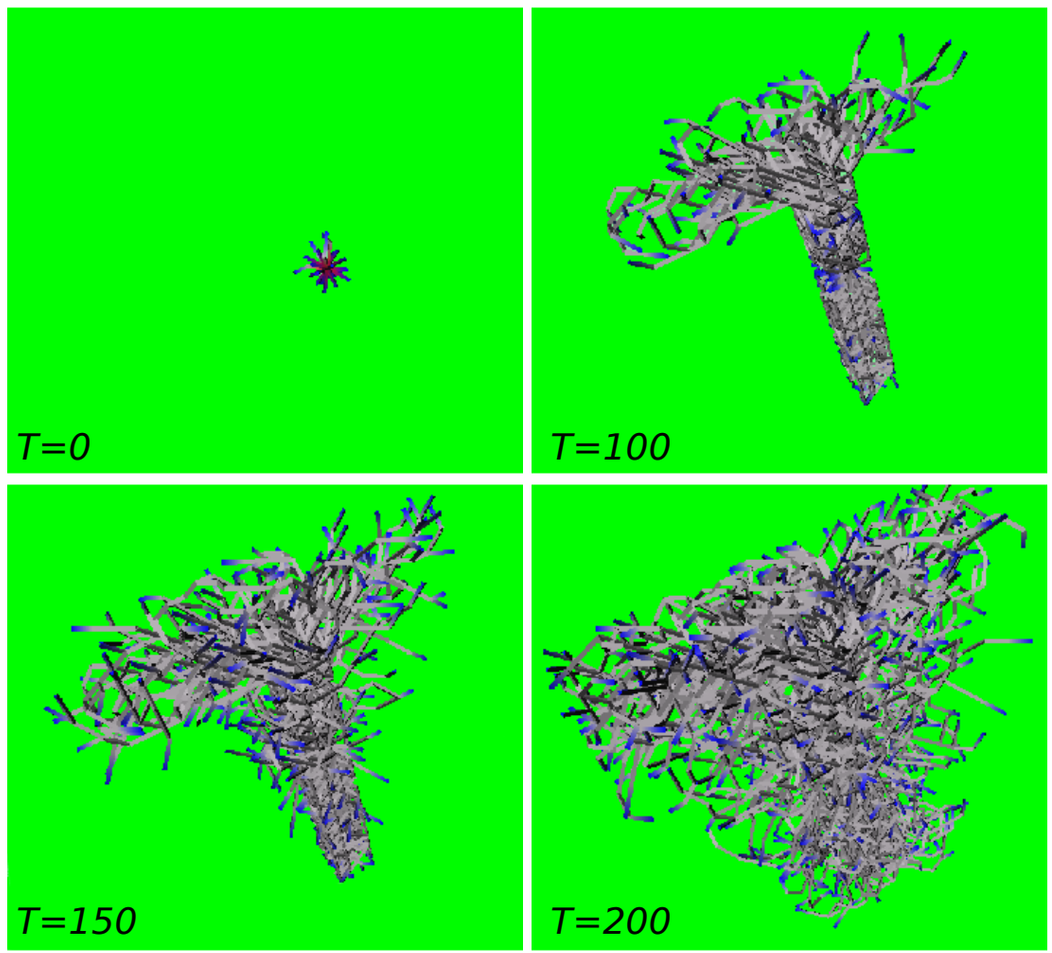

Figure 4.19. Mycelium of Fusarium graminearum growing in three-dimensional space. Virtual stem is T-shaped, and fungus is inoculated in center of this area. Fungus fills this area and gradually invades surrounding tissue

of mycelium are predominantly observed in the newly formed parts of the network, which still are unable to gather nutrients from the surrounding tissue.

The next experiment presented here assumes that mycelium grows in an inhomogeneous environment. The amount of nutrients and properties of the tissue vary in various parts of the plant. Empirical experiments [258] show that, at first, the mycelium (in this case, the *Fusarium graminearum* species) invades the soft and vascular tissues. In the model, such properties can be modeled by differentiating the cell walls resistance to enzymes. Figure 4.18 shows the results of a simulation of such a configuration. The environment is shaped in the form of channels that consist of cells with weak walls and low levels of nutrients (black areas). These channels are surrounded by cells with hard walls and high levels of nutrients. The model follows the aforementioned behavior. Initially, the mycelium sprouts grow inside the channels, and the more-resistant surroundings are attacked later. In a mature network, new sprouts are still more likely to invade the channels.

The main purpose of this model is to simulate mycelium development in three-dimensional space inside the internal structure of various parts of the plant (stems, spikes, and flowers). Figure 4.19 demonstrates the growth of the mycelium in a domain formed in the shape of a "T." Similar to the previous experiments, the mycelium first invades the less resistant areas inside this domain. Next, the sprouts are able to grow towards the more resistant areas outside the domain.

The model of *Fusarium graminearum* is conceptually simple but is relatively complex in terms of parameters. There are more than 20 parameters that control the various included subprocesses. Such a number of parameters results in the fact that its validation and calibration is a difficult task. Of course, some of these parameters are chosen based on empirical investigations. Unfortunately, there are also processes that are modeled using a "black box" methodology; in this case, the parameters must be tuned until the results resemble reality. In this particular model, the "black box" approach is used for all processes of dying — cells that are deprived of nutrients simply change their states to "dry" or "dead" after a defined period of time.

Graph of Cellular Automata accelerated by GPU

The model of *Fusarium graminearum* with the Graph of Cellular Automata was implemented for running on a GPU architecture [195]. Unlike classic Cellular Automata, the Graph of Cellular Automata is relatively difficult to implement with high performance. The irregularity of the data is the obvious reason, but the asynchronous updating also limits the possibility of massive parallel processing. However, there is still room for improvements in these particular models. Mycelium is obviously represented by the Graph of Cellular Automata, but there are no phenomena of directed flow (like blood or water flow) that enforce full asynchronous updating.

The distribution of nutrients in mycelium is modeled by using a simple diffusion rule (applied here in an irregular graph instead of a regular lattice).

As emphasized earlier, the edges in the Graph of Cellular Automata can be treated simply as an additional relationship of the neighborhood. Thus, there is additional information included in the cell data structure in this implementation that allows for the fast mapping of a graph into a regular array of cells. Next, these arrays are transferred into the GPU global memory, and the related kernels are invoked. Although, the graph data is packed into regular arrays, and the performance is reduced by the irregular patterns of access. Detailed tests [195] showed that, depending on the size of the Cellular Automata grid and size of the graph (the number of nodes), the speedup of GPU implementation over the optimized CPU version varies from from tenfold to one-thousandfold.

4.4. Summary of using Graph of Cellular Automata models

The three models presented above demonstrate the effectiveness of the Graph of Cellular Automata framework in modeling certain types of multiscale systems consisting of a transportation network and a consuming or producing environment. As was shown, the framework can be applied to any of these phenomena with no (or few) major changes.

In order to adopt the framework for specific phenomena, only a few components must be replaced or modified. First of all, the formation of a network is a process that is specific for the modeled phenomena. In the cases of an anastomosing river and a vascular network, the graph is constructed by adding nodes on paths towards the highest altitude or TAF concentration. In the model of mycelium, a network is formed by a procedure mimicking random walking. Instead, the environments require almost the same rules that are responsible for distributing nutrients or products.

Such a flexible framework is useful from the point of view of parallel or distributed computation. The computational architecture developed for a specific model can be easily translated to other models.

The idea of modeling dynamic networks with the Cellular Automata approach has been observed in various models developed approximately at the same time by other authors. The main difference lies in the fact that the graphs were not used explicitly in these models. Moreover, they did not use the graph tools and methodology to represent some specific process (like flow or verification of the results). The Graph of Cellular Automata was also the inspiration for other models of phenomena that include networks; i.e., models of tumor development by Dzwinel and Wcisło [259, 260].

5. Cellular Automata as an environment for agents

Cellular Automata can be treated not only as a specific modeling tool but also as a very convenient simulation framework. The data structures and dependencies among them are regular, static, and simple, which provides benefits when the model is implemented and later when it is executed. On the other hand, the usage of classic Cellular Automata approach might significantly limit the possible area of application in some cases. Investigations on population dynamics is an area where Cellular Automata is widely used, albeit with extensions that make them quite similar to another computational paradigm: agent-based modeling. In this chapter, the agent-based model of the ecology of living organisms enclosed within the Cellular Automata framework is discussed.

5.1. Agents and agent-based systems

The terms "agent" and "agent-based systems" were developed in the 80s and 90s of the 20th century. The main source of this paradigm was the idea that some problems can be solved by dividing them into subtasks and integrating the results into one global solution. This methodology is obvious and widespread in parallel and distributed computing, where the processed data is distributed over a set of computational nodes to speed up the calculations. However, an "agent" is definitely something more than a task invoked on a remote computational node.

One of the most often-cited definitions of an agent is as follows (from Woolridge [261]):

> "An agent is a computer system **situated in an environment**, and capable of undertaking **independent**, **autonomous** actions in this environment in order to fulfill **defined objectives**"

This definition emphasizes the fact that an agent is autonomous. The meaning of the concept of autonomy is intuitively known and obvious; however, a more precise definition should be introduced in this context (otherwise, we will be forced to say that "everything is an agent"). In fact, the tasks invoked during parallel computation by the MPI platform can process the data independently in order to perform some particular calculations. In [160], Cetnarowicz made an attempt to indicate which features distinguish agents from other methodologies. He emphasized the two main features that differentiate agents from other types of programming constructions (procedures, objects):

- autonomy from other agents,
- ability to observe the environment.

Cetnarowicz defines the term of autonomy in the context of dependencies between two algorithms. A given algorithm A1 is autonomous with respect to algorithm A2 when A1 is able to proceed with its computations without knowing the state of A2 (it uses only its own state and the state of the environment). Analogously, an agent is independent from another agent when it is able to conduct its computations without knowing of state of that agent.

The ability of observation is defined as the tracking of the changes made in the environment by other agents. The purpose of observations is to build a model of the environment (including the results of the other acting agents). An agent using this model is able to propose an appropriate strategy.

Apart from these two main features, some authors complete this list with the following [262]:

- responsiveness, reactiveness,
- goal-oriented,
- social ability,
- mobility,
- adaptability, learning ability,
- rationality.

Agent-based systems are considered also as a part of Artificial Intelligence researches [263]. In this context, the idea of an intelligent agent evolved over time. Initially, the agents were simply equipped with the algorithms of intelligent behavior like learning or reasoning (deliberative architecture). This approach met substantial problems related to the computational complexity of AI algorithms. Hence, in the 1980s some researchers suggested that intelligent behavior can be a product of an interaction between an agent and its environment (reactive architecture) [264]. This approach also failed in some applications and finally, the hybrid systems became the

most widespread approach [157]. The hybrid architectures were implemented using a layered approach, where lowest levels react to signals from sensors while higher levels use knowledge and models of surrounding environment [265].

Having the term of an agent defined, we can define an agent-based system as a computer system that consists of one or more agents and an environment. Configuration with one agent is possible, but this approach is relatively rarely used. Most applications assume that there are several agents that interact with each other to perform some tasks. The agent-based modeling approach introduces a method for solving some problems. It is a method that is based on a distribution a problem over the population of entities that produces partial solutions that can then be used to propose a global solution.

An agent-based system is also a very convenient paradigm from the point of view of implementation and processing. Object-oriented and component programming paradigms naturally support defining and implementing agents and agent-based systems. The decentralized architecture of ABS supports distributed and parallel computing.

The above definition of an agent-based system allows us to clearly distinguish this methodology from Cellular Automata. The original definition says that Cellular Automata consist of a set of objects that synchronously change their states based on the state of the neighboring cells as well as its own state (from previous time–step). Thus, the cells are not autonomous. Also, it is difficult to clearly indicate what role the environment plays for the cells. However, when various extensions and modifications to the primary definition are considered, autonomy of the cells and environment might be considered.

Multi-agent systems

In traditional agent-based systems, the size of the agents' population is not a crucial issue. The number of agents is selected to ensure the success of the computation but without unnecessary calculations. Also, mutual interaction among the agents is not necessary to obtain the expected results. The agents can simply work independently in order to produce many potential solutions. Nevertheless, cooperation among agents can be also a valuable method of solving some problems. Systems that involve many interacting agents are usually called multi-agent systems (MAS). The agents cooperate or compete with each other in order to fulfill common or individual goals.

Like Cellular Automata, multi-agent systems have found hundreds of applications in models of any phenomena in which the cooperation of many individuals is crucial [261, 266]. In fact, some experts classify Cellular Automata as a special case of a multi-agent system in which the agents (cells) are situated in a space and their behavior is governed by relatively simple rules of local interactions.

Implementations of agent-based systems

As emphasized above, the crucial component of an agent-based model is an autonomous and independent agent. The model might consist of several agents (one is also possible, but rarely) that perform their actions, gather information from the environment, and communicate. An agent-based system is characterized by a lack of global control, decentralized data, asynchrony, non-local interactions, etc. In fact, such a system partially resembles peer-to-peer networks in which computer systems collaborate to provide some services (i.e., file sharing or cryptocurrencies). Thus, similar challenges are posed for efficient implementations.

The programming of the agent-based systems is now supported by various programming techniques like object-oriented programming and component programming. Nonetheless, other issues related to the computational architecture need to be resolved [267, 268]:

- how to decompose problems among agents and how to collect their results into one valuable solution,
- how to organize communication among agents,
- how to allocate resources for agents.

These challenges led to the definition of several standards and platforms for agent-based computing. A well-known example of such a set of standards is FIPA (the Foundation for Intelligent Physical Agents)[1]. This non-profit organization defines the specifications of the interoperability of agent-based systems developed using various programming tools.

The basic infrastructure for agent-based systems is provided by agent platforms. They provide templates for defining the agents, communication protocols, and tools for distributing data and collecting results. There are several publicly available platforms that can be used for building agent-based models, like NetLogo[2], MASON[3], JADE[4], Repast[5], and AgE[6].

Agent-based systems (and especially multi-agent systems) require resources (computational power and memory) in many applications that cannot be provided by a single machine. Nearly all of the platforms mentioned above have versions or extensions that facilitate high-performance computing (see the survey in [269]). In most cases, they have an additional layer that allows for the distribution of computations among the available computational nodes. Usually, this communication is

[1] www.fipa.org
[2] ccl.northwestern.edu/netlogo
[3] cs.gmu.edu/~eclab/projects/mason
[4] jade.tilab.com
[5] repast.github.io
[6] age.agh.edu.pl

provided by one of the well-known libraries, like MPI (Message Passing Interface), RMI (Remote Method Invocation), or JMS (Java Message Service). In order to efficiently exploit the resources, most platforms have various methods for balancing the computing load (static or dynamic).

Individual-based modeling

The paradigm of Individual-based modeling (IBM) [270] was developed in the 1970s for investigations related to ecology. This paradigm assumes that any population can be simulated through the modeling of each member individually. Each individual has its own set of attributes and rules of behavior. These attributes and behaviors might vary among individuals, and they can also change over time due to adaptation and/or as a result of programmed changes (e.g., juveniles versus mature individuals).

Individual-based modeling provides a completely different methodology of modeling a population from traditional methods based on mathematical equations. One of the most popular equation-based models is Lotka-Volterra (also known as the Predator-Prey Model), which simulates a population consisting of two types of individuals. This model uses global parameters like population size as well as birth and death rates. The equations are constructed to reflect how these global parameters evolve over time and how they are connected to each other. The well-known set of equations has the following form:

$$\begin{aligned} \frac{dx}{dt} &= \alpha x - \beta xy \\ \frac{dy}{dt} &= \delta xy - \gamma y \end{aligned} \tag{5.1}$$

The first equation refers to the population of the prey. They have an unlimited amount of food; the term αx represents the exponential growth of this population (α represents the growth rate of the prey), and β represent the effectiveness of the predators. The population of the prey is limited proportionally to both populations. Analogically, the second equation reflects the changes in the population of the predators. Its increase depends on its fertility and the prey numbers. The population of predators is reduced only by the death rate (γ).

The main advantages of the Lotka-Volterra model (and other similar ones [271]) are their simplicity and almost-zero requirements for computing power. However, it is quite difficult to add components that will reflect additional factors in the model of population formulated in this way. In fact, such an extension requires us to find a global process and a model for this factor. Trying to include factors such as the diversity of life in individuals, the heterogeneity of a population, seasonality, and survival strategies is difficult.

A model based on individual-based modeling can be relatively easily extended by new factors only if they can be expressed in the form of rules of behavior. The implementation of multiple species, complex food chains, the spatial heterogeneity of a population, and various life strategies are simple and intuitive [272, 273]. A model based on IBM is built using a bottom-up method. The model is constructed by implementing several of the lowest-level submodels that describe how individuals act in basic situations: moving, gathering food, managing energy, and reacting to other individuals as well as the surrounding environment. This method is easy and convenient for the modeler considering the fact that such basic activities are relatively well-researched in the case of many natural phenomena. The global dynamics of a population should emerge as a result of the interactions among individuals and be compared to what we observe in reality. IBM allows us to observe all levels of a model, from a microscopic (individual) view through a mesoscopic level (i.e., the interaction of one or more individuals) until reaching the macroscopic level (population).

The main drawbacks of IBM are the relatively higher effort necessary to implement the model as well as the high demands for memory and computing power. The model requires a variety of data structures to represent the various objects that are intended to be modeled. It is also necessary to write all of the code that manages these objects and represent the actions of the modeled individuals. During the simulation, the memory for all of these objects must be allocated as well as reliably and efficiently managed when long-time simulations are conducted. All of these requirements results in the fact that simulating a framework for IBM models must be carefully designed and implemented. The necessity for a detailed representation of each individual during the simulation, including the temporary fluctuations of populations (i.e., due to seasonal blooming), requires the computing environment to be efficient and elastic.

At first sight, the general idea of IBM appears to be identical to agent-based modeling. This is mostly true; sometimes, these two names are used interchangeably. However, the origins of these paradigms are located in different scientific disciplines. Anyway, the IBM models can be naturally implemented using ABM methodology and the tools/frameworks developed in this area. What is more, the requirements that IBM sets for its computing platform are met by reliable and highly efficient ABM platforms.

Evolutionary multi-agent systems

Evolution can be seen as a long-lasting process of searching for optimal solutions, which is a trait that helps ensure success in survival. This “natural computation” lasts through hundreds of generations, and each of them proposes and tests a new candidate for an optimal solution. Based on the analogy to this natural process,

a large group of frameworks and algorithms known as evolutionary or genetic algorithms have been proposed [274, 275, 276]. This class of heuristics is often applied to problems that are difficult to solve using traditional approaches. A classic example of such a problem is the global optimization problem [277, 278].

Typical evolutionary computation starts with the generation of an initial pool of random genotypes. Next, the following steps are repeated in consecutive iterations:

1. Generate phenotypes for all genotypes in pool.
2. Evaluate fitness of phenotypes.
3. Remove worst phenotypes and related genotypes.
4. Generate new pool of genotypes from old genotypes using genetic operators: mutation and/or crossing over.

The original definition of a genetic algorithm does not assume that a genotype and phenotype are connected with any individual that has life and exists in any environment. The quality of a genotype (fitness) is not tested through any kind of competition for resources. The evaluation of fitness means that only some indicators are calculated. Such a method significantly speeds up calculations, but it works when the fitness is given using relatively simple and unequivocal formulas.

Adding a genotype to an agent is a way back to the biological inspirations of both paradigms. Such an agent better mimics living organisms because the knowledge that it gains can now be passed on to its offspring, improving the next generations. Agents observe and communicate with other agents and the environment; in this way, new factors and knowledge that influence the evolution might be considered in the model. Moreover, agent-based systems are naturally decentralized. This feature results int the fact that the evolution of agents also proceeds in a decentralized and even parallel manner. A population of evolving agents might discover various solutions in parallel.

Agent-based systems in which the agents are equipped with genotypes, the ability to reproduce, and inheritance are called EMASs (Evolutionary Multi-Agent Systems) [279, 280, 281, 282], This approach assumes that a problem is solved using a population of agents whose change is controlled by the rules of reproduction, inheritance, and "natural" selection. Natural selection in EMAS simply means the death of an agent. Usually, this happens when an agent exhausts its life energy. This reserve can be replenished by an agent by gathering nutrients from the environment, for example. However, the environmental resources are limited and shared among the agents.

Unlike the evolutionary algorithms recalled above, selection is realized by the direct implementation of mortality and reproduction. Agents might die due to various reasons, but the most natural one is the lack of energy. Such a situation is usually caused by unfavorable environment conditions to which an agent cannot adopt. Death (without reproduction) means that this genotype is completely eliminated from the

simulation. An agent uses its energy to maintain its vital functions and/or perform its expected actions. Energy is usually retrieved from the environments (nutrients); as this is a shared resource, the agents must compete in order to meet their needs. The amount of nutrients in the environment is one of the factors that can be used to limit the size of a population. In ecology, this is called carrying capacity. For agent-based models, it can be treated as a regulator that controls the size of a population and as criteria for evaluating the fitness function.

EMAS was invented to solve engineering and mathematical problems related to the search for optima in various systems [262, 283, 284]. The software tools that were developed and tuned for these kinds of computations [285, 286] can also be applied in the models of biological and ecological processes.

5.2. Evolving agents in Cellular Automata environment

The issues addressed in the previous section lead to the conclusion that models based on the individual-based modeling approach need a computationally efficient framework that is able to handle accurate (in terms of behavior representation) and large-scale simulations. In the case of models of population, the spatial distribution of individuals in an environment favors the analogous organization of the computations. With their grid, Cellular Automata can be used in these models in a twofold application: as the model of an environment (indispensable in modeling ecology) and the implementation framework (which organizes the processing as well as the agents).

A combination of agent-based modeling and the Cellular Automata paradigm have been applied in models of ecology and the evolution of marine microorganisms called Foraminifera [287, 288]. The models were developed in a project entitled "eVOLUTUS: a simulator of multiscale evolutionary processes tested on Foraminifera." The project's main objective was to design a new algorithmic framework for testing and simulating evolutionary principles and their consequences in a defined environment. It was determined that Foraminifera would be used as the model organism. These single-celled eukaryotes occupy the marine benthic and pelagic zones throughout the world and have an extraordinary fossil record (since the Cambrian Period (540 million years ago). This makes Foraminifera the perfect subject of studies involving computational experiments — the qualitative and quantitative results can be verified by comparing them to an immense amount of fossil data.

Foraminifers live in the salt water of the seas and oceans. There are two groups of species that differ significantly in life strategy: benthic and planktonic foraminifers. Representatives of the benthic species live in a thin layer of the bottom sediment near seashores up to a depth of 100 meters. They actively crawl through the grains

of sand and devour algae, diatoms, bacteria, and other particles of organic matter. Planktonic foraminfers float in the open water, catching other microorganisms by using a network of pseudopodia.

Foraminifera reproduce using two methods: sexual and asexual. The planktonic species uses only the first method. An individual produces thousands of gametes that float in the open water until they met gametes produced by other individuals. After fusing the two gametes, a new individual is created with a double set of chromosomes (diploid individuals). The benthic species has a more complex reproduction strategy. They use sexual and asexual reproduction methods alternately in consecutive generations. Asexual reproduction proceeds simply by dividing a parent body among its children (mitosis). The offspring produced in this way have a single set of chromosomes (haploid individuals). The reproduction processes are accompanied by changes in the chromosomes due to mutation.

The overwhelming majority of the foraminiferal species cover their bodies using hard shells made of calcium carbonate ($\mathrm{CaCO_3}$) or from glued-together grains of sand. Figure 5.1 presents few Foraminiferal shells made of chambers arranged in specific manner. After the death of an individual, its shell falls to the bottom of the sea and becomes a part of the sediment rock. After millions of years, such rock formations can become fragmented; therefore, foraminiferal shells can be found in various places around the Earth (often undamaged).

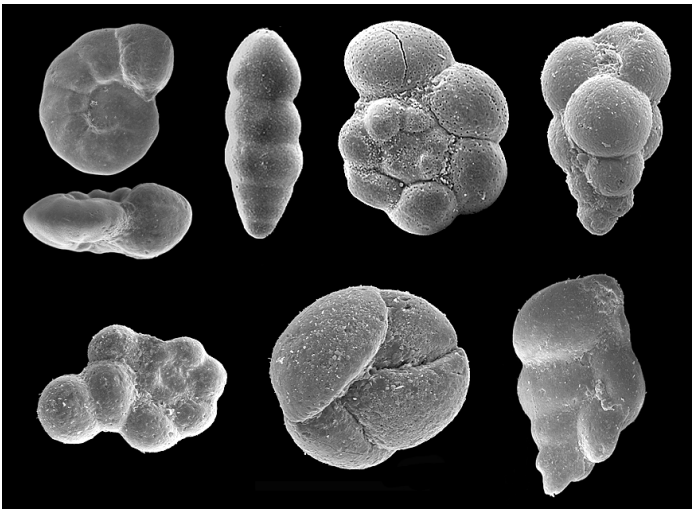

Figure 5.1. Real foraminiferal shell seen under microsocpe — average size of shell varies from 100 micrometers to few centimeters

The shells are usually constructed from a series of connected chambers that are built throughout the whole life of an individual. Each chamber has a hole (called

an aperture) that provides a means of communication between a cytoplasm and the external environment. The arrangement of the chambers in the shell is characteristic for the species.

Foraminifers build their shells in a very peculiar, almost algorithmic way. There are several models that rely on this observation [289, 290, 291, 292]. These models use simple the geometrical transformation of basic shapes (circles, spheres, ellipses) to build a model of a foraminiferal shape. In the case of the most recent model proposed by Topa & Tyszka [292, 293], an important novelty was the introduction of a moving reference system in which each new chamber is added to the shell. This component resulted in the fact that this model is able to reproduce the widest range of shapes, including "impossible forms" [294] (see Fig. 5.2).

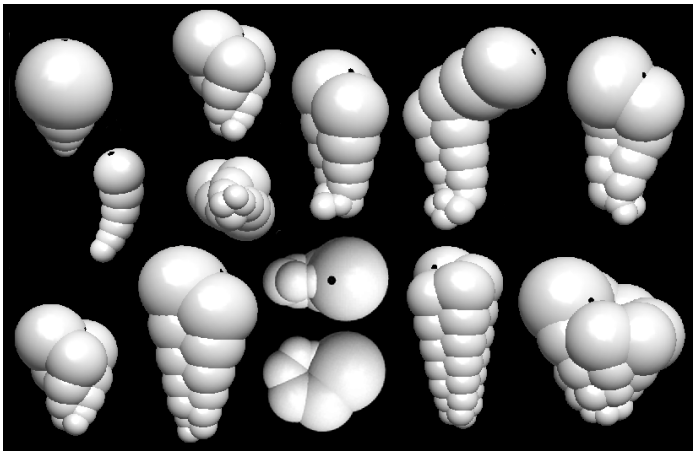

Figure 5.2. Foraminiferal shells generated by the model with moving reference system (Topa & Tyszka [293]). The same model is able to generate high diversity of shells shapes

The eVolutus project was aimed at introducing a complete model of foraminiferal morphology and physiology as well as their behavior. Agents represent the foraminifer individuals that move actively or passively, gather food to maintain their vital functions, grow, and reproduce. They die after reproduction or when their energy resources are exhausted. All of these activities are controlled by parameters whose values have been selected on the basis of current knowledge about the physiology of Foraminifera [295]. In eVolutus, these parameters are treated as genes, and their set (genotype) can be inherited during reproduction as well as modified by typical genetic operators: mutation and crossing over. However, it should be emphasized that none of the "genes" correspond to the real genes existing in foraminiferal DNA. In [296], the term "hypergene" was introduced to differentiate them from real genes. A hypergene should be rather treated as an approximation of many real genes, and

the interactions between them are expressed as some phenotypic traits. The model of the development of foraminiferal shells mentioned above is also a part of eVolutus. The morphological parameters produced by this model (size of shell, volume, weight) allows us to more precisely represent the behavior of the modeled foraminifers.

eVolutus: Population of agents in Cellular Automata environment

The eVolutus simulators incorporate several submodels related to the various aspects of foraminiferal physiology as well as the process occurring in their habitat. These models are discussed with all details in [296, 297]. Figure 5.3 presents the general structure of the eVolutus models.

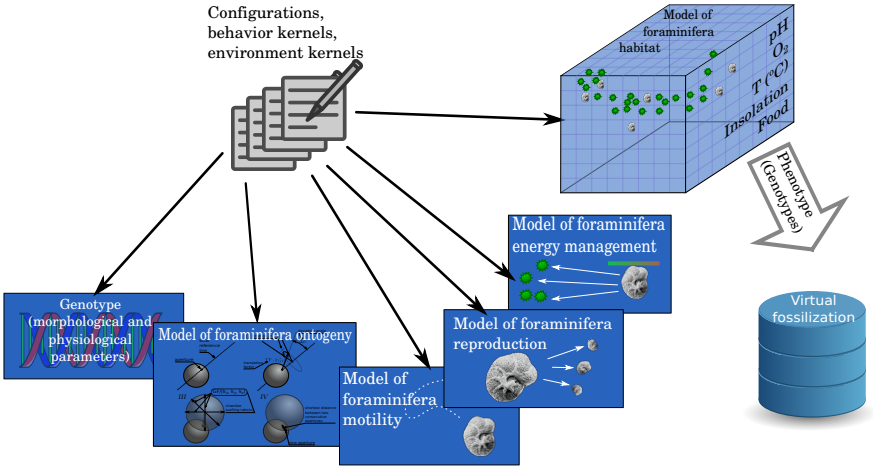

Figure 5.3. Architecture of eVolutus simulator, previously published in [296]

The contributing models describe the genotype, energy management, reproduction, motility, and ontogenesis (growth) of each individual. These agents are located in a virtual habitat that is modeled in a manner similar to the Cellular Automata approach; e.g., the three-dimensional space is partitioned using a regular grid. The size of the whole habitat as well as the size of each cell depends on the type of habitat.

Agents are located in a model of an environment that reflects one of the two types of habitats: a thin layer of a sea floor or an open ocean. In both cases, there are several parameters that describe the physical and chemical properties of these habitats; these hold important meanings for agent behavior. Moreover, the information about how these parameters vary in space and time is crucial from the point of view of the investigated phenomena and processes.

No matter what its type, the habitat is represented using a regular grid. Each cell is described by parameters that reflect some physicochemical properties such

as insolation, oxygen saturation, pH, salinity, etc. However, in a real foraminiferal habitat, food is available as organic particles of various origins and sizes; here, there is only one parameter that describes the food availability. All of the parameters may be modified during the simulation according a defined set of rules. New values can be calculated using their own state as well as the states of all of its neighbors. Once again, this basic principle of Cellular Automata is reasonable here.

It was assumed that each cell of a habitat can be occupied by one or more agents and that their precise positions inside a cell will not be tracked. This principle was made based on the observation that foraminifers usually do not directly interact with each other. They do not attack each other, and their methods of reproduction do not require intimate contact. All agents inside a particular cell use the same food supply. During sexual reproduction, all individuals have the same probability of fusing gametes. It is also assumed that agents from different cells do not influence each other.

This approach has several positive effects on the complexity of a model (as well as its computational efficiency). First of all, there is no need to calculate which agents are direct neighbors. Also, the positions of the agents are stored with a precision determined by the cell size. There is no communication between agents from different cells except at the moment when an agent moves between the cells.

The main parts of these submodels are expressed in the form of rules or formulas that are executed by the agents when the appropriate action must be taken. The rules are encoded in the form of a short function called a kernel by analogy, to usually short functions executed massively parallel by GPUs. Kernels that contain agents' actions are called behavioral ones, while those that modify the properties of a habitat are called environmental. The functionality of kernels provides a high level of configurability because the experimenter can freely modify the code of the kernels according to the hypothesis that is verified. Listing 5.1 demonstrates a sample behavioral kernel that is used to determine the moment of growth.

```
1  function canCreateChamber(envState, foramState, time) {
2    var volumeOfCytoplasm
3      = foramState.energy/foramState.genotype.metabolicEffectiveness[0];
4    var needMoreSpace
5      = volumeOfCytoplasm > 0.95 * foramState.shell.volumeShell;
6    var energyEnough
7      = foramState.energy > energyNeededForGrowth(envState, foramState, time);
8    var growthProbable
9      = rand() > growthProbability(envState, foramState, time);
10   return needMoreSpace && energyEnough && growthProbable;
11 }
```

Listing 5.1. Sample behavioral kernel used by agent to decide whether growth is possible. Agent calculates current amount of cytoplasm and next checks whether it needs more space in itse shell and whether necessary amount of energy is available

In eVolutus, the kernels are written using the Javascript language and executed as Java native code by using the Nashorn library[7]. It should be emphasized that the experimenter does not modify any part of the simulator nor recompile it.

eVolutus introduces a tool of virtual fossilization. Fossilization is a process in which the bodies of living organisms turn into stone after their death. Today, using various techniques, some information about these organisms can be retrieved from the stone. Here, virtual fossilization collects data about all of the foraminifers that were modeled during the simulation. Agents that die as a result of reproduction or due to other reasons are stored in the database. Apart from the genotype, the simulator saves the age of an agent, its shell parameters, its volume of cytoplasm at the moment of death, and so on. Also, information about its parents is remembered and used to reconstruct a phylogenetic tree. In fact, the set of data that is virtually fossilized can be configured by the experimenter depending on the type of the required analysis. All of the collected data can be analyzed using external tools; e.g., to observe how a particular parameter (hypergene) or trait changes in consecutive generations.

Agent definition in eVolutus

In eVolutus, each foraminiferal individual is represented by an agent. The state of an agent is described by several parameters:

- energy level, which is also proportional to the amount of cytoplasm stored by a foraminifer (a detailed explanation of this dependency can be found in [297]),
- age,
- number of chambers and shell volume,
- genotype (or two genotypes in the case of a diploidal individual).

The list above does not include certain auxiliary parameters that are defined to facilitate some computations; e.g. `isHibernated`. Moreover, new parameters might be included by also using kernel functionality.

The genotype of an agent contains a list of parameters (hypergenes) with their values. These parameters are used by submodels of the foraminiferal physiology and morphology, and they influence agents' behavior and life strategy. During reproduction, the genotype is forwarded to the offspring (but with genetic operators applied). The mutation operator simply modifies parameters by a random factor within a defined range. In sexual reproduction, two genotypes are merged (crossing-over operator). According to the defined scenario of an experiment, a different method of merging is used [298].

[7] https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn

The behavior of an agent is governed by rules assigned to various situations:

- gathering food,
- energy dissipation due to maintaining vital functions,
- movement, active or passive,
- reproduction,
- growth: adding new chamber to shell,
- hibernation due to high stress (food shortage),
- death due to reproduction or energy shortage.

Figures 5.4, 5.5, and 5.6 present sample results from various experiments with this model. Since the detailed analysis requires more biological and ecological background here, the charts only illustrate how the model works. Short related comments are in the caption, while a more detailed explanation and discussion can be found in [296].

Figure 5.7 also presents the results of the simulations, but information on the death and newly born individuals is included. Peaks denote blooms of foraminifers when thousands of offspring are produced. Most of these juveniles die shortly after.

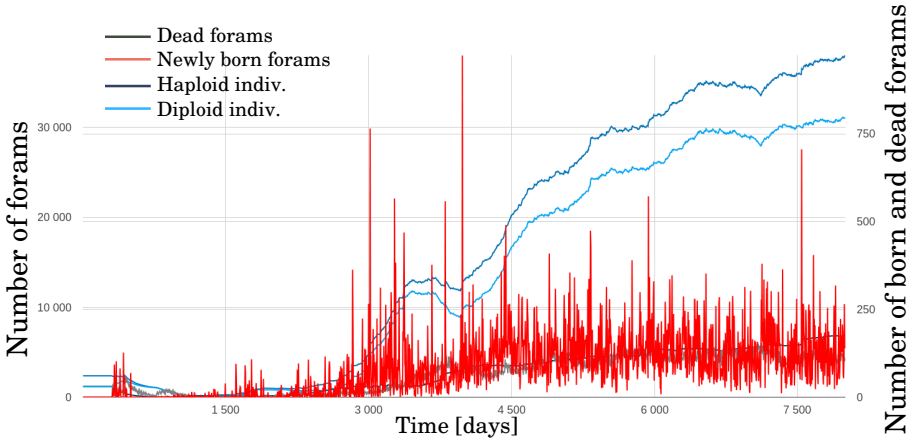

Figure 5.7. Dynamics of population of Foraminifera presented together with information on newly born individuals. In act of reproduction, parents produce thousands of offspring, but most live only briefly.

The eVolutus simulator were verified and validated only partially using some well-known and commonly observed behaviors of Foraminifera [296, 297]. At this moment, there are no empirical data that allow to perform this process completely. They should be collected in experiments optimized for the eVolutus needs.

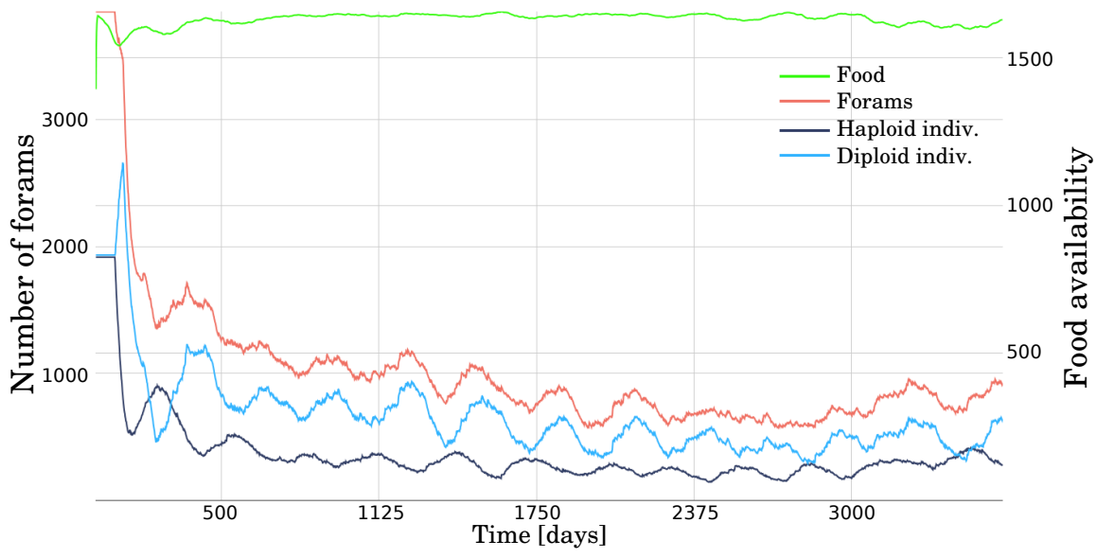

Figure 5.4. Results of experiment with population of benthic foraminifers (two types of reproduction and two types of ploidy). Scenario of experiment assumes that no mutations are applied. Oscillation of population size and food availability is clearly visible. Additionally, chart distinguishes between haploid and diploid individuals. Interlacement of peaks in sizes of their populations is also visible

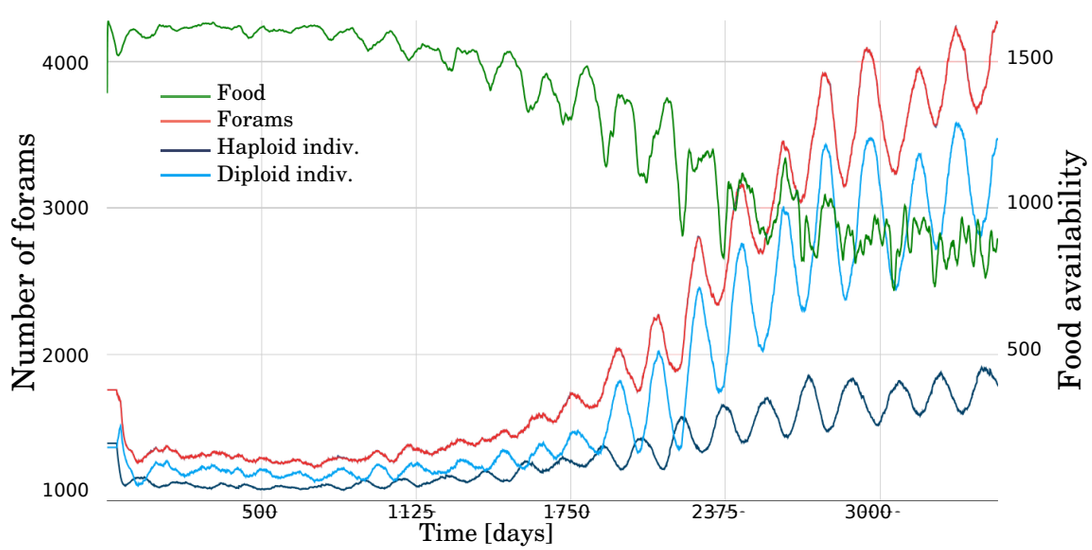

Figure 5.5. Population of evolving foraminifers: mutation rate of 20% in hypergene that is responsible for growth of consecutive chambers in shell. Individuals adopt to existing conditions, and population size can significantly grows during simulation

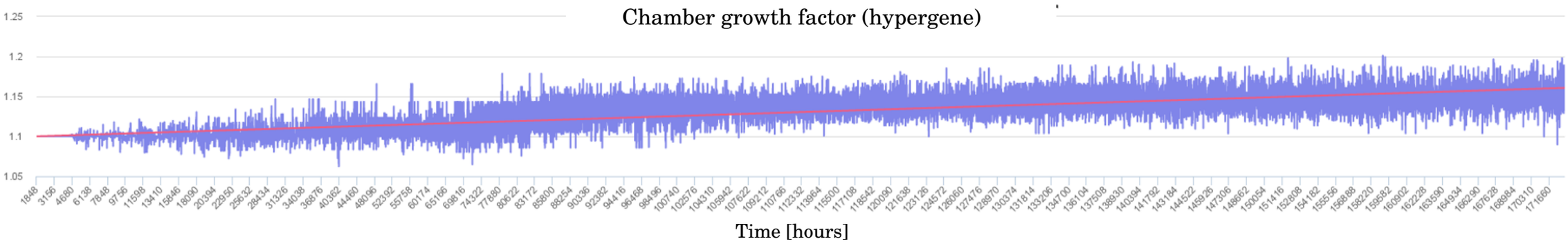

Figure 5.6. Change of hypergene responsible for enlargement of chamber during foraminiferal ontogenesis. Chart is created by collecting data of population of foraminifers that spans over 20 years of simulation. Environmental conditions in this experiment promote larger forms; in consecutive generations, parameters that influence this trait are modified

eVolutus implementation using AgE simulation platform

One of the models designed in the eVolutus project was intended to simulate large populations of foraminifers in order to present the evolution processes as faithfully as possible. In order to accomplish this goal, an agent-based modeling platform that supports distributed/parallel computing was employed to implement the model. AgE (Agent Evolution) [286] is a platform that supports the paradigms of multi-agent system and evolutionary multi-agent system (EMAS), and it provides a low-level interface for managing agents in a distributed environment.

AgE organizes the agents using a tree structure. A set of normal agents are grouped using an aggregate, which is also a special type of agent. The aggregate stores environmental data and the context of processing for stored agents. Top-level aggregates (called workplaces) and all of their children (agents and aggregates) can be distributed over a set of computational nodes. In eVolutus, the tree structure of the AgE agents is mapped onto the Cellular Automata grid that better fits the needs of these phenomena (see Fig. 5.8).

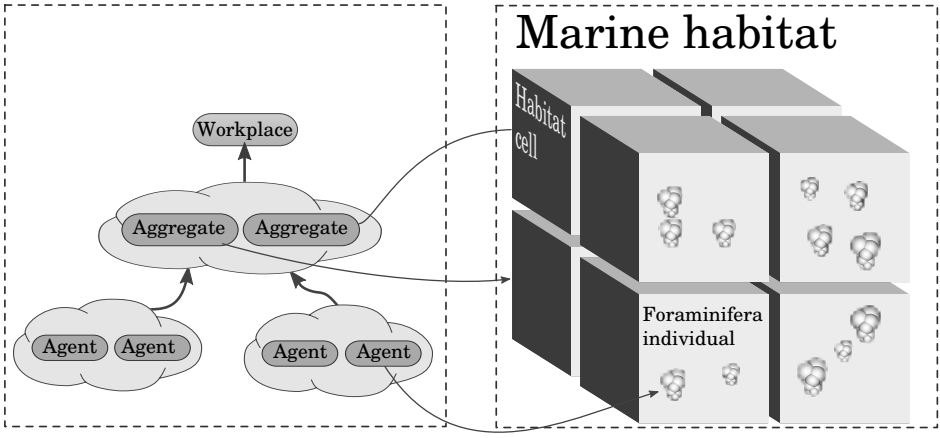

Figure 5.8. Mapping of AgE tree hierarchy onto regular grid of cells in eVolutus habitat [299]

Distributed model of foraminiferal habitat

The AgE platform was intentionally designed to allow for distributed (and parallel) computing. The populations of agents can be distributed among the several workplaces that are located on the different computational nodes. The workplaces are connected using a service bus, which is used to send agent data during migration and exchange information about the environment. It also assures the reliability of a migration — none of the agents may disappear nor appear out of nowhere during

this operation. Figure 5.9 presents performance of distributed implementation of eVolutus — only the AgE built-in functionality was used to scatter computation over the network of workstations.

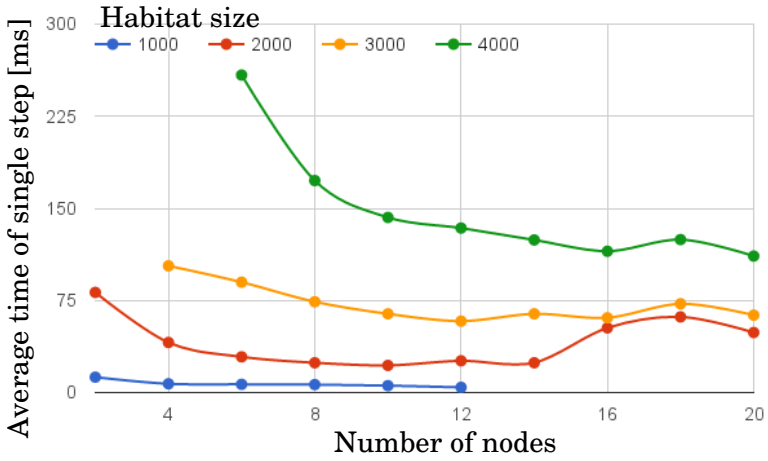

Figure 5.9. Results of performance tests for model of planktonic foraminifers. Habitat was box of open ocean with four different sizes ($x \times x$) and same depth (100 meters). During simulation, population of foraminifers reached level of 10^6 individuals. Simulations were performed on network of workstation with different numbers of nodes

In a model like eVolutus, there are some important issues that influence the efficiency of the distributed implementation. First of all, a global synchronization is necessary — all agents must live in the same moment of time of a simulation. Another challenge is the "weight" of the eVolutus agents. Apart from the set of hypergenes, their states are described by several properties related to the current state of the morphology: age, number of chambers, volume of cytoplasm (equivalent to stored energy), total volume of shell, etc. As a result, migration can be very time consuming – especially for a large group of agents.

Moreover, such a situation is not exceptional in this type of simulation. Many species of living organisms tend to migrate in groups, sometimes due to their social nature, and sometimes due to the distribution of food in the environment. Another effect observed in this type of simulation is an unbalanced distribution of agents inside the habitat. As a result, some regions of the habitat are processed without great effort while others require more time to update all of their agents. The necessity of the global synchronization of a simulation results in the fact that efficiency is decreased by the computational node with the heaviest load.

Potential methods of load balancing might be based on a more fine-grained division of the habitat lattice and assigning a few fragments to each node (see Fig. 5.10). This approach is the simplest and does not require any changes to the code of the eVolutus simulator. The more sophisticated method balances the number of agents processed on each node. This requires additional mapping among the agents representing foraminifers and aggregating agents representing fragments of the habitat. It allows us to logically distribute agents among the computational nodes in isolation from their real spatial distribution.

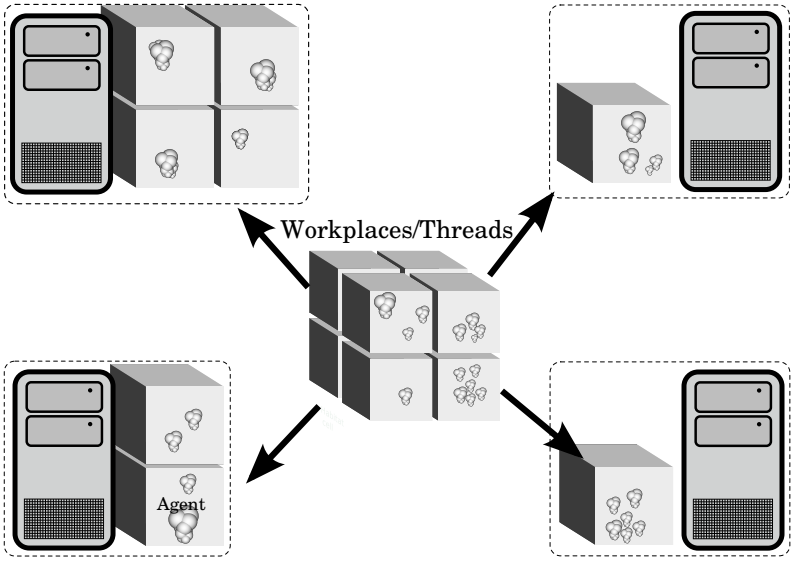

Figure 5.10. Thread-level load balancing. Lattice of habitat is partitioned into set of workplaces. Each computational node might get one or more workplaces depending on agents inside it.

Agent-based simulation accelerated by GPU: the eVolutus case

There are many examples of using GPU computing for accelerating agent-based models. Most of them are models of specific phenomena or processes. Only a few general-purpose agent-based platforms support GPU computing; e.g., FLAME and FLAMEGPU [300]. The main reason for such a situation is the fact that agent-based systems generally do not fit the method of computing that is realized by GPU platforms. The streaming processing used by GPUs means that the same sequence of instructions is synchronously applied to many streams of data. Inversely, agent-based computing prefers the heterogeneity of data and asynchronous updating. However,

potential benefits might be expected when the nature of the modeled phenomena at least partially fit the streaming processing approach. This is the situation in the case of eVolutus.

Simulations using eVolutus involve thousands or more agents that perform a set of actions at each timestep. In the basic implementation, the AgE platform invokes the agents' respective actions sequentially. In the implementation for GPUs, a group of agents must be processed at the same time. This requires some changes in the AgE platform as well as in eVolutus itself. This version of eVolutus is discussed in [299]. Unlike in the original AgE, the version for GPUs handles agents in a bit different way. An aggregate that contains a group of agents (all foraminifers that are currently in a single cell) retrieves the necessary data from each agent, packs it into an array, and sends it to the GPU memory. The GPU executes the functions corresponding to the agents' actions. Finally, the aggregate receives the array with results, unpacking the data as well as the updated agent states. As some actions performed by the agents are related to a shared resource (which is the environment), the processing of agents data cannot be realized in a fully independent manner.

The actions performed by the eVolutus agent might be divided into two classes: internal and external. In the context of this simulator, an internal action means that, as a result of its performing change, only the state of an agent:

- hibernates — agent suspends its activity due to unfavorable conditions and stay in this state until positive signals from the environment bring it back to activity,
- grows — agent enlarges its shell (by building new chamber) in order to store growing amount of cytoplasm,
- consumes energy — agent uses its stored energy to maintain its vital functions.

The external actions of an agent change the internal state of the agent; they also influence the other agents as well as the environment:

- gathering food — agent reduces amount of food in environment and increase its stored cytoplasm,
- migrate — agent actively or passively changes its position in environment (position is quantified by size of habitat cells),
- reproduce — agent produce offspring (sexually or asexually) and die,
- die — agent is removed from the environment.

Internal actions can be easily parallelized and executed with high efficiency due to the lack of competition and synchronization in accessing the data. The external actions influence other components of the simulator. In some situations, it requires synchronization when the shared data is modified by various agents (e.g., gathering food). Actions of migration, reproduction, and death requires synchronization at the end of a simulation step (when the populations in the aggregates are updated).

Most actions in eVolutus consists of two stages. Initially, an agent verifies whether the conditions necessary to continue the actions are fulfilled. Additionally, some initial calculations might be performed. For example, an agent verifies whether the maturity age has been reached and the energy resources are sufficient before reproduction, and it subsequently calculates the number and size of the offspring (or gametes). These operations do not change any external resources (or internal) and can be treated as potential candidates for massive parallel processing. At the second stage of an external action, the shared resources are modified; e.g., new agents are added to the environment.

Agent actions processed by GPU

The internal actions and the first stage of the external actions can be designed to be executed on a GPU. Simply, the code of the behavioral kernels can be written in a form of CUDA or OpenCL kernels instead of in JavaScript. The main issue is how to efficiently pack and send the data that these kernels will use to calculate the desired results. The streaming processors prefer homogeneous data to be stored using regular data structures like arrays, where the schema of the transactions is also regular. In turn, the natural organization of data agent-based systems is a heterogeneous structure (`struct` type in C/C++) that aggregates all of the various properties of an agent in one object.

The most straightforward solution is to use arrays for storing agent data. However, this approach is not effective nor convenient from the point of view of a programmer. The benefits of using object-oriented programming and component methodology in agent-based systems has already been discussed and emphasized in this chapter. Thus, before sending all of the necessary properties and parameters to the GPU memory, they have to be repacked from structures into arrays. When the calculations on the GPU end, the results must also be sent to the host memory and placed in the agents' data structures. Obviously, these operations introduce additional overhead and in order to reduce their influence, computations must be carefully organized.

In [299], the approach introduced above was demonstrated in practice. As a platform for GPU computing, the Nvidia CUDA environment was used. As emphasized above, the main assumption made in this implementation was to keep the general eVolutus architecture. Thus, only the `step()` function of an agent was an area with significant changes. The asynchronous calling of a step function is replaced by its parallel execution for a group of agents. At each single step of the simulation, the aggregates (cells of grid) are activated one by one. Each aggregate invokes a GPU computation for all of the owned agents. The agents within the single aggregate are synchronized, and their data is translated into an array and sent to the global memory.

Next, the simulators invoke the execution of one or more kernels via the JCUDA library (Java bindings for CUDA). The results of the computations are retrieved from the arrays and used to govern the behavior of the agent (see Fig. 5.11).

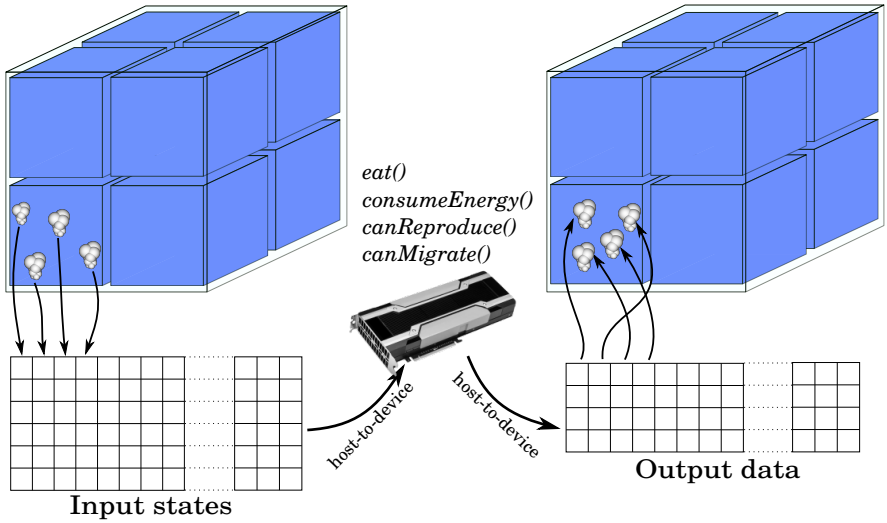

Figure 5.11. Updating agent states using GPU. Necessary data is retrieved from agent objects, packed into array, and sent to device. After computation, agents are updated according to results received from device

The results of the execution of the kernels related to the internal actions are used to directly modify the agents' states. The results obtained from the GPU kernels that implement the first stage of the external actions are used to control the functions that finalize these actions on the host processor(s). When all agents inside all of the aggregates are updated, the environmental kernels are also executed on the GPU in a similar manner. The parameters for all cells are packed into an array, sent to the GPU, and the environment is updated (after the execution of the related GPU kernels).

The proposed approach keeps the advantages of the object-oriented methodology. Any extension to agent functionality can be introduced without great effort. Only a few points in the code of the simulator must be modified:

- agent step() function,
- new GPU kernels related to new functions must be implemented,
- procedure of packing and unpacking agent data must extended if necessary.

Unfortunately, such an approach affects the performance. The basic performance test shows that the proposed solution is not highly efficient. Figure 5.12 shows the computation time for the CPU-only and GPU-accelerated implementations. In these

tests, various initial numbers of agents inside a single habitat cell were defined in the simulation scenarios. The tests showed that the efficiency of the GPU version was unable to outperform the CPU version when the number of agents inside a single container was fewer than 512 agents. For higher numbers of agents, the GPU is faster; however, this is still not a significant speedup.

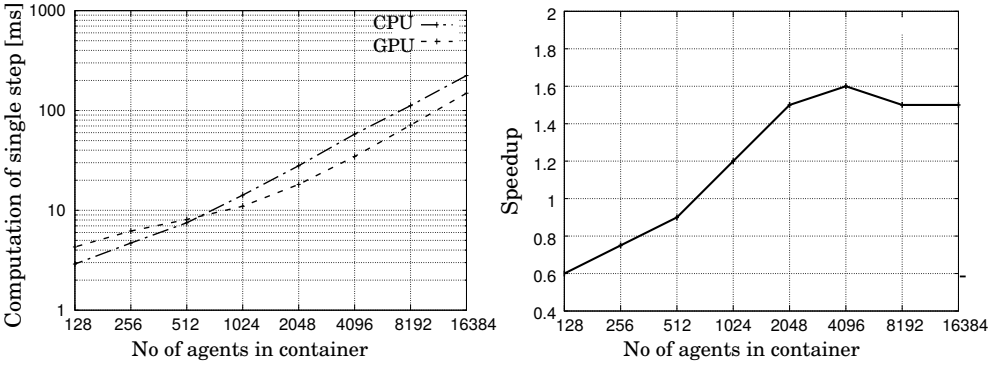

Figure 5.12. Comparison of performance and speedup of agent updating in eVolutus using CPU and GPU implementations

The main reason for such a poor efficiency is still a relatively small fraction of the computation performed using the GPU as well as the necessity of frequently transferring the data from the host memory to the device and back. Unfortunately, this overhead cannot be avoided if this architecture of the simulator is preserved.

5.3. On efficiency of Cellular Automata with Agents and Agents with Cellular Automata

eVolutus mainly uses the agent-based modeling paradigm. The Cellular Automata paradigm is used in this model to spatially organize the computation. The regular lattice is used to determine the position of the agents inside the space of the environment and the possible directions of the migrations. Also, the environment and its evolution is represented using this approach. However, the agents have their extensive sets of properties and complex behaviors. This combination can be named Agents over (with/in) Cellular Automata, as the latter paradigm can be treated as an environment for the first methodology.

Conversely, the model of pedestrian dynamics whose GPU-based implementation was discussed in Chapter 3 uses a methodology that can be partially described as Cellular Automata with Agents. In that model, the Cellular Automata methodology also represents the environment and organizes the computation of the

model. Moreover, the pedestrians can also be represented as the state of the automata ("free"/"occupied by a person"); however, when the complexity of their behavior grows, it is much more convenient to treat them as simple "agents." Additionally, most models of pedestrian dynamics are designed to simulate these phenomena under certain circumstances; e.g., evacuation. Their behavior is, in fact, limited only to the typical activity undertaken during such events: moving towards evacuation points and avoiding obstacles. Thus, the Cellular Automata approach prevails in this combination; the agent approach is a kind of an extension here. As it was demonstrated, such an approach benefits from the model being implemented with massive parallelism (GPU).

The pure agent-based models or models in which the ABM paradigm dominates cannot compete with Cellular Automata models when it comes to efficiency. The main reason for this is the high level of complexity and autonomy of the agents. As emphasized at the beginning of this chapter, the agent is autonomous, which means that it is able to perform its actions independently from other agents. As a result, the models that use ABM are full of heterogenous individuals that perform various actions at the same time. This is a different situation than in the case of CA, where the individuals are tightly linked and synchronized in their actions.

In the eVolutus models, the use of the agent-based methodology plays an essential role due to the complexity and diversity of the modeled behavior. In [301], the Cellular Automata approach was applied to simulate a foraminiferal population, but the model was deprived of any components related to genetics and included only a very simple model of the reproduction and energetics. The model was able to present typical predator-prey dynamics (foraminifers and algae). The implementation was made using the Python language, and its purpose was to test some assumption rather than to gain higher efficiency in the simulations. A very similar model was implemented using the Scala language in order to perform high-performance simulations [302]. This time, the implementation was able to simulate huge populations, but the model was still unable to realistically model some issues of foraminiferal physiology (especially reproduction).

6. Summary

Computer simulations are now an indispensable method for scientific investigations. Efficient and flexible modeling methods are necessary in order to use computer simulations in various scientific disciplines. The need for such methods is driven by the development in those disciplines where new problems and questions need attention. Intensive development of computer hardware and software is another factor that forces progress in modeling methods. Problems which in the past were solvable with supercomputers can now be processed using desktop machines equipped with a graphic card. At the same time it became possible to solve problems of a much larger spatio-temporal scale and with a much higher degree of complexity. Various modeling methods together with implementation strategies are developed within the scope of computer science in order to meet these needs.

In the area of computer modeling, particular attention is paid to Complex Systems. Since their global behavior emerges as a result of interactions between many components, local behavior of Complex Systems can only be predicted using step-by-step simulations. Usually, we are aware of how the individual parts work, but this knowledge rarely explains the global behavior of these phenomena. Research on Complex Systems is essential as the property of complexity is present within most natural phenomena. In computer science, there are many modeling methods aimed at simulations of complex systems and most of them rely on directly representing the components and the interactions of these systems. These methods are intensively developed in order to extend their application to a broader range of systems and to achieve the highest possible efficiency in modeling.

Cellular Automata are one of the most popular modeling paradigms, especially when Complex Systems are considered. Today it is a method with hundreds applications across various scientific disciplines. Chapter 1 contains an extensive survey of various applications of Cellular Automata, emphasizing modifications and extensions that authors introduce in order to better represent the modeled phenomena. This survey shows that Cellular Automata is a very flexible paradigm and in practice it might be treated rather like a computational framework which is characterized by

discretization of time and space and the existence of many parts that interact with each other according to defined rules. These features facilitate computer implementations of Cellular Automata models including parallel ones.

This monograph provides examples of Cellular Automata flexibility and efficiency. This method in its original sense, where a system is simply represented by a regular grid, is not effective when the modeled system is composed of processes occurring in different spatio-temporal scales. The extension proposed by the Author is able to include these processes into one consistent model and fits into the general philosophy of Cellular Automata. Thus the well-known implementation schemes typical for this paradigm can be easily applied. Moreover, they can be additionally optimized for the newest computing platforms like graphic processing units (GPU), which in turn opens up new opportunities for using valuable applications.

Efficient and flexible implementation of the Cellular Automata model encourage the use of this paradigm not only for simulation purposes, but also as a kind of computational framework. It can be useful when space and time in modeled phenomena is important. A regular grid facilitates management of simulated entities, organizes their processing and defines the scheme of their neighborhood. Discrete time helps to define the dynamics of the modeled processes. The well-known and tested computational architectures developed for pure Cellular Automata models can be applied in these model with success.

The solutions proposed by the Author are illustrated by practical application for several natural phenomena. In all cases their usefulness was demonstrated and the models brought new contribution into investigations related to these phenomena.

To sum up, the original results presented in this monograph are as follows:

1. Definition of a new extension to the Cellular Automata paradigm which allows one to model multiscale phenomena. In particular, the extension is able to represent systems composed of transportation networks located inside the consuming and producing environment. The processes related to the network are fast, global and irregular, while the environment develops much slower and its evolution depends on local conditions like nutrient saturation. The proposed extension is able to harmoniously unite both the processes in one model.

2. Introduction of a new algorithm for Cellular Automata models optimized for stream processing. Widely available graphic processing units (GPU) can be used to conduct computations other than rendering, but in order to benefit from their massively parallel architecture, new optimized algorithms need to be proposed. This approach was demonstrated using two Cellular Automata models and in both cases the increase in speed as compared to implementation for ordinary processors is very high (100 times or more).

3. Application of Cellular Automata as a computational framework for models using other paradigms like agent-based systems, multiagent systems and evolutionary multiagent systems (EMAS). In such the models, Cellular Automata represent the agents' habitat and its evolution is governed by internal and external factors. Additionally, the Cellular Automata and particularly their grid is used to organize processing of the population of agents. In this way, the well-known schemes for parallel processing can be applied directly in such the models. This approach was illustrated by the model of population dynamics and evolution of small living creatures. Implementations for a distributed platform and for a GPU allow one to conduct simulations including 10^6 and more agents across thousands of generations.

The results presented in this monograph are believed to be a starting point for application the Cellular Automata and proposed extensiones for modeling other processes or phenomena belonging to the class of Complex Systems, which have not been considered here.

Bibliography

[1] Wolfram, S.: *A new kind of science*. Wolfram Media Champaign, 2002.

[2] Wolfram, S.: *Undecidability and intractability in theoretical physics*. Physical Review Letters, vol. 54, no. 8, 1985, pp. 735–738.

[3] Neumann, J. von: *Theory of Self-Reproducing Automata*. Ed. by A. W. Burks, University of Illinois Press, 1966.

[4] Gardner, M.: *Mathematical Games: On Cellular Automata, self-reproduction, the Garden of Eden and the Game life*. Scientific American, vol. 224, no. 2, 1971, pp. 112–117.

[5] Rendell, P.: *A Universal Turing Machine in Conway's Game of Life*. In: *International Conference on High Performance Computing and Simulation HPCS 2011*. 2011, pp. 764–772.

[6] Weisstein, E. W.: *Elementary Cellular Automaton. From MathWorld–A Wolfram Web Resource.* URL: http : / / mathworld . wolfram . com / ElementaryCellularAutomaton.html (Retrieved 12/01/2018).

[7] Sayama, H.: *Introduction to the modeling and analysis of complex systems*. New York, USA, Open SUNY Textbooks, 2015.

[8] Kondo, S. and Miura, T.: *Reaction-diffusion model as a framework for understanding biological pattern formation*. Science, vol. 329, no. 5999, 2010, pp. 1616–1620.

[9] Bar-Yam, Y.: *Dynamics of complex systems*. Westview Press, 1997.

[10] Kaneko, K.: *Spatiotemporal intermittency in coupled map lattices*. Progress of Theoretical Physics, vol. 74, no. 5, 1985, pp. 1033–1044.

[11] Axelrod, R.: *Advancing the Art of Simulation in the Social Sciences*. In: *Simulating Social Phenomena*. Ed. by R. Conte, R. Hegselmann, and P. Terna, Springer Berlin Heidelberg, 1997, pp. 21–40.

[12] Epstein, J. M.: *Agent-based computational models and generative social science*. Complexity, vol. 4, no. 5, 1999, pp. 41–60.

[13] Rogers, H.: *Theory of recursive functions and effective computability*. United States, MIT Press Cambridge, 1987.

[14] Blum, M.: *A Machine-Independent Theory of the Complexity of Recursive Functions*. Journal of the ACM, vol. 14, no. 2, 1967, pp. 322–336.

[15] Wolfram, S.: *Statistical mechanics of cellular automata*. Reviews of Modern Physics, vol. 55, no. 3, 1983, pp. 601–644.

[16] Gerhardt, M. and Schuster, H.: *A cellular automaton describing the formation of spatially ordered structures in chemical systems*. Physica D: Nonlinear Phenomena, vol. 36, no. 3, 1989, pp. 209–221.

[17] Dilao, R. and Sainhas, J.: *Validation and calibration of models for reaction–diffusion systems*. International Journal of Bifurcation and Chaos, vol. 8, no. 6, 1998, pp. 1163–1182.

[18] Torrens, P. M.: *Calibrating and Validating Cellular Automata Models of Urbanization*. In: *Urban Remote Sensing*. Wiley-Blackwell, 2011. Chap. 23, pp. 335–345.

[19] Jin, C. J. et al.: *Calibration and validation of cellular automaton traffic flow model with empirical and experimental data*. IET Intelligent Transport Systems, vol. 12, no. 5, 2018, pp. 359–365.

[20] Frisch, U., Hasslacher, B., and Pomeau, Y.: *Lattice-gas automata for the Navier-Stokes equation*. Physical Review Letters, vol. 56, no. 14, 1986, pp. 1505–1508.

[21] Toffoli, T.: *Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics*. Physica D: Nonlinear Phenomena, vol. 10, no. 1, 1984, pp. 117–127.

[22] Chopard, B. and Droz, M.: *Cellular automata model for the diffusion equation*. Journal of Statistical Physics, vol. 64, no. 3-4, 1991, pp. 859–892.

[23] Yang, X.-S. and Young, Y.: *Cellular automata, PDEs, and pattern formation*. In: *Handbook of Bioinspired Algorithms and Applications*. Ed. by S. Olariu and A. Y. Zomaya, Chapman and Hall/CRC, 2010.

[24] Crisci, G. et al.: *Lava Flow Simulation Bv A Discrete Cellular Model: First Implementation*. International Journal of Modelling and Simulation, vol. 6, no. 4, 1986, pp. 137–140.

[25] Di Gregorio, S. and Serra, R.: *An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata*. Future Generation Computer Systems, vol. 16, no. 2-3, 1999, pp. 259–271.

[26] D'Ambrosio, D. et al.: *First simulations of the Sarno debris flows through cellular automata modelling*. Geomorphology, vol. 54, no. 1-2, 2003, pp. 91–117.

[27] Lupiano, V. et al.: *A modelling approach with macroscopic cellular automata for hazard zonation of debris flows and lahars by computer simulations*. International Journal of Geology, vol. 9, 2015, pp. 35–46.

[28] Kaneko, K.: *Spatiotemporal chaos in one-and two-dimensional coupled map lattices*. Physica D: Nonlinear Phenomena, vol. 37, no. 1-3, 1989, pp. 60–82.

[29] Cintula, P., Fermüller, C. G., and Noguera, C.: *Fuzzy Logic. The Stanford Encyclopedia of Philosophy*. Ed. by E. N. Zalta, 2017. URL: https://plato.stanford.edu (Retrieved 12/01/2018).

[30] Gong, Y.-G. and Liu, L.: *A fuzzy cellular automaton model based on NaSchH model*. In: *2nd International Conference on Signal Processing Systems ICSPS 2010, Dalian, China*. Vol. 2. 2010, pp. 518–522.

[31] Płaczek, B.: *A Traffic Model Based on Fuzzy Cellular Automata*. Journal of Cellular Automata, vol. 8, 2013, pp. 261–282.

[32] Płaczek, B.: *Fuzzy Cellular Model for On-Line Traffic Simulation*. In: *Parallel Processing and Applied Mathematics : 8th international conference, PPAM 2009: Wrocław, Poland, September 13-16, 2009: revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 6068, Berlin, Heidelberg, Springer, 2010, pp. 553–560.

[33] Al-Ahmadi, K. et al.: *Calibration of a fuzzy cellular automata model of urban dynamics in Saudi Arabia*. Ecological Complexity, vol. 6, no. 2, 2009, pp. 80–101.

[34] Liu, Y. and Phinn, S. R.: *Modelling urban development with cellular automata incorporating fuzzy-set approaches*. Computers, Environment and Urban Systems, vol. 27, no. 6, 2003, pp. 637–658.

[35] Sahin, U., Uguz, S., and Sahin, F.: *Salt and pepper noise filtering with fuzzy-cellular automata*. Computers & Electrical Engineering, vol. 40, no. 1, 2014, pp. 59–69.

[36] Sadeghi, S., Rezvanian, A., and Kamrani, E.: *An efficient method for impulse noise reduction from images using fuzzy cellular automata*. AEU-International Journal of Electronics and Communications, vol. 66, no. 9, 2012, pp. 772–779.

[37] Uguz, S., Sahin, U., and Sahin, F.: *Edge detection with fuzzy cellular automata transition function optimized by PSO*. Computers & Electrical Engineering, vol. 43, 2015, pp. 180–192.

[38] Hardy, J., Pomeau, Y., and De Pazzis, O.: *Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions*. Journal of Mathematical Physics, vol. 14, no. 12, 1973, pp. 1746–1759.

[39] McNamara, G. R. and Zanetti, G.: *Use of the Boltzmann Equation to Simulate Lattice-Gas Automata*. Physical Review Letter, vol. 61, no. 20, 1988, pp. 2332–2335.

[40] Higuera, F. J. and Jiménez, J.: *Boltzmann Approach to Lattice Gas Simulations*. Europhysics Letters, vol. 9, no. 7, 1989, pp. 663–668.

[41] Wolf-Gladrow, D. A.: *Lattice-gas cellular automata and lattice Boltzmann models: an introduction*. Springer Berlin Heidelberg, 2004.

[42] Chen, H., Chen, S., and Matthaeus, W. H.: *Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method*. Physical Review A, vol. 45, no. 8, 1992, pp. 5339–5342.

[43] Chen, S. and Doolen, G. D.: *Lattice Boltzmann method for fluid flows*. Annual review of fluid mechanics, vol. 30, no. 1, 1998, pp. 329–364.

[44] Liu, Y.: *A lattice Boltzmann model for blood flows*. Applied Mathematical Modelling, vol. 36, no. 7, 2012, pp. 2890–2899.

[45] Ouared, R. and Chopard, B.: *Lattice Boltzmann simulations of blood flow: non-Newtonian rheology and clotting processes*. Journal of Statistical Physics, vol. 121, no. 1-2, 2005, pp. 209–221.

[46] Gao, J. et al.: *Reactive transport in porous media for CO2 sequestration: Pore scale modeling using the lattice Boltzmann method*. Computers & Geosciences, vol. 98, 2017, pp. 9–20.

[47] Chen, L. et al.: *Nanoscale simulation of shale transport properties using the lattice Boltzmann method: permeability and diffusivity*. Scientific Reports, vol. 5, 2015, pp. 1–8.

[48] He, X., Chen, S., and Zhang, R.: *A lattice Boltzmann scheme for incompressible multiphase flow and its application in simulation of Rayleigh–Taylor instability*. Journal of Computational Physics, vol. 152, no. 2, 1999, pp. 642–663.

[49] Hou, S. et al.: *Evaluation of two lattice Boltzmann models for multiphase flows*. Journal of Computational Physics, vol. 138, no. 2, 1997, pp. 695–713.

[50] Thurey, N. and Rude, U.: *Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids*. Computing and Visualization in Science, vol. 12, no. 5, 2009, pp. 247–263.

[51] Kryza, T. and Dzwinel, W.: *Coupling Lattice Boltzmann Gas and Level Set Method for Simulating Free Surface Flow in GPU/CUDA Environment*. In: *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015, revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 8385, Springer Berlin Heidelberg, 2014, pp. 731–740.

[52] Baxter, G. and Behringer, R.: *Cellular automata models for the flow of granular materials*. Physica D: Nonlinear Phenomena, vol. 51, no. 1, 1991, pp. 465–471.

[53] Yanagita, T.: *Three-Dimensional Cellular Automaton Model of Segregation of Granular Materials in a Rotating Cylinder*. Physical Review Letters, vol. 82, 17 1999, pp. 3488–3491.

[54] Katsura, N. et al.: *Development of new simulation method for flow behavior of granular materials in a blast furnace using cellular automaton*. ISIJ International, vol. 45, no. 10, 2005, pp. 1396–1405.

[55] Helbing, D. and Johansson, A.: *Pedestrian, crowd and evacuation dynamics*. In: *Encyclopedia of complexity and systems science*. Ed. by R. A. Meyers, Springer Berlin Heidelberg, 2009, pp. 6476–6495.

[56] Kirchner, A. and Schadschneider, A.: *Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics*. Physica A: Statistical Mechanics and Its Applications, vol. 312, no. 1-2, 2002, pp. 260–276.

[57] Yamamoto, K., Kokubo, S., and Nishinari, K.: *Simulation for pedestrian dynamics by real-coded cellular automata (RCA)*. Physica A: Statistical Mechanics and Its Applications, vol. 379, no. 2, 2007, pp. 654–660.

[58] Burstedde, C. et al.: *Simulation of pedestrian dynamics using a two-dimensional cellular automaton*. Physica A: Statistical Mechanics and Its Applications, vol. 295, no. 3-4, 2001, pp. 507–525.

[59] Varas, A. et al.: *Cellular automaton model for evacuation process with obstacles*. Physica A: Statistical Mechanics and Its Applications, vol. 382, no. 2, 2007, pp. 631–642.

[60] Kłusek, A., Topa, P., and Wąs, J.: *Towards Effective GPU Implementation of Social Distances Model for Mass Evacuation*. In: *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, part 2*. Ed. by Roman Wyrzykowski, Lecture Notes in Computer Science, no. 9574, Springer Berlin Heidelberg, 2016, pp. 550–559.

[61] Kłusek, A. et al.: *An implementation of the Social Distances Model using multi-GPU systems*. The International Journal of High Performance Computing Applications, vol. 32, no. 4, 2018, pp. 482–495.

[62] Dijkstra, J., Jessurun, J., and Timmermans, H. J.: *A multi-agent cellular automata model of pedestrian movement*. In: *International conference, Pedestrian and evacuation dynamics 2001, Duisburg, Germany*. Ed. by M. Schreckenberg and S. Sharma, Springer Berlin Heidelberg, 2001, pp. 173–181.

[63] Klügl, F. and Rindsfüser, G.: *Large-Scale Agent-Based Pedestrian Simulation*. In: *MATES'07 Proceedings of the 5th German conference on Multiagent System Technologies, Leipzig, Germany — September 24 - 26, 2007*. Ed. by P. Petta et al., Lecture Notes in Computer Science, no. 4687, Springer Berlin Heidelberg, 2007, pp. 145–156.

[64] Nagel, K. and Schreckenberg, M.: *A cellular automaton model for freeway traffic*. Journal de Physique I, vol. 2, no. 12, 1992, pp. 2221–2229.

[65] Nagel, K. et al.: *Two-lane traffic rules for cellular automata: A systematic approach*. Physical Review E, vol. 58, no. 2, 1998, pp. 1425.

[66] Benjamin, S. C., Johnson, N. F., and Hui, P.: *Cellular automata models of traffic flow along a highway containing a junction*. Journal of Physics A: Mathematical and General, vol. 29, no. 12, 1996, pp. 3119.

[67] Chopard, B., Luthi, P. O., and Queloz, P.-A.: *Cellular automata model of car traffic in a two-dimensional street network*. Journal of Physics A: Mathematical and General, vol. 29, no. 10, 1996, pp. 2325.

[68] Simon, P. and Gutowitz, H. A.: *Cellular automaton model for bidirectional traffic*. Physical Review E, vol. 57, no. 2, 1998, pp. 2441.

[69] Schadschneider, A. and Schreckenberg, M.: *Traffic flow models with 'slow-to-start' rules*. Annalen der Physik, vol. 509, no. 7, 1997, pp. 541–551.

[70] Chung, K., Hui, P., and Gu, G.: *Two-dimensional traffic flow problems with faulty traffic lights*. Physical Review E, vol. 51, no. 1, 1995, pp. 772.

[71] Wagner, P.: *Traffic simulations using cellular automata: Comparison with reality*. In: *Workshop on Traffic and Granular Flow: HLRZ, Forschungszentrum Julich, Germany October 9-11*. World Scientific, 1995, pp. 199–204.

[72] Brockfeld, E. et al.: *Optimizing traffic lights in a cellular automaton model for city traffic*. Physical Review E, vol. 64, no. 5, 2001, pp. 056132 (1-12).

[73] Esser, J. and Schreckenberg, M.: *Microscopic simulation of urban traffic based on cellular automata*. International Journal of Modern Physics C, vol. 8, no. 5, 1997, pp. 1025–1036.

[74] Wang, R. and Ruskin, H. J.: *Modeling traffic flow at a single-lane urban roundabout*. Computer Physics Communications, vol. 147, no. 1-2, 2002, pp. 570–576.

[75] Wang, R. and Liu, M.: *A realistic cellular automata model to simulate traffic flow at urban roundabouts*. In: *5th International Conference on Computational Science ICCS 2005, Atlanta, GA, USA, May 22-25*. Lecture Notes in Computer Science, no. 3515, Springer Berlin Heidelberg, 2005, pp. 420–427.

[76] Batty, M.: *Urban modelling*. Cambridge University Press Cambridge, 1976.

[77] Couclelis, H.: *From cellular automata to urban models: new principles for model development and implementation*. Environment and planning B: Planning and design, vol. 24, no. 2, 1997, pp. 165–174.

[78] Santé, I. et al.: *Cellular automata models for the simulation of real-world urban processes: A review and analysis*. Landscape and Urban Planning, vol. 96, no. 2, 2010, pp. 108–122.

[79] Wu, F.: *Simulating temporal fluctuations of real estate development in a cellular automata city*. Transactions in GIS, vol. 7, no. 2, 2003, pp. 193–210.

[80] Batty, M. and Xie, Y.: *Possible urban automata*. Environment and Planning B: Planning and Design, vol. 24, no. 2, 1997, pp. 175–192.

[81] Batty, M., Xie, Y., and Sun, Z.: *Modeling urban dynamics through GIS-based cellular automata*. Computers, environment and urban systems, vol. 23, no. 3, 1999, pp. 205–233.

[82] White, R., Engelen, G., and Uljee, I.: *The use of constrained cellular automata for high-resolution modelling of urban land-use dynamics*. Environment and Planning B: Planning and Design, vol. 24, no. 3, 1997, pp. 323–343.

[83] Shi, W. and Pang, M. Y. C.: *Development of Voronoi-based cellular automata-an integrated dynamic model for Geographical Information Systems*. International Journal of Geographical Information Science, vol. 14, no. 5, 2000, pp. 455–474.

[84] O'Sullivan, D.: *Exploring spatial process dynamics using irregular cellular automaton models*. Geographical Analysis, vol. 33, no. 1, 2001, pp. 1–18.

[85] Asnaashari, M. and Meybodi, M. R.: *Irregular cellular learning automata and its application to clustering in sensor networks*. In: *Proceedings of 15th Conference on Electrical Engineering (15th ICEE), Communication, Telecommunication Research Center, Tehran, Iran*. 2007, pp. 14–28.

[86] Janssens, K. G.: *Random grid, three-dimensional, space-time coupled cellular automata for the simulation of recrystallization and grain growth*. Modelling and Simulation in Materials Science and Engineering, vol. 11, no. 2, 2003, pp. 157–171.

[87] Topa, P.: *Towards a Two-Scale Cellular Automata Model of Tumour-Induced Angiogenesis*. In: *Cellular automata : 7th international conference on Cellular Automata for research and industry, ACRI 2006: Perpignan, France, September 20–23, 2006*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 4173, Springer Berlin Heidelberg, 2006, pp. 337–346.

[88] Topa, P.: *The graph of cellular automata applied for modelling tumour induced angiogenesis*. In: *Parallel Processing and Applied Mathematics: 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 8385, Springer Berlin Heidelberg, 2014, pp. 711–720.

[89] Topa, P.: *Dynamically reorganising vascular networks modelled using cellular automata approach*. In: *Cellular automata: 8th international conference on Cellular automata for research and industry, ACRI 2008 : Yokohama, Japan, September 23–26, 2008*. Ed. by H. Umeo et al., Lecture Notes in Computer Science, no. 5191, Springer Berlin Heidelberg, 2008, pp. 494–499.

[90] Flache, A. and Hegselmann, R.: *Do irregular grids make a difference? Relaxing the spatial regularity assumption in cellular models of social dynamics*. Journal of Artificial Societies and Social Simulation, vol. 4, no. 4, 2001, pp. 1–6.

[91] Baetens, J. M. and De Baets, B.: *Phenomenological study of irregular cellular automata based on Lyapunov exponents and Jacobians*. Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 20, no. 3, 2010, pp. 033112(1–15).

[92] Fates, N.: *A guided tour of asynchronous cellular automata*. In: *International Workshop on Cellular Automata and Discrete Complex Systems AUTOMATA 2013*. Ed. by J. Kari, M. Kutrib, and A. Malcher, Lecture Notes in Computer Science, no. 8185, Springer Berlin Heidelberg. 2013, pp. 15–30.

[93] Bandman, O.: *Parallel Composition of Asynchronous Cellular Automata Simulating Reaction Diffusion Processes*. In: *9th International Conference on Cellular Automata for Research and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 6350, Springer Berlin Heidelberg, 2010, pp. 395–398.

[94] Ruxton, G. D. and Saravia, L. A.: *The need for biological realism in the updating of cellular automata models*. Ecological Modelling, vol. 107, no. 2, 1998, pp. 105–112.

[95] Bersini, H. and Detours, V.: *Asynchrony induces stability in cellular automata based models*. In: *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1994, pp. 382–387.

[96] Blok, H. J. and Bergersen, B.: *Synchronous versus asynchronous updating in the "Game of Life"*. Physical Review E, vol. 59, no. 4, 1999, pp. 3876.

[97] Fates, N.: *Asynchronism Induces Second-Order Phase Transitions in Elementary Cellular Automata*. Journal of Cellular Automata, vol. 4, no. 1, 2009, pp. 21–38.

[98] Lee, J. et al.: *Asynchronous Game of Life*. Physica D: Nonlinear Phenomena, vol. 194, no. 3, 2004, pp. 369–384.

[99] Ninagawa, S., Yoneda, M., and Hirose, S.: *1/f fluctuation in the "Game of Life"*. Physica D: Nonlinear Phenomena, vol. 118, no. 1, 1998, pp. 49–52.

[100] Silva, F. and Correia, L.: *An experimental study of noise and asynchrony in elementary cellular automata with sampling compensation*. Natural Computing, vol. 12, no. 4, 2013, pp. 573–588.

[101] Takada, Y. et al.: *Construction universality in purely asynchronous cellular automata*. Journal of Computer and System Sciences, vol. 72, no. 8, 2006, pp. 1368–1385.

[102] Baetens, J. M., Van der Weeën, P., and De Baets, B.: *Effect of asynchronous updating on the stability of cellular automata*. Chaos, Solitons & Fractals, vol. 45, no. 4, 2012, pp. 383–394.

[103] Agapie, A., Höns, R., and Mühlenbein, H.: *Markov chain analysis for one-dimensional asynchronous cellular automata*. Methodology and Computing in Applied Probability, vol. 6, no. 2, 2004, pp. 181–201.

[104] Fates, N. A. and Morvan, M.: *An experimental study of robustness to asynchronism for elementary cellular automata*. Complex Systems, vol. 16, no. 1, 2004.

[105] Manzoni, L.: *Some formal properties of asynchronous cellular automata*. In: *9th International Conference on Cellular Automata for Research and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 6350, Springer Berlin Heidelberg, 2010, pp. 419–428.

[106] Manzoni, L.: *Asynchronous cellular automata and dynamical properties*. Natural Computing, vol. 11, no. 2, 2012, pp. 269–276.

[107] Bandini, S., Bonomi, A., and Vizzari, G.: *What do we mean by asynchronous CA? A reflection on types and effects of asynchronicity*. In: *9th International Conference on Cellular Automata for Research and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 6350, Springer Berlin Heidelberg, 2010, pp. 385–394.

[108] Bandini, S., Bonomi, A., and Vizzari, G.: *An analysis of different types and effects of asynchronicity in cellular automata update schemes*. Natural Computing, vol. 11, no. 2, 2012, pp. 277–287.

[109] Schönfisch, B. and Roos, A. de: *Synchronous and asynchronous updating in cellular automata*. BioSystems, vol. 51, no. 3, 1999, pp. 123–143.

[110] Li, W., Zomaya, A. Y., and Al-Jumaily, A.: *Cellular automata based models of wireless sensor networks*. In: *Proceedings of the 7th ACM International Symposium on Mobility Management and Wireless Access, MOBIWAC'09, Tenerife, Canary Islands, Spain — October 26 - 27, 2009*. ACM New York, USA, 2009, pp. 1–6.

[111] Saghiri, A. M. and Meybodi, M. R.: *A closed asynchronous dynamic model of cellular learning automata and its application to peer-to-peer networks*. Genetic Programming and Evolvable Machines, vol. 18, no. 3, 2017, pp. 313–349.

[112] Saghiri, A. M. and Meybodi, M. R.: *An adaptive super-peer selection algorithm considering peers capacity utilizing asynchronous dynamic cellular learning automata*. Applied Intelligence, vol. 48, no. 2, 2018, pp. 271–299.

[113] Bandini, S. et al.: *An Asynchronous Cellular Automata-Based Adaptive Illumination Facility*. In: *AI*IA 2009: Emergent Perspectives in Artificial Intelligence, XIth International Conference of the Italian Association for Artificial Intelligence Reggio Emilia, Italy, December 9-12, 2009*. Ed. by R. Serra and R. Cucchiara, Lecture Notes in Computer Science, no. 5883, Berlin, Heidelberg, Springer Berlin Heidelberg, 2009, pp. 405–415.

[114] Bandini, S. et al.: *A Cellular Automata-Based Modular Lighting System*. In: *9th International Conference on Cellular Automata for Research and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 6350, Berlin, Heidelberg, Springer Berlin Heidelberg, 2010, pp. 334–344.

[115] O'Sullivan, D. and Torrens, P. M.: *Cellular models of urban systems*. In: *Theory and practical issues on cellular automata*. Springer, 2001, pp. 108–116.

[116] Ke, X., Qi, L., and Zeng, C.: *A partitioned and asynchronous cellular automata model for urban growth simulation*. International Journal of Geographical Information Science, vol. 30, no. 4, 2016, pp. 637–659.

[117] Margolus, N.: *Physics-like models of computation*. Physica D: Nonlinear Phenomena, vol. 10, no. 1, 1984, pp. 81–95.

[118] Kari, J.: *Reversible Cellular Automata*. In: *Proceedings of the 9th International Conference on Developments in Language Theory*. DLT'05, Palermo, Italy, Springer-Verlag, 2005, pp. 57–68.

[119] Toffoli, T. and Margolus, N.: *Cellular automata machines: a new environment for modeling*. MIT Press, 1987.

[120] Bastien, C. and Michel, D.: *Cellular automata modeling of physical systems*. London Cambridge Univ Press, 1998.

[121] Rajchenbach, J. et al.: *Experiments on bidimensional models of sand: study of the dynamics*. In: *Scale Invariance, Interfaces, and Non-Equilibrium Dynamics*. Ed. by A. McKane et al., Boston, MA, Springer, 1995, pp. 313–327.

[122] Toffoli, T. and Margolus, N.: *Invertible cellular automata: A review*. Physica D, vol. 45, 1991, pp. 229–253.

[123] Morita, K., Margenstern, M., and Imai, K.: *Universality of reversible hexagonal cellular automata*. RAIRO-Theoretical Informatics and Applications, vol. 33, no. 6, 1999, pp. 535–550.

[124] Encinas, L. H. et al.: *Modelling forest fire spread using hexagonal cellular automata*. Applied Mathematical Modelling, vol. 31, no. 6, 2007, pp. 1213–1227.

[125] Trunfio, G. A.: *Predicting wildfire spreading through a hexagonal cellular automata model*. In: *6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004*. Ed. by P. M. A. Sloot et al., Lecture Notes in Computer Science, no. 3305, Springer Berlin Heidelberg, 2004, pp. 385–394.

[126] Bays, C.: *Cellular Automata in Triangular, Pentagonal and Hexagonal Tessellations*. In: *Encyclopedia of Complexity and Systems Science*. Ed. by R. A. Meyers, Springer New York, 2009, pp. 892–900.

[127] Róka, Z.: *Simulations between cellular automata on Cayley graphs*. Theoretical Computer Science, vol. 225, no. 1-2, 1999, pp. 81–111.

[128] Marr, C. and Hütt, M.-T.: *Cellular automata on graphs: Topological properties of ER graphs evolved towards low-entropy dynamics*. Entropy, vol. 14, no. 6, 2012, pp. 993–1010.

[129] Topa, P., Dzwinel, W., and Yuen, D. A.: *A multiscale cellular automata model for simulating complex transportation systems*. International Journal of Modern Physics C, vol. 17, no. 10, 2006, pp. 1437–1459.

[130] Chen, Q. and Mynett, A. E.: *Effects of cell size and configuration in cellular automata based prey–predator modelling*. Simulation Modelling Practice and Theory, vol. 11, no. 7, 2003, pp. 609–625.

[131] Ménard, A. and Marceau, D. J.: *Exploration of spatial scale sensitivity in geographic cellular automata*. Environment and Planning B: Planning and Design, vol. 32, no. 5, 2005, pp. 693–714.

[132] Kocabas, V. and Dragicevic, S.: *Assessing cellular automata model behaviour using a sensitivity analysis approach*. Computers, Environment and Urban Systems, vol. 30, no. 6, 2006, pp. 921–953.

[133] White, R. and Engelen, G.: *Cellular automata and fractal urban form: a cellular modelling approach to the evolution of urban land-use patterns*. Environment and planning A, vol. 25, no. 8, 1993, pp. 1175–1199.

[134] Wąs, J., Gudowski, B., and Matuszyk, P. J.: *Social distances model of pedestrian dynamics*. In: *Cellular automata : 7th international conference on Cellular Automata for research and industry, ACRI 2006: Perpignan, France, September 20–23, 2006*. Ed. by S. El Yacoubi, B. Chopard, and S. Bandini, Lecture Notes in Computer Science, no. 4173, Springer Berlin Heidelberg, 2006, pp. 492–501.

[135] Moreno, N., Wang, F., and Marceau, D. J.: *Implementation of a dynamic neighborhood in a land-use vector-based cellular automata model*. Computers, Environment and Urban Systems, vol. 33, no. 1, 2009, pp. 44–54.

[136] Li, X.-G. et al.: *A realistic two-lane cellular automata traffic model considering aggressive lane-changing behavior of fast vehicle*. Physica A: Statistical Mechanics and Its Applications, vol. 367, 2006, pp. 479–486.

[137] Chopard, B., Luthi, P. O., and Queloz, P.-A.: *Cellular automata model of car traffic in a two-dimensional street network*. Journal of Physics A: Mathematical and General, vol. 29, no. 10, 1996, pp. 2325.

[138] Dupuis, A. and Chopard, B.: *Parallel simulation of traffic in Geneva using cellular automata*. Parallel and Distributed Computing Practices Journal, vol. 1, 2001, pp. 89–107.

[139] Wastavino, L. et al.: *Modeling traffic on crossroads*. Physica A: Statistical Mechanics and Its Applications, vol. 381, 2007, pp. 411–419.

[140] Camara, D.: *Non-Uniform Cellular Automata. A Review*. Department of Computer Science University of Maryland, 2009.

[141] Vichniac, G. Y., Tamayo, P., and Hartman, H.: *Annealed and quenched inhomogeneous cellular automata (INCA)*. Journal of Statistical Physics, vol. 45, no. 5, 1986, pp. 875–883.

[142] Cattaneo, G. et al.: *Non-uniform Cellular Automata*. In: *Language and Automata Theory and Applications LATA 2009, Tarragona, Spain, April 2-8, 2009*. Ed. by A. H. Dediu, A. M. Ionescu, and C. Martín-Vide, Lecture Notes in Computer Science, no. 5457, Springer Berlin Heidelberg, 2009.

[143] Dennunzio, A., Formenti, E., and Provillard, J.: *Non-uniform cellular automata: Classes, dynamics, and decidability*. Information and Computation, vol. 215, 2012, pp. 32–46.

[144] Dennunzio, A., Formenti, E., and Provillard, J.: *Non-uniform cellular automata: Classes, dynamics, and decidability*. Information and Computation, vol. 215, 2012, pp. 32–46.

[145] Sipper, M.: *Quasi-Uniform Computation-Universal cellular automata*. In: *Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain, June 4 - 6, 1995 Proceedings*. Ed. by F. Morán et al., Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), no. 929, Springer Berlin Heidelberg, 1995, pp. 544–554.

[146] Ren, G. et al.: *Heterogeneous cellular automata model for straight-through bicycle traffic at signalized intersection*. Physica A: Statistical Mechanics and Its Applications, vol. 451, 2016, pp. 70–83.

[147] Lan, L. W. and Chang, C.-W.: *Inhomogeneous cellular automata modeling for mixed traffic with cars and motorcycles*. Journal of advanced transportation, vol. 39, no. 3, 2005, pp. 323–349.

[148] Mallikarjuna, C. and Rao, K. R.: *Cellular automata model for heterogeneous traffic*. Journal of Advanced Transportation, vol. 43, no. 3, 2009, pp. 321–345.

[149] Małecki, K.: *The Use of Heterogeneous Cellular Automata to Study the Capacity of the Roundabout*. In: *Artificial Intelligence and Soft Computing, 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11-15, 2017*. Ed. by L. Rutkowski et al., Lecture Notes in Computer Science, no. 10246, 2017, pp. 308–317.

[150] López, L., Burguerner, G., and Giovanini, L.: *Addressing population heterogeneity and distribution in epidemics models using a cellular automata approach*. BMC Research Notes, vol. 7, no. 1, 2014, pp. 234.

[151] Medernach, D. et al.: *Long-term evolutionary dynamics in heterogeneous cellular automata*. In: *GECCO '13 Proceedings of the 15th annual conference on Genetic and evolutionary computation, Amsterdam, The Netherlands — July 06 - 10, 2013*. ACM. 2013, pp. 231–238.

[152] Ryan, C. et al.: *Evolution of heterogeneous cellular automata in fluctuating environments*. In: *Proceedings of the European Conference on Artificial Life, ALIFE 2016, Cancun, Mexico, July 4 - 8, 2016*. MIT Press, 2016, pp. 216–223.

[153] Phon-Amnuaisuk, S.: *Composing using heterogeneous cellular automata*. In: *EvoWorkshops 2009: Proceedings of Workshops on Applications of Evolutionary Computation, Tübingen, Germany, April 15-17, 2009*. Ed. by M. Giacobini et al., Lecture Notes in Computer Science, no. 5484, Springer Berlin Heidelberg, 2009, pp. 547–556.

[154] Ermentrout, G. B. and Edelstein-Keshet, L.: *Cellular automata approaches to biological modeling*. Journal of Theoretical Biology, vol. 160, no. 1, 1993, pp. 97–133.

[155] Cattaneo, G., Dennunzio, A., and Farina, F.: *Towards a Two-Scale Cellular Automata Model of Tumour-Induced Angiogenesis*. In: *Cellular automata : 7th international conference on Cellular Automata for research and industry, ACRI 2006: Perpignan, France, September 20–23, 2006*. Ed. by Stefania Bandini [et al.]., Lecture Notes in Computer Science, no. 4173, Springer Berlin Heidelberg, 2006, pp. 446–451.

[156] Farina, F. and Dennunzio, A.: *A predator–prey cellular automaton with parasitic interactions and environmental effects*. Fundamenta Informaticae, vol. 83, no. 4, 2008, pp. 337–353.

[157] Wooldridge, M. and Jennings, N. R.: *Intelligent agents: Theory and practice*. The Knowledge Engineering Review, vol. 10, no. 2, 1995, pp. 115–152.

[158] Railsback, S. F. and Grimm, V.: *Agent-based and individual-based modeling: a practical introduction*. Princeton University Press, 2011.

[159] Shoham, Y.: *Agent-oriented programming*. Artificial intelligence, vol. 60, no. 1, 1993, pp. 51–92.

[160] Cetnarowicz, K.: *From algorithm to agent*. In: *9th International Conference on Computational Science, Baton Rouge, USA, May 25-27, 2009*. Ed. by G. Allen et al., Lecture Notes in Computer Science, no. 5545, Springer Berlin Heidelberg, 2009, pp. 825–834.

[161] Shanthi, M. and Rajan, E.: *Agent based cellular automata: A novel approach for modeling spatiotemporal growth processes.* International Journal of Application or Innovation in Engineering and Management, vol. 1, no. 3, 2012, pp. 56–61.

[162] Muci, A. L. et al.: *A combination of cellular automata and agent-based models for simulating the root surface colonization by bacteria*. Ecological modelling, vol. 247, 2012, pp. 1–10.

[163] Elalaouy, E., Rhoulami, K., and Rahmani, M. D.: *A Novel Modeling based Agent Cellular Automata for Advanced Residential Mobility Applications*. International Journal of Advanced Computer Science and Applications, vol. 8, no. 7, 2017, pp. 337–343.

[164] Mustafa, A. et al.: *Coupling agent-based, cellular automata and logistic regression into a hybrid urban expansion model (HUEM)*. Land Use Policy, vol. 69, 2017, pp. 529–540.

[165] Topa, P. and Paszkowski, M.: *Anastomosing transportation networks*. In: *Parallel Processing and Applied Mathematics: 4th International Conference, PPAM'2001 Nałęczów, Poland, September 9–12, 2001: revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 2328, Berlin, Heidelberg, Springer Berlin Heidelberg, 2002, pp. 904–911.

[166] Topa, P. and Dzwinel, W.: *Consuming Environment with Transportation Network Modeled Using Graph of Cellular*. In: *Parallel Processing and Applied Mathematics 5th International Conference, PPAM 2003, Czestochowa, Poland, September 7-10, 2003. Revised Papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 3019, Springer Berlin Heidelberg, 2004, pp. 513–520.

[167] D'Ambrosio, D. et al.: *A cellular automata model for soil erosion by water*. Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere, vol. 26, no. 1, 2001, pp. 33–39.

[168] Rinaldi, P. R. et al.: *Cellular automata algorithm for simulation of surface flows in large plains*. Simulation Modelling Practice and Theory, vol. 15, no. 3, 2007, pp. 315–327.

[169] Shao, Q. et al.: *RunCA: A cellular automata model for simulating surface runoff at different scales*. Journal of Hydrology, vol. 529, 2015, pp. 816–829.

[170] Thomas, R. and Nicholas, A.: *Simulation of braided river flow using a new cellular routing scheme*. Geomorphology, vol. 43, no. 3, 2002, pp. 179–195.

[171] Coulthard, T. J. and Wiel, M. J. V. D.: *A cellular model of river meandering*. Earth Surface Processes and Landforms: The Journal of the British Geomorphological Research Group, vol. 31, no. 1, 2006, pp. 123–132.

[172] Gradziński, R. et al.: *Anastomosing system of the upper Narew river, NE Poland*. Annales Societatis Geologorum Poloniae, vol. 70, no. 3-4, 2000, pp. 219–229.

[173] Gradziński, R. et al.: *Vegetation-controlled modern anastomosing system of the upper Narew River (NE Poland) and its sediments*. Sedimentary Geology, vol. 157, no. 3-4, 2003, pp. 253–276.

[174] Makaske, B.: *Anastomosing rivers: a review of their classification, origin and sedimentary products*. Earth-Science Reviews, vol. 53, no. 3-4, 2001, pp. 149–196.

[175] Makaske, B.: *Anastomosing rivers: forms, processes and sediments*. Koninklijk Nederlands Aardrijkskundig Genootschap, 1998.

[176] Di Gregorio, S. and Serra, R.: *An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata*. Future Generation Computer Systems, vol. 16, no. 2-3, 1999, pp. 259–271.

[177] Topa, P. and Młocek, P.: *Using Shared Memory As a Cache in Cellular Automata*. Computer Science, vol. 14, no. 3, 2013, pp. 385–401.

[178] Topa, P. and Paszkowski, M.: *A cellular automata models of evolution of transportation networks*. Computer Science, vol. 4, 2002, pp. 7–19.

[179] Topa, P.: *A distributed cellular automata simulation on clusters of PCs*. In: *Proceedings of International Conference on Computational Science, ICCS'02, Amsterdam, The Netherlands, April 21 – 24, 2002*. Lecture Notes in Computer Science, no. 2331, Berlin, Heidelberg, Springer Berlin Heidelberg, 2002, pp. 783–792.

[180] Topa, P.: *A cellular automata approach for modelling complex river systems*. In: *Cellular automata : 7th international conference on Cellular Automata for research and industry, ACRI 2006: Perpignan, France, September 20–23, 2006*. Ed. by S. Bandini et al., Lecture Notes in Computer Science, no. 4173, Springer Berlin Heidelberg, 2006, pp. 482–491.

[181] Topa, P. and Mlocek, P.: *GPGPU Implementation of Cellular Automata Model of Water Flow*. In: *Parallel Processing and Applied Mathematics: 9th international conference, PPAM 2011: Toruń, Poland, September 11–14, 2011 : revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 7203, Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 630–639.

[182] Margolus, N., Toffoli, T., and Vichniac, G.: *Cellular Automata Supercomputers for Fluid-Dynamics Modeling*. Physical Review Letters, vol. 56, no. 16, 1986, pp. 1694–1696.

[183] Cannataro, M. et al.: *A parallel cellular automata environment on multicomputers for computational science*. Parallel Computing, vol. 21, no. 5, 1995, pp. 803–823.

[184] Spezzano, G. and Talia, D.: *Programming cellular automata algorithms on parallel computers*. Future Generation Computer Systems, vol. 16, no. 2-3, 1999, pp. 203–216.

[185] Topa, P.: *Informatyczne modele wzrostu w wybranych zagadnieniach geologii*. PhD Thesis. AGH University of Science and Technology, 2005.

[186] Tran, J., Jordan, D., and Luebke, D.: *New challenges for cellular automata simulation on the GPU*. In: *Proceedings of SIGGRAPH Conference, Los Angeles, US*. 2004.

[187] Gibson, M. J., Keedwell, E. C., and Savić, D. A.: *An investigation of the efficient implementation of cellular automata on multi-core CPU and GPU hardware*. Journal of Parallel and Distributed Computing, vol. 77, 2015, pp. 11–25.

[188] Rybacki, S., Himmelspach, J., and Uhrmacher, A. M.: *Experiments with single core, multi-core, and GPU based computation of cellular automata*. In: *1st International Conference on Advances in System Simulation 2009, September 20–25, Washington, USA*. IEEE Computer Society. 2009, pp. 62–67.

[189] Millán, E. N. et al.: *Performance analysis and comparison of cellular automata GPU implementations*. Cluster Computing, vol. 20, no. 3, 2017, pp. 2763–2777.

[190] Bolz, J. et al.: *Sparse matrix solvers on the GPU: conjugate gradients and multigrid*. Transactions on Graphics, vol. 22, no. 3, 2003, pp. 917–924.

[191] Krüger, J. and Westermann, R.: *Linear algebra operators for GPU implementation of numerical algorithms*. Transactions on Graphics, vol. 22, no. 3, 2003, pp. 908–916.

[192] Harris, M.: *Mapping computational concepts to GPUs*. In: *ACM SIGGRAPH 2005 Courses*. ACM. 2005, pp. 50.

[193] Kurdziel, M. and Boryczko, K.: *Dense Affinity Propagation on Clusters of GPUs*. In: *Parallel Processing and Applied Mathematics: 9th international conference, PPAM 2011 : Toruń, Poland, September 11–14, 2011 : revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 7203, Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 599–608.

[194] Marks, M. et al.: *Heterogeneous GPU&CPU cluster for high performance computing in cryptography*. Computer Science, vol. 13, 2012, pp. 63–79.

[195] Topa, P., Kuźniar, M., and Dzwinel, W.: *Graph of cellular automata as a metaphor of fusarium graminearum growth implemented in GPGPU CUDA computational environment*. In: *Parallel Processing and Applied Mathematics : 9th international conference, PPAM 2011 : Toruń, Poland, September 11–14, 2011 : revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 7204, Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 578–587.

[196] Topa, P.: *Cellular Automata Model Tuned for Efficient Computation on GPU with Global Memory Cache*. In: *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Torino, Italy, February 12-14, 2014*. Ed. by M. Aldinucci, D. D'Agostino, and P. Kilpatrick, IEEE Computer Society, 2014, pp. 380–383.

[197] Zheng, X., Zhong, T., and Liu, M.: *Modeling crowd evacuation of a building based on seven methodological approaches*. Building and Environment, vol. 44, no. 3, 2009, pp. 437–445.

[198] Pelechano, N. and Malkawi, A.: *Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches*. Automation in Construction, vol. 17, no. 4, 2008, pp. 377–385.

[199] Kirchner, A. and Schadschneider, A.: *Cellular Automaton Simulations of Pedestrian Dynamics and Evacuation Processes*. In: *Traffic and Granular Flow'01*. Ed. by M. Fukui et al., Berlin, Heidelberg, Springer Berlin Heidelberg, 2003, pp. 531–536.

[200] Nishinari, K. et al.: *Extended floor field CA model for evacuation dynamics*. IEICE Transactions on Information and Systems, vol. 87, no. 3, 2004, pp. 726–732.

[201] Wąs, J. and Lubaś, R.: *Towards realistic and effective agent-based models of crowd dynamics*. Neurocomputing, vol. 146, 2014, pp. 199–209.

[202] Peng, Y.-C. and Chou, C.-I.: *Simulation of pedestrian flow through a "T" intersection: A multi-floor field cellular automata approach*. Computer Physics Communications, vol. 182, no. 1, 2011, pp. 205–208.

[203] Helbing, D. et al.: *Simulation of pedestrian crowds in normal and evacuation situations*. Pedestrian and Evacuation Dynamics, vol. 21, no. 2, 2002, pp. 21–58.

[204] Mróz, H., Wąs, J., and Topa, P.: *The use of GPGPU in continuous and discrete models of crowd dynamics*. In: *Parallel Processing and Applied Mathematics: 10th international conference, PPAM 2013: Warsaw, Poland, September 8–11, 2013: revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 8385, Berlin, Heidelberg, Springer Berlin Heidelberg, 2014, pp. 679–688.

[205] Wąs, J., Mróz, H., and Topa, P.: *GPGPU computing for microscopic simulations of crowd dynamics*. Computing and Informatics, vol. 34, no. 6, 2015, pp. 1418–1434.

[206] Wąs, J. and Lubaś, R.: *Adapting Social Distances Model for Mass Evacuation Simulation.* Journal of Cellular Automata, vol. 8, 2013.

[207] Lubaś, R. et al.: *Verification and Validation of Evacuation Models – Methodology Expansion Proposition*. Transportation Research Procedia, vol. 2, 2014, pp. 715–723.

[208] Lubaś, R. et al.: *Validation and Verification of CA-Based Pedestrian Dynamics Models.* Journal of Cellular Automata, vol. 11, no. 4, 2016, pp. 285–298.

[209] Gawad, J. and Pietrzyk, M.: *Application of CAFE multiscale model to description of microstructure development during dynamic recrystallization*. Archives of Metallurgy and Materials, vol. 52, no. 2, 2007, pp. 257.

[210] Alemani, D. et al.: *Combining cellular automata and lattice Boltzmann method to model multiscale avascular tumor growth coupled with nutrient diffusion and immune competition*. Journal of Immunological Methods, vol. 376, no. 1-2, 2012, pp. 55–68.

[211] Hoekstra, A. G. et al.: *Complex automata: multi-scale modeling with coupled cellular automata*. In: *Simulating complex systems by cellular automata*. Springer, 2010, pp. 29–57.

[212] Hoekstra, A. G. et al.: *Multi-scale Modeling with Cellular Automata: The Complex Automata Approach*. In: *Cellular automata : 8th international conference on Cellular automata for research and industry, ACRI 2008 : Yokohama, Japan, September 23–26, 2008*. Ed. by H. Umeo et al., Lecture

Notes in Computer Science, no. 5191, Springer Berlin Heidelberg, 2008, pp. 192–199.

[213] Róka, Z.: *One-way cellular automata on Cayley graphs*. Theoretical Computer Science, vol. 132, no. 1-2, 1994, pp. 259–290.

[214] O'Sullivan, D.: *Graph-cellular automata: a generalised discrete urban and regional model*. Environment and Planning B: Planning and Design, vol. 28, no. 5, 2001, pp. 687–705.

[215] Bak, P.: *How nature works: the science of self-organized criticality*. Springer Science & Business Media, 2013.

[216] Topa, P.: *Network Systems Modelled by Complex Cellular Automata Paradigm*. In: *Cellular Automata — Simplicity Behind Complexity*. Ed. by A. Salcido, Intech Open, 2010, pp. 259–274.

[217] Boccaletti, S. et al.: *Complex networks: Structure and dynamics*. Physics Reports, vol. 424, no. 4-5, 2006, pp. 175–308.

[218] Czech, W. et al.: *Exploring complex networks with graph investigator research application*. Computing and Informatics, vol. 30, no. 2, 2012, pp. 381–410.

[219] Bridge, J. S.: *The interaction between channel geometry, water flow, sediment transport and deposition in braided rivers*. Geological Society, London, Special Publications, vol. 75, no. 1, 1993, pp. 13–71.

[220] Rigon, R. et al.: *Optimal channel networks: a framework for the study of river basin morphology*. Water Resources Research, vol. 29, no. 6, 1993, pp. 1635–1646.

[221] Tarboton, D. G.: *Fractal river networks, Horton's laws and Tokunaga cyclicity*. Journal of Hydrology, vol. 187, no. 1-2, 1996, pp. 105–117.

[222] Rodriguez-Iturbe, I. and Rinaldo, A.: *Fractal river basins: chance and self-organization*. Cambridge University Press, 2001.

[223] Tarboton, D. G., Bras, R. L., and Rodriguez-Iturbe, I.: *The fractal nature of river networks*. Water Resources Research, vol. 24, no. 8, 1988, pp. 1317–1322.

[224] Turcotte, D. L.: *Self-organized complexity in geomorphology: Observations and models*. Geomorphology, vol. 91, no. 3-4, 2007, pp. 302–310.

[225] Carmeliet, P. and Jain, R. K.: *Angiogenesis in cancer and other diseases*. Nature, vol. 407, no. 6801, 2000, pp. 249.

[226] Carmeliet, P.: *Angiogenesis in life, disease and medicine*. Nature, vol. 438, no. 7070, 2005, pp. 932.

[227] Anderson, A. R. and Chaplain, M.: *Continuous and discrete mathematical models of tumor-induced angiogenesis*. Bulletin of Mathematical Biology, vol. 60, no. 5, 1998, pp. 857–899.

[228] Mantzaris, N. V., Webb, S., and Othmer, H. G.: *Mathematical modeling of tumor-induced angiogenesis*. Journal of Mathematical Biology, vol. 49, no. 2, 2004, pp. 111–187.

[229] Peirce, S. M.: *Computational and mathematical modeling of angiogenesis*. Microcirculation, vol. 15, no. 8, 2008, pp. 739–751.

[230] Stokes, C. L. and Lauffenburger, D. A.: *Analysis of the roles of microvessel endothelial cell random motility and chemotaxis in angiogenesis*. Journal of Theoretical Biology, vol. 152, no. 3, 1991, pp. 377–403.

[231] Plank, M., Sleeman, B., and Jones, P.: *A mathematical model of tumour angiogenesis, regulated by vascular endothelial growth factor and the angiopoietins*. Journal of Theoretical Biology, vol. 229, no. 4, 2004, pp. 435–454.

[232] Markus, M., Böhm, D., and Schmick, M.: *Simulation of vessel morphogenesis using cellular automata*. Mathematical Biosciences, vol. 156, no. 1-2, 1999, pp. 191–206.

[233] Patel, A. A. et al.: *A cellular automaton model of early tumor growth and invasion: the effects of native tissue vascularity and increased anaerobic tumor metabolism*. Journal of Theoretical Biology, vol. 213, no. 3, 2001, pp. 315–331.

[234] Alarcón, T., Byrne, H. M., and Maini, P. K.: *A cellular automaton model for tumour growth in inhomogeneous environment*. Journal of Theoretical Biology, vol. 225, no. 2, 2003, pp. 257–274.

[235] Macklin, P. et al.: *Multiscale modelling and nonlinear simulation of vascular tumour growth*. Journal of Mathematical Biology, vol. 58, no. 4-5, 2009, pp. 765–798.

[236] Bentley, K., Gerhardt, H., and Bates, P. A.: *Agent-based simulation of notch-mediated tip cell selection in angiogenic sprout initialisation*. Journal of Theoretical Biology, vol. 250, no. 1, 2008, pp. 25–36.

[237] Artel, A. et al.: *An agent-based model for the investigation of neovascularization within porous scaffolds*. Tissue Engineering Part A, vol. 17, no. 17-18, 2011, pp. 2133–2141.

[238] Bergers, G. and Song, S.: *The role of pericytes in blood-vessel formation and maintenance*. Neuro-oncology, vol. 7, no. 4, 2005, pp. 452–464.

[239] Owens, G. K.: *Regulation of differentiation of vascular smooth muscle cells*. Physiological Reviews, vol. 75, no. 3, 1995, pp. 487–517.

[240] Tonini, T., Rossi, F., and Claudio, P. P.: *Molecular basis of angiogenesis and cancer*. Oncogene, vol. 22, no. 42, 2003, pp. 6549.

[241] Owen, M. R. et al.: *Angiogenesis and vascular remodelling in normal and cancerous tissues*. Journal of Mathematical Biology, vol. 58, no. 4-5, 2009, pp. 689–721.

[242] McDougall, S. R. et al.: *Mathematical modelling of flow through vascular networks: implications for tumour-induced angiogenesis and chemotherapy strategies*. Bulletin of Mathematical Biology, vol. 64, no. 4, 2002, pp. 673–702.

[243] Topa, P. and Dzwinel, W.: *Using network descriptors for comparison of vascular systems created by tumour-induced angiogenesis*. Theoretical and Applied Informatics, vol. 21, no. 2, 2009, pp. 83–94.

[244] Czech, W. and Yuen, D. A.: *Efficient graph comparison and visualization using GPU*. In: *14th International Conference on Computational Science and Engineering (CSE), 2011 IEEE*. 2011, pp. 561–566.

[245] Bushnell, W., Hazen, B., and Pritsch, C.: *Histology and physiology of Fusarium head blight*. In: *Fusarium head blight of wheat and barley*. Ed. by K. J. Leonard and W. R. Bushnell, APS Press, 2003.

[246] McMullen, M. et al.: *A unified effort to fight an enemy of wheat and barley: Fusarium head blight*. Plant Disease, vol. 96, no. 12, 2012, pp. 1712–1728.

[247] Pritsch, C. et al.: *Fungal development and induction of defense response genes during early infection of wheat spikes by Fusarium graminearum*. Molecular Plant-Microbe Interactions, vol. 13, no. 2, 2000, pp. 159–169.

[248] Goswami, R. S. and Kistler, H. C.: *Heading for disaster: Fusarium graminearum on cereal crops*. Molecular Plant Pathology, vol. 5, no. 6, 2004, pp. 515–525.

[249] Prosser, J. I.: *Mathematical Modelling of Fungal Growth*. In: *The Growing Fungus*. Ed. by N. A. R. Gow and G. M. Gadd, Dordrecht, Springer Netherlands, 1995, pp. 319–335.

[250] Paustian, K. and Schnürer, J.: *Fungal growth response to carbon and nitrogen limitation: Application of a model to laboratory and field data*. Soil Biology and Biochemistry, vol. 19, no. 5, 1987, pp. 621–629.

[251] Lamour, A. et al.: *Modelling the growth of soil-borne fungi in response to carbon and nitrogen*. Mathematical Medicine and Biology: A Journal of the IMA, vol. 17, no. 4, 2000, pp. 329–346.

[252] Regalado, C. et al.: *The origins of spatial heterogeneity in vegetative mycelia: a reaction-diffusion model*. Mycological Research, vol. 100, no. 12, 1996, pp. 1473–1480.

[253] Halley, J. M. et al.: *Competition, succession and pattern in fungal communities: towards a cellular automaton model*. Oikos, 1994, pp. 435–442.

[254] Boswell, G. P. et al.: *A mathematical approach to studying fungal mycelia*. Mycologist, vol. 17, no. 4, 2003, pp. 165–171.

[255] Laszlo, J. A. and Silman, R. W.: *Cellular automata simulations of fungal growth on solid substrates*. Biotechnology advances, vol. 11, no. 3, 1993, pp. 621–633.

[256] Boswell, G. P.: *Modelling mycelial networks in structured environments*. Mycological Research, vol. 112, no. 9, 2008, pp. 1015–1025.

[257] Boswell, G. P. et al.: *The development of fungal networks in complex environments*. Bulletin of Mathematical Biology, vol. 69, no. 2, 2007, pp. 605.

[258] Miller, S. S. et al.: *Use of a Fusarium graminearum strain transformed with green fluorescent protein to study infection in wheat (Triticum aestivum)*. Canadian Journal of Plant Pathology, vol. 26, no. 4, 2004, pp. 453–463.

[259] Wcisło, R. and Dzwinel, W.: *Particle Based Model of Tumor Progression Stimulated by the Process of Angiogenesis*. In: *8th International Conference on Computational Science ICCS 2008, Kraków, Poland, June 23-25, 2008, Proceedings*. Ed. by M. Bubak et al., Lecture Notes in Computer Science, no. 5102, Springer Berlin Heidelberg, 2008, pp. 177–186.

[260] Wcisło, R. et al.: *A 3-D model of tumor progression based on complex automata driven by particle dynamics*. Journal of Molecular Modeling, vol. 15, no. 12, 2009, pp. 1517.

[261] Wooldridge, M.: *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[262] Byrski, A.: *Agent-based metaheuristics in search and optimisation*. AGH University of Science and Technology Press, 2013.

[263] Russell, S. J. and Norvig, P.: *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[264] Brooks, R. A. et al.: *Intelligence without reason*. Artificial Intelligence: Critical Concepts, vol. 3, 1991, pp. 107–63.

[265] Muller, J. P. and Müller, J. P.: *The design of intelligent agents: a layered approach*. Vol. 1177. Springer Science & Business Media, 1996.

[266] Uhrmacher, A. M. and Weyns, D.: *Multi-Agent systems: Simulation and applications*. CRC Press, 2009.

[267] Jennings, N. R., Sycara, K., and Wooldridge, M.: *A roadmap of agent research and development*. Autonomous Agents and Multi-agent Systems, vol. 1, no. 1, 1998, pp. 7–38.

[268] Kisiel-Dorohnicki, M.: *Agentowe architektury populacyjnych systemów inteligencji obliczeniowej*. Vol. 269. Rozprawy Monografie. Kraków, Wydawnctwa AGH, 2013.

[269] Rousset, A. et al.: *A survey on parallel and distributed multi-agent systems for high performance computing simulations*. Computer Science Review, vol. 22, 2016, pp. 27–46.

[270] Grimm, V. and Railsback, S. F.: *Individual-based modeling and ecology*. Princeton University Press, 2013.

[271] Wangersky, P. J.: *Lotka-Volterra population models*. Annual Review of Ecology and Systematics, vol. 9, no. 1, 1978, pp. 189–218.

[272] DeAngelis, D. L.: *Individual-based models and approaches in ecology: populations, communities and ecosystems*. CRC Press, 2018.

[273] DeAngelis, D. and Grimm, V.: *Individual-based models in ecology after four decades*. F1000prime Reports, vol. 6, no. 39, 2014, pp. 1–6.

[274] Goldberg, D. E.: *Genetic algorithms*. Pearson Education India, 2006.

[275] Schwefel, H.: *Evolution and optimum seeking*. John Wiley & Sons, 1995.

[276] Back, T., Hammel, U., and Schwefel, H.-P.: *Evolutionary computation: Comments on the history and current state*. IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, 1997, pp. 3–17.

[277] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 1989.

[278] Obuchowicz, A.: *Evolutionary algorithms for global optimization and dynamic system diagnosis*. Lubusky Scientific Society, 2003.

[279] Cetnarowicz, K., Kisiel-Dorohinicki, M., and Nawarecki, E.: *The application of evolution process in multi-agent world to the prediction system*. In: *Proceedings of the Second International Conference on Multi-Agent Systems, ICMAS*. Vol. 96. 1996, pp. 26–32.

[280] Kisiel-Dorohinicki, M.: *Agent-Oriented Model of Simulated Evolution*. In: *SOFSEM 2002: Theory and Practice of Informatics*. Ed. by W. I. Grosky and F. Plášil, Springer Berlin Heidelberg, 2002, pp. 253–261.

[281] Pieter, J. and Jong, E. D. de: *Evolutionary multi-agent systems*. In: *International Conference on Parallel Problem Solving from Nature*. Springer Berlin Heidelberg, 2004, pp. 872–881.

[282] Byrski, A. et al.: *Evolutionary multi-agent systems*. The Knowledge Engineering Review, vol. 30, no. 2, 2015, pp. 171–186.

[283] Krzywicki, D. et al.: *Computing agents for decision support systems*. Future Generation Computer Systems, vol. 37, 2014, pp. 390–400.

[284] Siwik, L. and Kisiel-Dorohinicki, M.: *Semi-elitist Evolutionary Multi-agent System for Multiobjective Optimization*. In: *6th International Conference on Computational Science ICCS 2006, Reading, UK, May 28-31, 2006*. Ed. by V. N. Alexandrov et al., Lecture Notes in Computer Science, no. 3993, Springer Berlin Heidelberg, 2006, pp. 831–838.

[285] Czerwinski, B., Debski, R., and Pietak, K.: *Distributed Agent-Based Platform for Large Scale Evolutionary Computations*. In: *International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2011, June 30 - July 2, 2011, Korean Bible University, Seoul, Korea*. 2011, pp. 462–466.

[286] Faber, Ł. et al.: *Agent-Based Simulation in AgE Framework*. In: *Advances in Intelligent Modelling and Simulation: Simulation Tools and Applications*. Ed. by A. Byrski et al., Springer Berlin Heidelberg, 2012, pp. 55–83.

[287] Hemleben, C., Spindler, M., and Anderson, O. R.: *Modern planktonic foraminifera*. Springer Science & Business Media, 2012.

[288] Murray, J. W.: *Ecology and palaeoecology of benthic foraminifera*. Routledge, 2014.

[289] Signes, M. et al.: *A model for planktic foraminiferal shell growth*. Paleobiology, vol. 19, no. 1, 1993, pp. 71–91.

[290] Brasier, M. D.: *Architecture and evolution of the foraminiferid test—a theoretical approach*. In: *Aspects of Micropalaeontology*. Ed. by F. T. Banner and A. R. Lord, Springer, 1982, pp. 1–41.

[291] De Renzi, M.: *Shell coiling in some larger foraminifera: general comments and problems*. Paleobiology, vol. 14, no. 4, 1988, pp. 387–400.

[292] Tyszka, J. and Topa, P.: *A new approach to modeling of foraminiferal shells*. Paleobiology, vol. 31, no. 3, 2005, pp. 522–537.

[293] Topa, P. and Tyszka, J.: *Local minimization paradigm in numerical modelling of foraminiferal shells*. In: *International Conference on Computational Science ICCS 2002, Amsterdam, The Netherlands, April 21–24, 2002*. Ed. by P. M. A. Sloot et al., Lecture Notes in Computer Science, no. 2329, Springer Berlin Heidelberg, 2002, pp. 97–106.

[294] Tyszka, J.: *Morphospace of foraminiferal shells: results from the moving reference model*. Lethaia, vol. 39, no. 1, 2006, pp. 1–12.

[295] Komosinski, M. et al.: *Multi-agent simulation of benthic foraminifera response to annual variability of feeding fluxes*. Journal of Computational Science, vol. 21, 2017, pp. 419–431.

[296] Topa, P. et al.: *Modelling ecology and evolution of Foraminifera in the agent-oriented distributed platform*. Journal of Computational Science, vol. 18, 2017, pp. 69–84.

[297] Komosinski, M. et al.: *Application of a Morphological Similarity Measure to the Analysis of Shell Morphogenesis in Foraminifera*. In: *Man–Machine Interactions 4: 4th International Conference on Man–Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6–9, 2015*. Ed. by A. G. et al. (eds.), vol. 391. Advances in Intelligent Systems and Computing, no. 391, 2016, pp. 216–225.

[298] Gwiazda, T. D.: *Algorytmy genetyczne: kompendium. Operator mutacji dla problemów numerycznych*. Wydawnictwo Naukowe PWN, 2007.

[299] Piętak, K. and Topa, P.: *Towards Multi-Agent Simulations Accelerated by GPU*. In: *12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017, revised selected papers*. Ed. by R. Wyrzykowski et al., Lecture Notes in Computer Science, no. 10778, Springer Berlin Heidelberg, 2018, pp. 456–465.

[300] Richmond, P. and Romano, D.: *Template-driven agent-based modeling and simulation with CUDA*. In: *GPU Computing Gems Emerald Edition*. Elsevier, 2011, pp. 313–324.

[301] Kaziród, M. et al.: *Agent-oriented foraminifera habitat simulation*. Procedia Computer Science, vol. 51, no. 1, 2015, pp. 1062–1071.

[302] Bujas, J. et al.: *High-performance computing framework with desynchronized information propagation for large-scale simulations*. Journal of Computational Science, 2018. In press.