16

Cellular automata simulations - tools and techniques

Henryk Fukś
Brock University
Canada

1. Introduction

The purpose of this chapter is to provide a concise introduction to cellular automata simulations, and to serve as a starting point for those who wish to use cellular automata in modelling and applications. No previous exposure to cellular automata is assumed, beyond a standard mathematical background expected from a science or engineering researcher.

Cellular automata (CA) are dynamical systems characterized by discreteness in space, time, and in state variables. In general, they can be viewed as cells in a regular lattice updated synchronously according to a local interaction rule, where the state of each cell is restricted to a finite set of allowed values. Unlike other dynamical systems, the idea of cellular automaton can be explained without using any advanced mathematical apparatus. Consider, for instance, a well-known example of the so-called majority voting rule. Imagine a group of people arranged in a line line who vote by raising their right hand. Initially some of them vote "yes", others vote "no". Suppose that at each time step, each individual looks at three people in his *direct neighbourhood* (himself and two nearest neighbours), and updates his vote as dictated by the majority in the neighbourhood. If the variable $s_i(t)$ represents the vote of the i-th individual at the time t (assumed to be an integer variable), we can write the *CA rule* representing the voting process as

$$s_i(t+1) = \text{majority}\{s_{i-1}(t), s_i(t), s_{i+1}(t)\}. \tag{1}$$

This is illustrated in Figure 1, where periodic boundary conditions are used, that is, the right

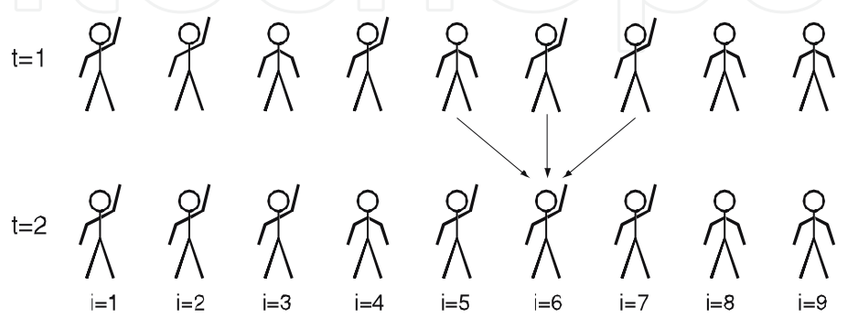


Fig. 1. Example of CA: majority voting rule.

neighbour of $i = 9$ is considered to be $i = 1$, and similarly for $i = 1$. If eq. (1) is iterated many times for $t = 1, 2, 3, \ldots$, we have a discrete-time dynamical system, often referred to as one-dimensional cellular automaton. Of course, in a general case, instead of eq. (1) we can use another rule, and instead of one-dimensional lattice we may consider higher-dimensional structures. Voting rule in 2-dimensions, for example, could be defined as

$$s_{i,j}(t+1) = \text{majority}\left\{s_{i-1,j}(t), s_{i+1,j}(t), s_{i,j}(t), s_{i,j-1}(t), s_{i,j+1}(t)\right\}, \tag{2}$$

where the variable $s_{i,j}(t)$ represents the vote of the individual located at (i, j) at time t, assuming that i, j and t are all integer variables. Again, by iterating the above rule for $t = 1, 2, 3, \ldots$, we obtain a two-dimensional dynamical system, known as two-dimensional majority-voting cellular automaton.

Given the simplicity of CA definition, it is not surprising that cellular automata enjoy tremendous popularity as a modelling tool, in a number of diverse fields of science and engineering. While it is impossible to list all applications of CA in this short chapter, we will mention some monographs and other publications which could serve as a starting point for exploring CA-based models. The first serious area of applications of CA opened in mid-80's, when the development of lattice gas cellular automata (LGCA) for hydrodynamics initiated an extensive growth of CA-based models in fluid dynamics, including models of Navier-Stokes fluids and chemically reacting systems. Detailed discussion of these models, as well as applications of CA in nonequilibrium phase transition modelling can be found in (Chopard and Drozd, 1998). In recent years, the rise of fast and inexpensive digital computers brought a new wave of diverse applications of CA in areas ranging from biological sciences (e.g., population dynamics, immune system models, tumor growth models, etc.) to engineering (e.g. models of electromagnetic radiation fields around antennas, image processing, interaction of remote sensors, traffic flow models, etc.) An extensive collection of articles on applications of cellular automata can be found in a series of conference volumes produced bi-annually in connection with International Conference on Cellular Automata for Research and Industry (Bandini et al., 2002; Sloot et al., 2004; Umeo et al., 2008; Yacoubi et al., 2006) as well as in Journal of Cellular Automata, a new journal launched in 2006 and dedicated exclusively to CA theory and applications. Among the applications listed above, traffic flow models should be singled out as one of the most important and extensively studies areas. A good albeit already somewhat dated review of these models can be found in (Chowdhury et al., 2000). For readers interested in the theory of cellular automata, computational aspects of CA are discussed in (Ilachinski, 2001; Wolfram, 1994; 2002), while more mathematical approach is presented in Kari (2005).

Discrete nature of cellular automata renders CA-based models suitable for computer simulations and computer experimentation. One can even say that computer simulations are almost mandatory for anyone who wants to understand behavior of CA-based model: apparent simplicity of CA definition is rather deceiving, and in spite of this simplicity, dynamics of CA can be immensely complex. In many cases, very little can be done with existing mathematical methods, and computer simulations are the only choice.

In this chapter, we will describe basic problems and methods for CA simulations, starting from the definition of the "simulation" in the context of CA, and followed by the discussion of various types of simulations, difficulties associated with them, and methods used to resolve these difficulties. Most ideas will be presented using one-dimensional examples for the sake of clarity.

2. Cellular automata

We will start from some remarks relating CA to partial differential equations, which are usually more familiar to modelers that CA. Cellular automata are often described as fully discrete analogs of partial differential equations (PDEs). In one dimension, PDE which is first-order in time can be written as

$$u_t(x,t) = F(u, u_x, u_{xx}, \ldots), \tag{3}$$

where $u(x,t)$ is the unknown function, and subscripts indicate differentiation. Informally, cellular automata can be obtained by replacing derivatives in (3) by difference quotients

$$u_t \rightarrow \frac{u(x,t+\epsilon) - u(x,t)}{\epsilon}, \tag{4}$$

$$u_x \rightarrow \frac{u(x+h,t) - u(x-h,t)}{2h}, \tag{5}$$

$$u_{xx} \rightarrow \frac{u(x+2h,t) - 2u(x,t) + u(x-2h,t)}{4h^2}, \tag{6}$$

etc. With these substitutions, and by taking $h = \epsilon = 1$ one can rewrite (3) as

$$u(x,t+1) = u(x,t) + F\left(u, \frac{u(x+1,t) - u(x-1,t)}{2}, \frac{u(x+2,t) - 2u(x,t) + u(x-2,t)}{4}, \ldots\right). \tag{7}$$

One can now see that the right hand side of the above equation depends on

$$u(x,t), u(x \pm 1,t), u(x \pm 2,t), \ldots \tag{8}$$

and therefore we can rewrite rewrite (7) as

$$u(x,t+1) = f(u(x-r,t), u(x-r+1,t), \ldots, u(x+r,t)), \tag{9}$$

where f is called a *local function* and the integer r is called a *radius* of the cellular automaton. The local function for cellular automata is normally restricted to take values in a finite set of symbols $\mathcal{G}$. To reflect this, we will use symbol $s_i(t)$ to denote the value of the (discrete) state variable at site i at time t, that is,

$$s_i(t+1) = f\Big(s_{i-r}(t), s_{i-r+1}(t), \ldots, s_{i+r}(t)\Big). \tag{10}$$

In the case of binary cellular automata, which are the main focus of this chapter, the local function takes values in the set $\{0,1\}$, so that $f : \{0,1\} \rightarrow \{0,1\}^{2r+1}$. Binary rules of radius 1 are called elementary rules, and they are usually identified by their Wolfram number $W(f)$, defined as

$$W(f) = \sum_{x_1,x_2,x_3=0}^{1} f(x_1,x_2,x_3) 2^{(2^2 x_1 + 2^1 x_2 + 2^0 x_3)}. \tag{11}$$

In what follows, the set of symbols $\mathcal{G} = \{0,1,...N-1\}$ will be be called *a symbol set*, and by $\mathcal{S}$ we will denote the set of all bisequences over $\mathcal{G}$, where by a bisequence we mean a function on $\mathbb{Z}$ to $\mathcal{G}$. The set $\mathcal{S}$ will also be called the *configuration space*. Most of the time, we shall assume that $\mathcal{G} = \{0,1\}$, so that the configuration space will usually be a set of bi-infinite binary strings. Corresponding to f (also called *a local mapping*) we can define a *global mapping* $F : \mathcal{S} \rightarrow \mathcal{S}$ such that $(F(s))_i = f(s_{i-r}, \ldots, s_i, \ldots, s_{i+r})$ for any $s \in \mathcal{S}$.

Comparing (3) and (10), we conclude that r plays a similar role in CA as the degree of the highest derivative in PDEs. We should stress, however, that the PDE defined by (3) and the cellular automaton (10) obtained by the above "discretization procedure" have usually very little in common. There exist discretization schemes (such as ultradiscretization) which seem to preserve some features of the dynamics while passing from PDE to CA, but they are beyond the scope of this chapter. We merely want to indicate here that conceptually, cellular automata are closely related to PDEs, although in contrast to PDEs, all variables in CA are discrete. Moreover, dependent variable u is bounded in the case of CA – a restriction which is not normally imposed on the dependent variable of a PDE.

2.1 Deterministic initial value problem

For PDEs, an initial value problem (also called a Cauchy problem) is often considered. It is the problem of finding $u(x,t)$ for $t > 0$ subject to

$$\begin{aligned} u_t(x,t) &= F(u, u_x, u_{xx}, \ldots), \quad \text{for } x \in \mathbb{R},\, t > 0, \\ u(x,0) &= G(x) \quad \text{for } x \in \mathbb{R}, \end{aligned} \tag{12}$$

where the function $G : \mathbb{R} \to \mathbb{R}$ represents the given initial data. A similar problem can be formulated for cellular automata: given

$$\begin{aligned} s_i(t+1) &= f\Big(s_{i-r}(t), s_{i-r+1}(t), \ldots, s_{i+r}(t)\Big), \\ s_i(0) &= g(i), \end{aligned} \tag{13}$$

find $s_i(t)$ for $t > 0$, where the initial data is represented by the given function $g : \mathbb{Z} \to \mathcal{G}$.
Here comes the crucial difference between PDEs and CA. For the initial value problem (12), we can in some cases obtain an exact solution in the sense of a formula for $u(x,t)$ involving $G(x)$. To give a concrete example, consider the classical Burgers equation

$$u_t = u_{xx} + u u_x. \tag{14}$$

If $u(x,0) = G(x)$, one can show that for $t > 0$

$$u(x,t) = 2\frac{\partial}{\partial x} \ln \left\{ \frac{1}{\sqrt{4\pi t}} \int_{-\infty}^{\infty} \exp\left[-\frac{(x-\xi)^2}{4t} - \frac{1}{2}\int_0^{\xi} G(\xi')d\xi' \right] d\xi \right\}, \tag{15}$$

which means that, at least in principle, knowing the initial condition $u(x,0)$, we can compute $u(x,t)$ for any $t > 0$. In most cases, however, no such formula is known, and we have to resort to numerical PDE solvers, which bring in a whole set of problems related to their convergence, accuracy, stability, etc.
For cellular automata, obtaining an exact formula for the solution of the initial value problem (13) analogous to (15) is even harder than for PDEs. No such formula is actually known for virtually any non-trivial CA. At best, one can find the solution if $g(i)$ is simple enough. For example, for the majority rule defined in eq. (1), if $g(i) = i \bmod 2$, then one can show without difficulty that

$$s_i(t) = (i+t) \bmod 2, \tag{16}$$

but such result can hardly be called interesting.[1]

[1] By $a \bmod 2$ we mean the integer remainder of division of a by 2, which is 0 for even a and 1 for odd a.

Nevertheless, for CA, unlike for PDEs, it is very easy to find the value of $s_i(t)$ for any $i \in \mathbb{Z}$ and any $t \in \mathbb{N}$ by direct iteration of the cellular automaton equation (10). Thus, in algorithmic sense, problem (13) is always solvable – all one needs to do is to take the initial data $g(x)$ and perform t iterations. In contrast to this, initial value problem for PDE cannot be solved exactly by direct iteration – all we can usually do is to obtain some sort of numerical approximation of the solution.

2.2 Probabilistic initial value problem

Before we address basic simulation issues for CA, we need to present an alternative, and usually more useful, way of posing the initial value problem for CA. Often we are interested in a range of initial conditions, with a given probability distribution. How to handle the initial value problem in such cases? This is best achieved by using some basic concepts of probability theory, namely the notion of the cylinder set and the probability measure. A reader unfamiliar with these concepts can skip to the end of this subsection.

An appropriate mathematical description of an initial distribution of configurations is a probability measure μ on $\mathcal{S}$, where $\mathcal{S}$ is the set of all possible configurations, i.e., bi-infinite strings of symbols.

Such a measure can be formally constructed as follows. If b is a block of symbols of length k, that is, $b = b_0 b_1 \dots b_{k-1}$, then for $i \in \mathbb{Z}$ we define a cylinder set as

$$C_i(b) = \{s \in \mathcal{S} : s_i = b_0, s_{i+1} = b_1 \dots, s_{i+k-1} = b_{k-1}\}. \tag{17}$$

The cylinder set is thus a set of all possible configurations with fixed values at a finite number of sites. Intuitively, the measure of the cylinder set given by the block $b = b_0 \dots b_{k-1}$, denoted by $\mu[C_i(b)]$, is simply a probability of occurrence of the block b in a position starting at i. If the measure μ is shift-invariant, that is, $\mu(C_i(b))$ is independent of i, we can drop the index i and simply write $\mu(C(b))$.

The Kolmogorov consistency theorem states that every probability measure μ satisfying the consistency condition

$$\mu[C_i(b_1 \dots b_k)] = \sum_{a \in \mathcal{G}} \mu[C_i(b_1 \dots b_k, a)] \tag{18}$$

extends to a shift invariant measure on $\mathcal{S}$. For $p \in [0,1]$, the Bernoulli measure defined as $\mu_p[C(b)] = p^j(1-p)^{k-j}$, where j is a number of ones in b and $k-j$ is a number of zeros in b, is an example of such a shift-invariant (or spatially homogeneous) measure. It describes a set of random configurations with the probability that a given site is in state 1 equal to p.

Since a cellular automaton rule with global function F maps a configuration in $\mathcal{S}$ to another configuration in $\mathcal{S}$, we can define the action of F on measures on $\mathcal{S}$. For all measurable subsets E of $\mathcal{S}$ we define $(F\mu)(E) = \mu(F^{-1}(E))$, where $F^{-1}(E)$ is an inverse image of E under F.

The probabilistic initial value problem can thus be formulated as follows. If the initial configuration was specified by μ, what can be said about $F^t\mu$ (i.e., what is the probability measure after t iterations of F)?

Often in practical applications we do not need to know $F^t\mu$, but given a block b, we want to know what is the probability of the occurrence of this block in a configuration obtained from a random configuration (sampled, for example, according to the measure μ_p) after t iterations of a given rule. In the simplest case, when $b = 1$, we will define the *density of ones* as

$$\rho(t) = (F^n\mu_p)(C(1)). \tag{19}$$

3. Simulation problems

The basic problem in CA simulations is often called the forward problem: given the initial condition, that is, the state of cells of the lattice at time $t = 0$, what is their state at time $t > 0$? On the surface, this is an easy question to answer: just iterate the CA rule t of times, and you will find the answer. Nevertheless, upon closer inspection, some potential problems emerge.

- **Infinite lattice problem:** In the initial value problem (13), the initial condition is an infinite sequence of symbols. How do we compute an image of an infinite sequence under the cellular automaton rule?
- **Infinite time problem:** If we are interested in $t \to \infty$ behavior, as we usually do in dynamical system theory, how can simulations be used to investigate this?
- **Speed of simulations:** What if the lattice size and the number of iterations which interest us are both finite, but large, and the running time of the simulation is too long? Can we speed it up?

Some of these problems are interrelated, as will become clear in the forthcoming discussion.

4. Infinite lattice problem

In statistical and solid state physics, it is common to use the term "bulk properties", indicating properties of, for example, infinite crystal. Mathematically bulk properties are easier to handle, because we do not have to worry about what happens on the surface, and we can assume that the underlying physical system has a full translational invariance.
In simulations, infinite system size is of course impossible to achieve, and periodic boundary conditions are used instead. In cellular automata, similar idea can be employed. Suppose that we are interested in solving an initial value problem as defined by eq. (13). Obviously, it is impossible to implement direct simulation of this problem because the initial condition $s_i(0)$ consists of infinitely many cells. Once can, however, impose boundary condition such that $g(i + L) = g(i)$. This means that the initial configuration is periodic with period L. It is easy to show that if $s_i(0)$ is periodic with period L, then $s_i(t)$ is also periodic with the same period. This means than in practical simulations one needs to store only the values of cells with $i = 0, 1, 2, \ldots, L - 1$, that is, a finite string of symbols.
If one is interested in some bulk property P of the system being simulated, it is advisable to perform a series of simulations with increasing L, and check how P depends on L. Quite often, a clear trend can be spotted, that is, as L increases, P converges to the "bulk" value. One has to be aware, however, that some finite size effects are persistent, and may be present for any L, and disappear only on a truly infinite lattice.
Consider, as a simple example, the well-know rule 184, for which the local rule is defined as

$$\begin{aligned} &f(0,0,0) = 0,\ f(0,0,1) = 0,\ f(0,1,0) = 0,\ f(0,1,1) = 1, \\ &f(1,0,0) = 1,\ f(1,0,1) = 1,\ f(1,1,0) = 0,\ f(1,1,1) = 1, \end{aligned} \tag{20}$$

or equivalently

$$s_i(t+1) = s_{i-1}(t) + s_i(t)s_{i+1}(t) - s_{i-1}(t)s_i(t). \tag{21}$$

Suppose that as the initial condition we take

$$s_i(0) = i \bmod 2 + \delta_{i,0} - \delta_{i,1}, \tag{22}$$

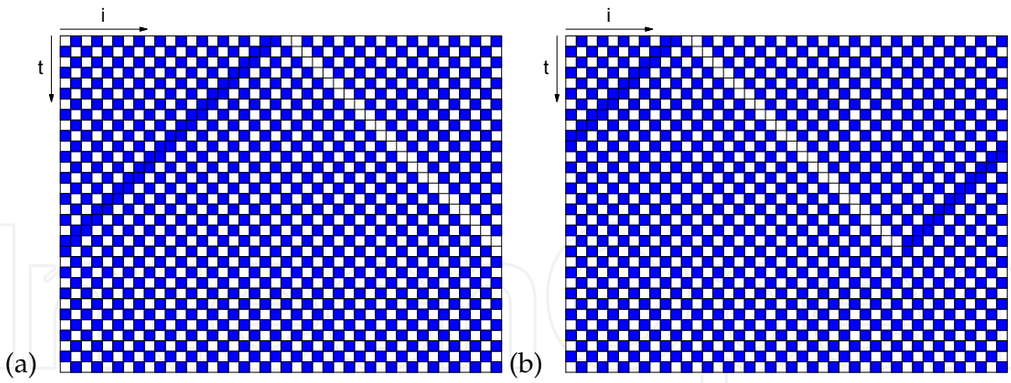


Fig. 2. Spread of defects in rule 184.

where $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise. One can show that under the rule 184, at time $t > 1$ we have the following solution of the initial value problem given by (21, 22),

$$s_i(t) = (i + t) \bmod 2 + \sum_{k=0}^{t} \delta_{-t+2k,i} - \sum_{k=0}^{t} \delta_{-t+2k+1,i}. \tag{23}$$

This can be verified by substituting (23) into (21). It is also straightforward to verify that in $s_i(t)$ a pair of adjacent ones is always present, since $s_{-t-1}(t) = s_{-t}(t) = 1$ for all $t \geq 0$, that is, at time t, sites with coordinates $i = -t - 1$ and $i = -t$ are in state 1.

Let us say that we are interested in the number of pairs 11 present in the configuration in the limit of $t \to \infty$, and we want to find it our by a direct simulation. Since the initial string $s_i(0)$ consists of 1100 preceded and followed by infinite repetition of 10, that is, $s_i(0) = \ldots 1010101100101010 \ldots$, we could take as the initial condition only part of this string corresponding to $i = -L$ to $L + 1$ for a given L, impose periodic boundary conditions on this string, and compute t consecutive iterations, taking t large enough, for example, $t = 2L$. It turns out that by doing this, we will always obtain a string which does not have any pair 11, no matter how large L we take! This is illustrated in Figure 2a, where first 30 iterations of a string given by eq. (22) with $i = -20$ to 21 and periodic boundary conditions is shown. Black squares represent 1, and white 0. The initial string ($t = 0$) is shown as the top line, and is followed by consecutive iterations, so that t increases in the downward direction. It is clear that we have two "defects", 11 and 00, moving in opposite direction. On a truly infinite lattice they would never meet, but on a periodic one they do meet and annihilate each other. This can be better observed in Figure 2b, in which the location of the initial defect has been shifted to the left, so that the annihilation occurs in the interior of the picture.

The disappearance of 11 (and 00) is an artifact of periodic boundaries, which cannot be eliminated by increasing the lattice size. This illustrates that extreme caution must be exercised when we draw conclusions about the behaviour of CA rules on infinite lattices from simulations of finite systems.

5. Infinite time problem

In various papers on CA simulations one often finds a statement that simulations are to be performed for long enough time so that the system "settles into equilibrium". The notion of

equilibrium is rarely precisely defined, and this is somewhat understandable: mathematically, the definition of the "equilibrium" in spatially extended dynamical system such as CA is far from obvious.

The idea of the "equilibrium" can be better understood if we note that it has its roots in dynamical systems theory. Consider a difference equation $x_{n+1} = f(x_n)$, where f is a given function. Point a which has the property $f(a) = a$ is called a fixed point. In many applications and mathematical models, the prevailing type of fixed points is the so-called hyperbolic fixed point, satisfying $|f'(a)| \neq 1$. Suppose that a is hyperbolic with $|f'(a)| < 1$. One can then show (Devaney, 1989) that for every initial value x_0 in some open interval around a, if we iterate $x_{n+1} = f(x_n)$ starting from that initial value, we have $\lim_{n\to\infty} x_n = a$, and that there exists nonnegative $A < 1$ such that

$$|x_n - a| \leq A^n |x_0 - a|. \tag{24}$$

This indicates that x_n approaches the fixed point *a exponentially fast* (or faster). It turns out that the hyperbolicity is a very common feature of dynamical systems, even in higher dimensions, and that exponential approach to a fixed point is also a very typical behavior.

In cellular automata, the notion of the fixed point can be formulated in the framework of both deterministic and probabilistic initial value problem. The deterministic fixed point of a given CA rule f is simply a configuration s such that the if the rule f is applied to s, one obtains the same configuration s. Deterministic fixed points in CA are common, but they are usually not that interesting.

A far more interesting is the fixed point in the probabilistic sense, that is, a probability measure μ which remains unchanged after the application of the rule. Using a more practical language, we often are interested in a probability of some block of symbols, and want to know how this probability changes with t. In the simplest case, this could be the probability of occurrence of 1 in a configuration at time t, which is often referred to as the density of ones and denoted by $\rho(t)$. In surprisingly many cases, such probability tends to converge to some fixed value, and the convergence is of the hyperbolic type.

Consider, as an example, a rule which has been introduced in connection with models of the spread of innovations (Fukś and Boccara, 1998). Let us assume that, similarly as in the majority voting rule, we have a group of individuals arranged on an infinite line. Each individual can be in one of two states: adopter (1) and non-adopter (0). Suppose that each time step, each non-adopter becomes adopter if and only if exactly one of his neighbours is an adopter. Once somebody becomes adopter, he stays in this state forever. Local function for this rule is defined by $f(1,0,1) = f(0,0,0) = 0$, and $f(x_0, x_1, x_2) = 1$ otherwise.

Let us say that we are interested in the density of ones, that is, the density of adopters at time t, to be denoted by $\rho(t)$. Assume that we start with a random initial configuration with the initial density of adopters equal to $p < 1$. This defines a probabilistic initial value problem with an initial Bernoulli measure μ_p. In (Fukś and Boccara, 1998) it has been demonstrated that

$$\rho(t) = 1 - p^2(1-p) - p\frac{(1-p)^3}{2-p} - \left[1 - p^2 + \frac{p(1-p)^2}{p-2}\right](1-p)^{2t+1}, \tag{25}$$

and

$$\lim_{t\to\infty} \rho(t) = 1 - p^2(1-p) - p\frac{(1-p)^3}{2-p}. \tag{26}$$

It is now clear that $\rho(\infty) - \rho(t) \sim A^t$, where $A = (1-p)^2$, meaning that $\rho(t)$ approaches $\rho(\infty)$ exponentially fast, similarly as in the case of hyperbolic fixed point discussed above. If

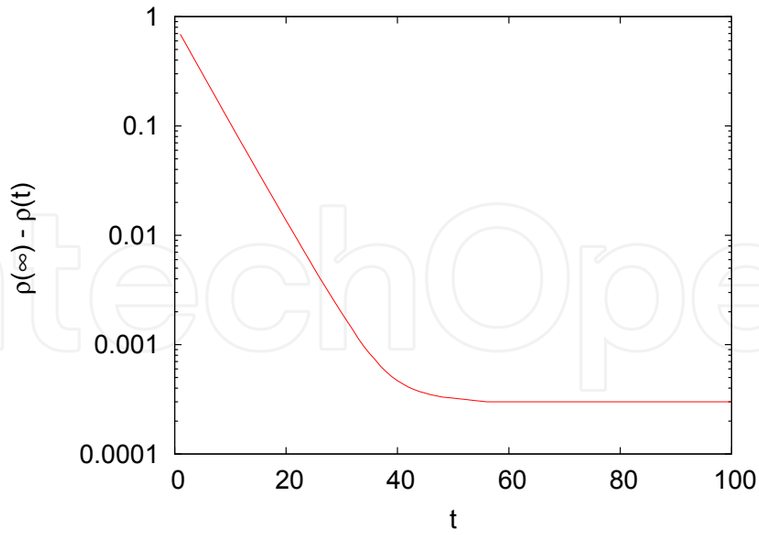


Fig. 3. Plot of $\rho(\infty) - \rho(t)$ as a function of time for simulation of rule 222 on a lattice of length 50000 starting from randomly generated initial condition with initial density $p = 0.1$.

we now simulate this rule on a finite lattice, $\rho(t)$ will initially exponentially decrease toward $\rho(\infty)$, so that $\rho(\infty) - \rho(t)$ plotted as a function of t in a semi-log graph will follow a straight line, and then, due to finite size of the lattice, it will level off, as shown in Figure 3. This plateau level is usually referred to as an "equilibrium". Note that the plateau is reached very quickly (in less then 60 iterations in this example), but it corresponds to a value of $\rho(t)$ slightly below the "true" value of $\rho(\infty)$. This is an extremely common behavior encountered in many CA simulations: we reach the plateau very quickly, but it is slightly off the "true" value. A proper way to proceed, therefore, is to record the value of the plateau for many different (and increasing) lattice sizes L, and from there to determine what happens to the plateau in the limit of $L \to \infty$. This is a much more prudent approach than simply "iterating until we reach the equilibrium", as it is too often done.

Of course, in addition to the behaviour analogous to the convergence toward a hyperbolic fixed point, we also encounter in CA another type of behaviour, which resembles a non-hyperbolic dynamics. Again, in order to understand this better, let us start with one-dimensional dynamical system. This time, consider the logistic difference equation

$$x_{t+1} = \lambda x_t (1 - x_t), \tag{27}$$

where λ is a given parameter $\lambda \in (0, 2)$. This equation has two fixed points $x^{(1)} = 0$ and $x^{(2)} = 1 - 1/\lambda$. For any initial value $x_0 \in (0, 1)$, when $\lambda < 1$, x_t converges to the first of these fixed points, that is, $x_t \to 0$, and otherwise it converges to the second one, so that $x_t \to 1 - 1/\lambda$ as $t \to \infty$.

In the theory of iterations of complex analytic functions, it is possible to obtain an approximate expressions describing the behavior of x_t near the fixed point as $t \to \infty$. This is done using a method which we will not describe here, by conjugating the difference equation with an appropriate Möbius transformation, which moves the fixed point to ∞. Applying this method,

one obtains formulas for the asymptotic behavior of x_t which can be summarized as follows:

$$x_t - x_\infty \sim \begin{cases} \lambda^t & \text{if } \lambda < \lambda_c, \\ 1/t & \text{if } \lambda = \lambda_c, \\ (2-\lambda)^t & \text{if } \lambda > \lambda_c, \end{cases} \tag{28}$$

where $\lambda_c = 1$. We can see that the approach toward the fixed point is exponential if $\lambda \neq 1$, and that it slows down as λ is getting closer to 1. At $\lambda = 1$, the decay toward the fixed point is not exponential, but takes a form of a power law, indicating that the fixed point is non-hyperbolic. This phenomenon, which is called a *transcritical bifurcation*, has a close analog in dynamics of cellular automata. In order to illustrate this, consider the following problem. Let s be an infinite string of binary symbols, i.e., $s = \ldots s_{-1}s_0s_1\ldots$. We will say that the symbol s_i has a *dissenting right neighbour* if $s_{i-1} = s_i \neq s_{i+1}$. By *flipping* a given symbol s_i we will mean replacing it by $1 - s_i$. Suppose that we simultaneously flip all symbols which have dissenting right neighbours, as shown in the example below.

$$\begin{array}{ccccccccccccccc} \cdots & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & \cdots \\ & & \downarrow & & & \downarrow & & & \downarrow & & & & & \\ \cdots & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & \cdots \end{array} \tag{29}$$

Assuming that the initial string is randomly generated, what is the probability $P_{dis}(t)$ that a given symbol has a dissenting right neighbour after t iterations of the aforementioned procedure?
It is easy to see that the process we have described is nothing else but a cellular automaton rule 142, with the following local function

$$\begin{aligned} f(0,0,0) = 0, f(0,0,1) = 1, f(0,1,0) = 1, f(0,1,1) = 1, \\ f(1,0,0) = 0, f(1,0,1) = 0, f(1,1,0) = 0, f(1,1,1) = 1, \end{aligned} \tag{30}$$

which can also be written in an algebraic form

$$f(x_0, x_1, x_2) = x_1 + (1 - x_0)(1 - x_1)x_2 - x_0x_1(1 - x_2). \tag{31}$$

In (Fukś, 2006), it has been demonstrated that the desired probability $P_{dis}(t)$ is given by

$$P_{\text{dis}}(t) = 1 - 2q - \sum_{j=1}^{t+1} \frac{j}{t+1} \binom{2t+2}{t+1-j} (2q)^{t+1-j} (1-2q)^{t+1+j}, \tag{32}$$

where $q \in [0, 1/2]$ is the probability of occurrence of the block 10 in the initial string. It is possible to show that in the limit of $t \to \infty$

$$\lim_{t\to\infty} P_{\text{dis}}(t) = \begin{cases} 2q & \text{if } q < 1/4, \\ 1 - 2q & \text{otherwise}, \end{cases} \tag{33}$$

indicating that $q = 1/4$ is a special value separating two distinct "phases".
Suppose now that we perform a simulation of rule 142 as follows. We take a finite lattice with randomly generated initial condition in which the proportion of blocks 10 among all blocks of length 2 is equal to q. We iterate rule 142 starting from this initial condition t times, and we count what is the proportion of sites having dissenting neighbours. If we did this, we would

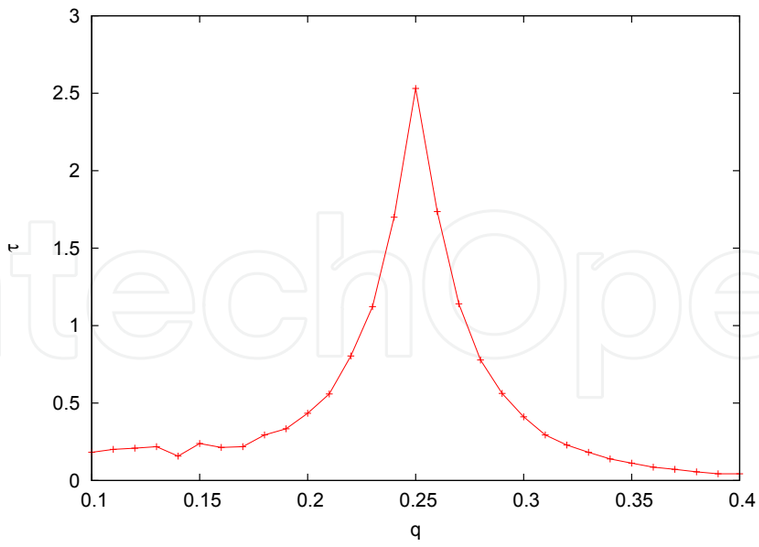


Fig. 4. Decay time as a function of q for rule 142.

observe that most of the time the number of dissenting neighbours behaves similarly to what we have seen in rule 222 (Figure 3), that is, quickly reaches a plateau. However, when q gets closer and closer to $1/4$, it takes longer and longer to get to this plateau.
A good way to illustrate this phenomenon is by introducing a quantity which can be called a decay time,

$$\tau = \sum_{t=0}^{\infty} |P_{dis}(t) - P_{dis}(\infty)|. \tag{34}$$

Decay time will be finite if $P_{dis}(t)$ decays exponentially toward $P_{dis}(\infty)$, and will become infinite when the decay is non-exponential (of power-law type). One can approximate τ in simulations by truncating the infinite sum at some large value t_{max}, and by using the fraction of sites with dissenting neighbours in place of $P_{dis}(t)$. An example of a plot of τ obtained this way as a function of q for simulations of rule 142 is show in Figure 4. We can see that around $q = 1/4$, the decay shows a sign of divergence, and indeed for larger values of t_{max} the height of the peak at $q = 1/4$ would increase. Such divergence of the decay indicates that at $q = 1/4$ the convergence to "equilibrium" is no longer exponential, but rather is analogous to an approach to a non-hyperbolic fixed point. In fact, it has been demonstrated (Fukś, 2006) that at $q = 1/4$ we have $|P_{dis}(t) - P_{dis}(\infty)| \sim t^{-1/2}$. Similar divergence of the decay time is often encountered in statistical physics in systems exhibiting a phase transition, and it is sometimes called "critical slowing down".
This example yields the following practical implications for CA simulations. If the CA rule or initial conditions which one investigates depend on a parameter which can be varied continuously, it is worthwhile to plot the decay time as a function of this parameter to check for signs of critical slowing down. Even if there is no critical slowing down, the decay time is an useful indicator of the rate of convergence to "equilibrium" for different values of the parameter, and it helps to discover non-hyperbolic behavior.
One should also note at this point that in some cellular automata, decay toward the "equilibrium" can be much slower than in the $q = 1/4$ case in the above example. This is usually

the case when the rule exhibits complex propagating spatiotemporal structures. For example, it has been found that in rule 54, the number of some particle-like spatiotemporal structures tends to zero as approximately $t^{-0.15}$ (Boccara et al., 1991). In cases like this, simulations are computationally very expensive, that is, very large number of iterations is required in order to get sufficiently close to the "equilibrium".

6. Improving simulation performance

If the decay toward the equilibrium is non-hyperbolic, the speed of simulations becomes an issue. The speed of simulations can be improved either in software, or by using a specialized hardware. We will discuss only selected software methods here, only mentioning that specialized CA hardware has been build in the past (for example, CAM-6 and CAM-8 machines, field programmable gate arrays), and some novel hardware schemes have been proposed or speculated upon (implementation of CA using biological computation, nanotechnology, or quantum devices). Additionally, CA simulations are very suitable for parallel computing environments – a topic which deserves a separate review, and therefore will not be discussed here.

Typically, CA simulations are done using integer arrays to store the values of $s_i(t)$. A basic and well-know method for computing $s_i(t+1)$ knowing $s_i(t)$ is to use a lookup table – that is, table of values of the local function f for all possible configurations of the neighbourhood. Two arrays are needed, one for $s_i(t)$ (let us call it "s") and one for $s_i(t+1)$ ("snew"). Once "s" is computed using the lookup table, one only needs to swap pointers to "s" and "snew". This avoids copying of the content of "snew" to "s", which would decrease simulation performance.

Beyond these basic ideas, there are some additional methods which can improve the speed of simulations, and they will be discussed in the following subsections.

6.1 Self-composition

If f and g are CA rules of radius 1, we can define a composition of f and g as

$$(f \circ g)(x_0, x_1, x_2, x_3, x_4) = f(g(x_0, x_1, x_2), g(x_1, x_2, x_3), g(x_2, x_3, x_4)). \tag{35}$$

Similar definition can be given for rules of higher radius. If $f = g$, $f \circ f$ will be called a self-composition. Multiple composition will be denoted by

$$f^n = \underbrace{f \circ f \circ \cdots \circ f}_{n \text{ times}}. \tag{36}$$

The self-composition can be used to speed-up CA simulations. Suppose that we need to iterate a given CA t times. We can do it by iterating f t times, or by first computing f^2 and iterate it $t/2$ times. In general, we can compute f^n first, and then perform t/n iterations of f^n, and in the end we will obtain the same final configuration. Obviously, this decreases the number of iterations and, therefore, can potentially speed-up simulations. There is, however, some price to pay: we need to compute f^n first, which also takes some time, and, moreover, iterating f^n will be somewhat slower due to the fact that f^n has a longer lookup table than f. It turns out that in practice the self-composition is advantageous if n is not too large. Let us denote by T_n the running time of the simulation performed using f^n as described above. The ratio T_n/T_1 (to be called a speedup factor) as a function of n for simulation of rule 18 on a lattice of 10^6 sites is shown in Figure 5. As one can see, the optimal value of n appears to be 7, which

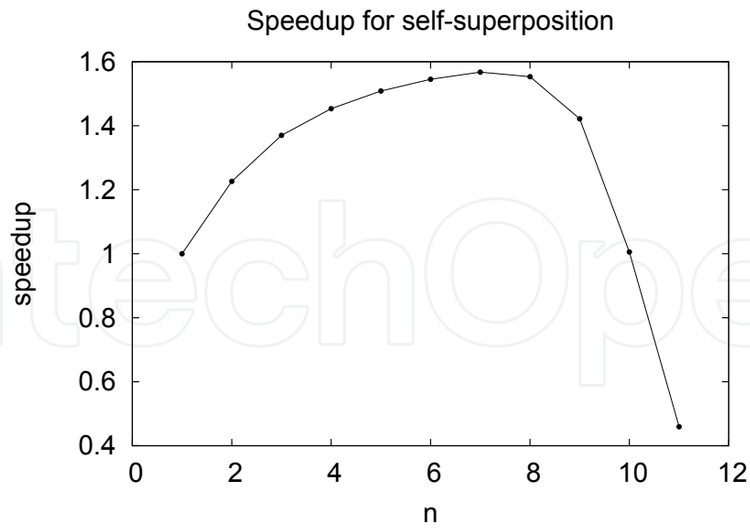


Fig. 5. Plot of the speedup factor T_n/T_1 as a function of n for self-composition of rule 18.

speeds up the simulation by 58%. Increasing n beyond this value does not bring any further improvement – on the contrary, the speedup factor rapidly decreases beyond $n = 7$, and at $n = 10$ we actually see a significant *decrease* of the running time compared to $n = 1$.
The optimal value of $n = 7$ is, of course, not universal, and for different simulations and different rules it needs to be determined experimentally. Self-composition is especially valuable if one needs to iterate the same rule over a large number of initial conditions. In such cases, it pays off to determine the optimal n first, then pre-compute f^n, and use it in all subsequent simulations.

6.2 Euler versus Langrage representation

In some cases, CA may posses so-called additive invariants, and these can be exploited to speed-up simulations. Consider again the rule 184 defined in eq (20) and (21). The above definition can also be written in a form

$$s_i(t+1) = s_i(t) + J(s_{i-1}(t), s_i(t)) - J(s_i(t), s_{i+1}(t)), \tag{37}$$

where $J(x_1, x_2) = x_1(1 - x_2)$. Clearly, for a periodic lattice with N sites, when the above equation is summed over $i = 1 \ldots N$, one obtains

$$\sum_{i=1}^{N} s_i(t+1) = \sum_{i=1}^{N} s_i(t), \tag{38}$$

due to cancellation of terms involving J. This means that the number of sites in state 1 (often called "occupied" sites) is constant, and does not change with time. This is an example of additive invariant of 0-th order. Rules having this property are often called number-conserving rules.
Since the number-conserving CA rules conserve the number of occupied sites, we can label each occupied site (or "particle") with an integer $n \in \mathbb{Z}$, such that the closest particle to the

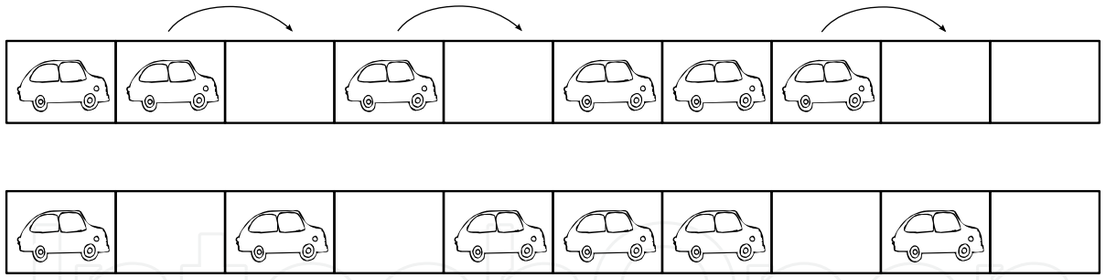


Fig. 6. Motion representation of rule 184 interpreted as a simplified road traffic model. The upper row represents configuration of cars/particles at time t, and the lower row at time $t+1$.

right of particle n is labeled $n+1$. If $y_n(t)$ denotes the position of particle n at time t, we can then specify how the position of the particle at the time step $t+1$ depends on positions of the particle and its neighbours at the time step t. For rule 184 one obtains

$$y_n(t+1) = y_n(t) + \min\{y_{n+1}(t) - y_n(t) - 1, 1\}. \tag{39}$$

This can be described in very simple terms: if a "particle" is followed by an empty site, it moves to the right, otherwise it stays in the same place. This is illustrated in Figure 6, where "particles" are depicted as "cars". Although this can hardly be called a realistic traffic flow model, it nevertheless exhibits some realistic features and can be further extended (Chowdhury et al., 2000).

Equation (39) is sometimes referred to as the motion representation. For arbitrary CA rule with additive invariant it is possible to obtain the motion representation by employing an algorithm described in (Fukś, 2000). The motion representation is analogous to Lagrange representation of the fluid flow, in which we observe individual particles and follow their trajectories. On the other hand, eq. (37) could be called Euler representation, because it describes the process at a fixed point in space (Matsukidaira and Nishinari, 2003).

Since the number of "particles" is normally smaller than the number of lattice sites, it is often advantageous to use the motion representation in simulations. Let ρ be the density of particles, that is, the ratio of the number of occupied sites and the total number of lattice sites. Furthermore, let T_L and T_E be, respectively, the execution time of one iteration for Lagrange and Euler representation. A typical plot of the speedup factor T_E/T_L as a function of ρ for rule 184 is shown in Figure 7. One can see that the speedup factor is quite close to $1/\rho$, shown as a continuous line in Figure 7. For small densities of occupied sites, the speedup factor can be very significant.

6.3 Bitwise operators

One well known way of increasing CA simulation performance is the use of bitwise operators. Typically, CA simulations are done using arrays of integers to store states of individual cells. In computer memory, integers are stored as arrays of bits, typically 32 or 64 bits long (to be called *MAXBIT*). If one wants to simulate a binary rule, each bit of an integer word can be set independently, thus allowing to store *MAXBIT* bit arrays in a single integer array.

In C/C++, one can set and read individual bits by using the following macros:

```
#define GETBIT(x, n) ((x>>n)&1)
#define SETBIT(x, n) (x | (1<<n))
```

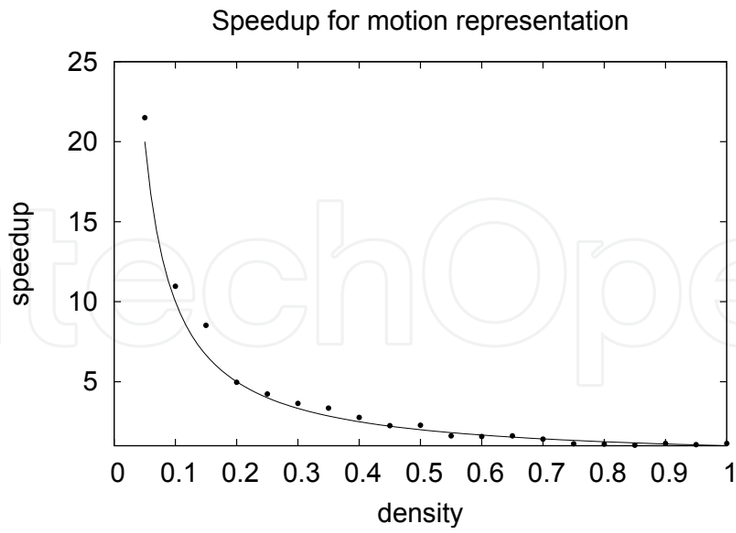


Fig. 7. Plot of the speedup factor T_E/T_L as a function of density ρ for rule 184.

```
#define ZEROBIT(x,n) (x & (~(1<<n)))
```

For example, if x is an integer variable, then

```
x=SETBIT(x,5)
x=ZEROBIT(x,4)
```

sets respectively the 5-th bit of x to value 1 and the 4-th bit to 0. Using these macros, we can, for instance, prepare *MAXBIT* different initial conditions, and store them in one integer array. The main advantage of such bit packing lies in the fact that we can then use bitwise operators to perform operations on all bits *simultaneously*. Consider, as an example, rule 184 again, as defined in eq. (20). It is easy to check that the local function for this rule can be written using logical "and" ($\wedge$) and "or" ($\vee$) operators,

$$f(x_0, x_1, x_2) = (x_0 \wedge \overline{x}_1) \vee (x_1 \wedge x_2), \tag{40}$$

where $\overline{x}$ denotes the negation of x. In C/C++ program, we can apply this rule to all BITMAX bits of the array as follows:

```
for(i=0; i<L; i++)
    snew[i]=(s[i-1] & ~s[i]) | (s[i] & s[i+1])
```

where L is the length of the lattice, s[i] denotes the state of site i, and snew[i] is the new state of this site. This method can potentially speed up simulations BITMAX times.

7. HCELL CA simulation library

Many software tools exist which are suitable for CA simulations, ranging from very specialized ones to very broad simulation environments. Partial list of these programs can be found in (Ilachinski, 2001). Most of these packages are great for "visual" explorations of CA, but they usually miss functions which a more mathematically inclined user would require.

For speed and flexibility in simulations, it is usually best to write the simulation in a compiled language, such as C/C++. For one dimensional cellular automata, the author developed a C++ library named HCELL, which is freely available under GNU Public License (Fukś, 2005). It allows fast simulation of deterministic and probabilistic CA, supports Wolfram rule numbering, four basic symmetry transformations, composition of rules (using operator overloading), totalistic rules, minimal rules etc. Spatial set entropy and spatial measure entropy can be computed for a given lattice configuration, assignment operator can be used for rules and lattices, rules can be tested for additive invariants, currents of additive quantities are computed in a symbolic form. Although this is not a visualization package, spatiotemporal patterns can be exported as EPS and Asymptote files.
HCELL allows to create two types of objects, *CA rules* and *lattices*. These objects can be manipulated using member functions and operators, and rules can be applied to lattices. Various global functions taking as arguments rules and lattices are also defined. In order to illustrate the use of HCELL in practical simulations, we will give some examples.

7.1 Example 1: the parity problem

As a first example, consider solution of the so-called parity problem for CA. For a finite binary string s, parity of s equals to 1 if the number of ones in the string is odd, otherwise it is zero. The parity problem is to find a CA which, when iterated starting from s and using periodic boundary conditions, will converge to all 1's if the parity of s is 1, otherwise it will converge to all 0's. Although it has been proven that there does not exist a single CA which would perform this task, in (Lee et al., 2001) a solution involving more than one rule has been found. For instance, if the lattice size is $L = 2q$ and q is odd, then applying the operator

$$G = F_{254}^{\lfloor L/2 \rfloor} F_{76}(F_{132}^{\lfloor L/2 \rfloor} F_{222}^{\lfloor L/2 \rfloor})^{\lfloor L/2 \rfloor} \tag{41}$$

to s produces a configuration with the desired property, that is, consisting of all 0's if the parity of s is 0, and consisting of all 1's otherwise. In the above, F_m denotes the global function associated with elementary cellular automaton rule with Wolfram number m. The following C++ program utilizing HCELL generates a random string or length 90, applies G to it, and prints the resulting configuration:

```
001 #include <hcell/hcell.h>
002 #include <iostream>
003 #include <math.h>
004 using namespace std;
005
006 int main()
007 {
008  //define needed rules
009  CArule1d r254(1,254), r76(1,76), r132(1,132), r222(1,222);
010  //create lattice
011  int L=90;  Lattice lat(L);
012  //initialize RNG
013  r250_init(1234);
014  //seed lattice randomly to get exactly 10 % sites occupied
015  lat.PreciseSeed(0.1);
016  int i, j;
017 for(j=0; j<L/2; j++)
018  {
019   for(i=0; i<L/2 ; i++) Evolve(&lat, r222);
```

```
020   for(i=0; i<L/2 ; i++) Evolve(&lat, r132);
021  }
022  Evolve(&lat, r76);
023  for(i=0; i<L/2 ; i++) Evolve(&lat, r254);
024  //print the resulting configuration
025  lat.Print();
026
027 return 0;
028 }
```

As we can see, all required operations are encapsulated. Elementary cellular automata rules are constructed by the statement `CArule1d r254(1,254)`, which is equivalent to "construct 1D CA rule with radius 1 and rule number 254, and name it r254". Applying a given rule to a lattice requires only a simple call to "Evolve" function, as shown in lines 19, 20 and 23. Figure 8 shows an example of a spatiotemporal patterns generated from five different initial conditions by applying the operator given by eq. (41). This figure has been produced by `ca2asy` tool included with HCELL. As expected, configurations with the parity 1 converge to all ones, and those with the parity 0 converge to all zeros.

7.2 Example 2: operations on rules

HCELL also allows various operations on rules. One can, in particular, construct composition of rules by using the overloaded $\star$ operator. For example, in order to verify that

$$f_{60} \circ f_{43} = f_{184} \circ f_{60}, \tag{42}$$

one only needs to check that the Hamming distance between rule tables $f_{60} \circ f_{43}$ and $f_{184} \circ f_{60}$ is zero, which in HCELL is as simple as

```
CArule1d r60(1,60), r184(1,184), r43(1,43);
cout << Hamming(r60*r43, r184*r60);
```

In addition to the composition, two other operators are overloaded as well: $+$ can be used to add rules (mod 2), and $=$ can be used as an assignment operator for both rules and lattices. This allows to write formulas for rules in a natural fashion. For example, one could define a new rule $f = f_{60} \circ f_{43} \oplus f_{184} \circ f_{60}$, where $\oplus$ represents mod 2 addition. Using HCELL, this could be achieved as

```
CArule1d r60(1,60), r184(1,184), r43(1,43), r;
r=r60*r43 + r184*r60;
```

One can also obtain transformed rules very easily, by calling appropriate member functions of `CArule1d`, for example,

```
rr=r.Reflected(void);
rc=r.Conjugated(void);
```

This constructs, respectively, the spatially reflected rule `rr` and the conjugated rule `rc`, where by Boolean conjugation we mean the interchange of 0's and 1's in the rule table.

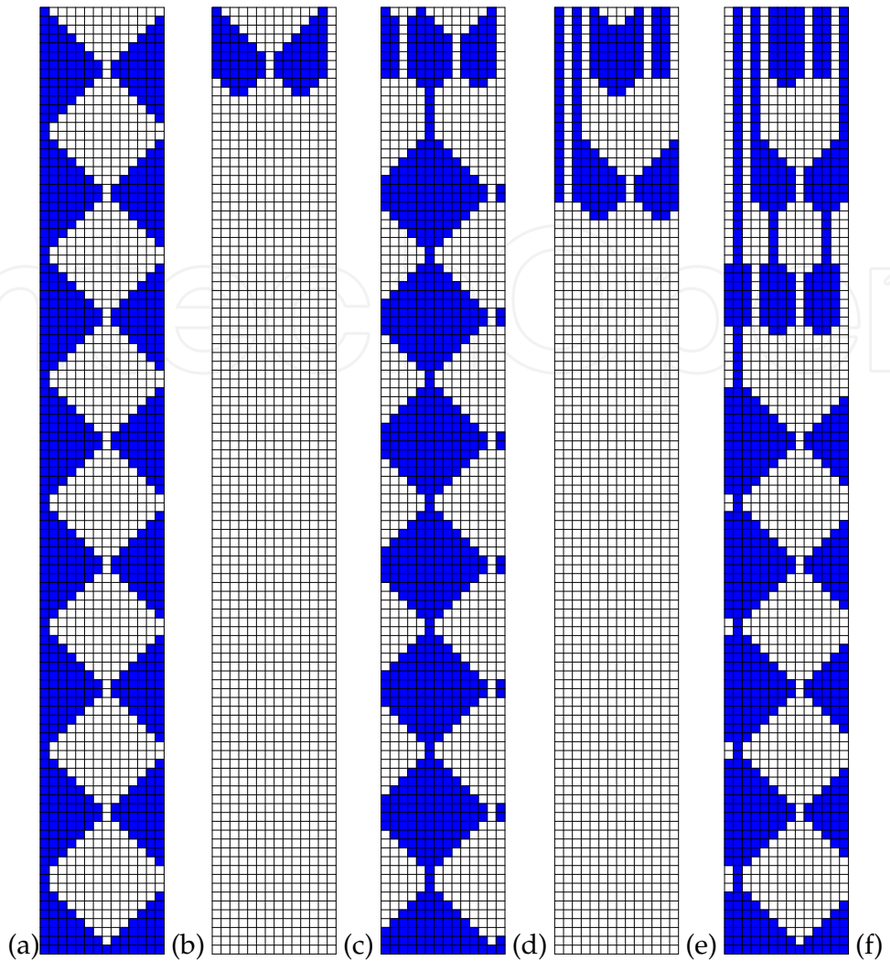


Fig. 8. Parity problem solution for lattice of length 14 and initial configuration with 1 (a), 2 (b), 3 (c), 6 (d) and 9 (e) sites in state 1.

7.3 Example 3: probabilistic density classification

In addition to these and many other capabilities for deterministic CA, HCELL also supplies a basic functionality for handling of probabilistic rules. To demonstrate this, let us denote by

$$P(s_i(t+1))|s_{i-1}(t), s_i(k), s_{i+1}(t)) \tag{43}$$

the probability that the site $s_i(t)$ with nearest neighbors $s_{i-1}(k), s_{i+1}(k)$ changes its state to $s_i(t+1)$ in a single time step. Consider a special probabilistic CA rule which is known to solve the so-called probabilistic density classification problem. If this rule is iterated starting from some initial string, the probability that all sites become eventually occupied is equal to the density of occupied sites in the initial string (Fukś, 2002). The following set of transition probabilities defines the aforementioned CA rule:

$$\begin{aligned} &P(1|0,0,0) = 0 \quad P(1|0,0,1) = p \quad P(1|0,1,0) = 1-2p \quad P(1|0,1,1) = 1-p \\ &P(1|1,0,0) = p \quad P(1|1,0,1) = 2p \quad P(1|1,1,0) = 1-p \quad P(1|1,1,1) = 1, \end{aligned} \tag{44}$$

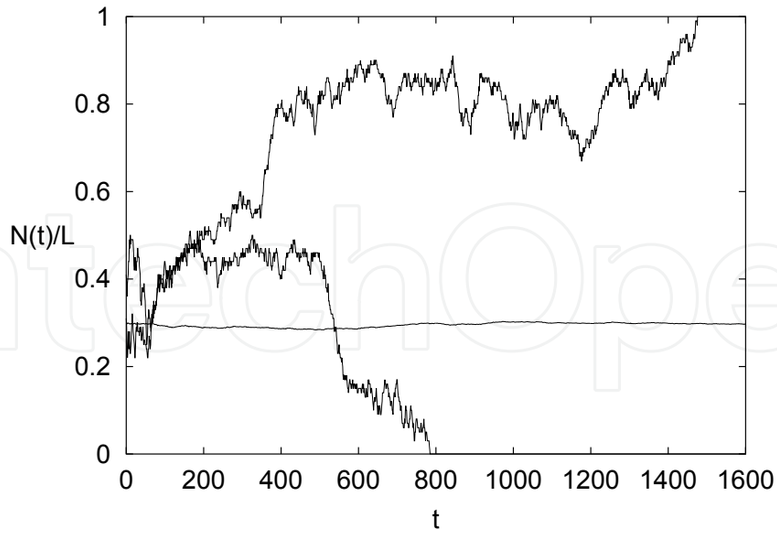


Fig. 9. Fraction of sites in state 1 $N(t)/L$ as a function of time t for two sample trajectories starting from initial configurations with $N(0) = 30$, $L = 100$, $p = 0.3$. The third line (almost horizontal) represents average of 1000 trajectories.

where $p \in (0, 1/2]$. The remaining eight transition probabilities can be obtained using $P(0|a,b,c) = 1 - P(1|a,b,c)$ for $a,b,c \in \{0,1\}$. In HCELL, this rule can be defined as

```
ProbabilisticRule1d r(1); //rule of radius 1
double p=0.25;
r.lookuptable[0]=0.0;
r.lookuptable[1]=p;
r.lookuptable[2]=1.0-2.0*p;
r.lookuptable[3]=1.0-p;
r.lookuptable[4]=p;
r.lookuptable[5]=2.0*p;
r.lookuptable[6]=1.0-p;
r.lookuptable[7]=1.0;
```

where we choose $p = 0.25$. The consecutive entries in the "lookuptable" are simply probabilities of eq. (44), in order of increasing neighbourhood code. Once the probabilistic rule is defined, it can be applied to a lattice `lat` by invoking the function `Evolve(&lat,r)`, just as in the case of deterministic rules. All the required calls to the random number generator are encapsulated and invisible to the user. Figure 9 shows two sample plots of $N(t)/L$ as a function of time, where $N(t)$ is the number of sites in state 1 and L is the length of the lattice. The value of $N(t)/L$ has been obtained by using the member function `Lattice.Density()` at every time step. The third line in Figure 9 represents an average of 1000 realizations of the process, and one can clearly see that indeed the probability of being in state 1 for any site is equal to the fraction of "occupied" sites in the initial condition, which is 0.3 in the illustrated case. These curves can be generated by HCELL in less than 20 line of code.
It is worthwhile to mention at this point that in HCELL, probabilistic rules can also be reconstructed from lattices. Suppose that `lat1` is the lattice state at time t, and `lat2` at time $t+1$,

where `lat2` has been obtained from `lat1` by applying an unknown rule r. Rule r can then be reconstructed by calling a special form of the constructor of probabilistic rules,

```
ProbabilisticRule1d r(&la1, &lat2, rad);
```

where `rad` is the desired rule radius. Of course, such reconstruction will only be approximate, and the accuracy of reconstruction will increase with the lattice size.
Many other features of HCELL, not mentioned here, are described in the documentation and examples which accompany source files. New version is currently under development. It will remove some limitations of the existing version. In particular, support for non-binary rules and numbering schemes for arbitrary rules is planned (current version supports Wolfram numbering for rules up to radius 2).

8. Conclusions

We presented an overview of basic issues associated with CA simulations, concentrating on selected problems which, in the mind of the author, deserve closer attention. We also demonstrated how HCELL can be used to perform some typical CA simulation tasks.
Obviously, many important topics have been omitted. In particular, the issue of dimensionality of space has not been addressed, and yet many important CA models require 2D, 3D, and higher dimensional lattices. Some collective phenomena in CA can only occur in high dimensions, and in terms of mathematical theory, dimensions higher than one often require quite different approach than 1D case. The reader is advised to consult the literature mentioned in the introduction to learn more about these topics.

9. References

Stefania Bandini, Bastien Chopard, and Marco Tomassini, editors. *Cellular Automata, 5th International Conference on Cellular Automata for Research and Industry, ACRI 2002, Geneva, Switzerland, October 9-11, 2002, Proceedings*, volume 2493 of *Lecture Notes in Computer Science*, 2002. Springer.

N. Boccara, J. Nasser, and M. Roger. Particle-like structures and interactions in spatio-temporal patterns generated by one-dimensional determinsitic cellular automaton rules. *Phys. Rev. A*, 44, July 1991.

B. Chopard and M. Drozd. *Cellular automata modeling of physical systems*. Cambridge UP, Cambridge, 1998.

D. Chowdhury, L. Santen, and A. Schadschneider. Statistical physics of vehicular traffic and some related systems. *Physics Reports*, 329:199–329, 2000.

R. L. Devaney. *An introduction to chaotic dynamical systems*. Addison-Wesley, Reading, MA, 1989.

H. Fukś. HCELL library for cellular automata simulations, 2005. URL `http://hcell.sourceforge.net`.

H. Fukś. A class of cellular automata equivalent to deterministic particle systems. In A. T. Lawniczak S. Feng and R. S. Varadhan, editors, *Hydrodynamic Limits and Related Topics*, Fields Institute Communications Series, Providence, RI, 2000. AMS.

H. Fukś. Non-deterministic density classification with diffusive probabilistic cellular automata. *Phys. Rev. E*, 66:066106, 2002.

H. Fukś. Dynamics of the cellular automaton rule 142. *Complex Systems*, 16:123–138, 2006.

H. Fukś and N. Boccara. Modelling diffusion of innovations with probabilistic cellular automata. In M. Delorme and J. Mazoyer, editors, *Cellular Automata: A Parallel Model*. Kluwer, Dordrecht, 1998.

Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific, Singapore, 2001.

J. Kari. Theory of cellular automata: a survey. *Theor. Comput. Sci.*, 334:3–33, 2005.

K. M. Lee, Hao Xu, and H. F. Chau. Parity problem with a cellular automaton solution. *Phys. Rev. E*, 64:026702, 2001.

J. Matsukidaira and K. Nishinari. Euler-Lagrange correspondence of cellular automaton for traffic-flow models. *Phys. Rev. Lett.*, 90:art. no.–088701, 2003.

Peter M. A. Sloot, Bastien Chopard, and Alfons G. Hoekstra, editors. *Cellular Automata, 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004*, volume 3305 of *Lecture Notes in Computer Science*, 2004. Springer.

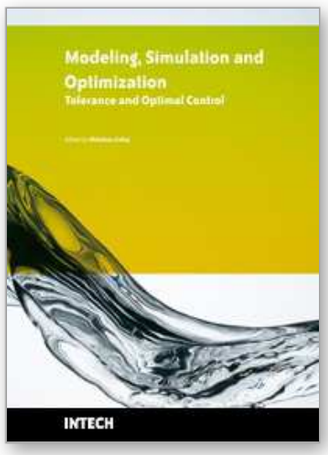
Hiroshi Umeo, Shin Morishita, Katsuhiro Nishinari, Toshihiko Komatsuzaki, and Stefania Bandini, editors. *Cellular Automata, 8th International Conference on Cellular Automata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008*, volume 5191 of *Lecture Notes in Computer Science*, 2008. Springer.

S. Wolfram. *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, Reading, Mass., 1994.

Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors. *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006*, volume 4173 of *Lecture Notes in Computer Science*, 2006. Springer.

IntechOpen



Modeling Simulation and Optimization - Tolerance and Optimal Control

Edited by Shkelzen Cakaj

ISBN 978-953-307-056-8
Hard cover, 304 pages
Publisher InTech
Published online 01, April, 2010
Published in print edition April, 2010

Parametric representation of shapes, mechanical components modeling with 3D visualization techniques using object oriented programming, the well known golden ratio application on vertical and horizontal displacement investigations of the ground surface, spatial modeling and simulating of dynamic continuous fluid flow process, simulation model for waste-water treatment, an interaction of tilt and illumination conditions at flight simulation and errors in taxiing performance, plant layout optimal plot plan, atmospheric modeling for weather prediction, a stochastic search method that explores the solutions for hill climbing process, cellular automata simulations, thyristor switching characteristics simulation, and simulation framework toward bandwidth quantization and measurement, are all topics with appropriate results from different research backgrounds focused on tolerance analysis and optimal control provided in this book.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen