SCIENCE ▪ PASSION ▪ TECHNOLOGY

Elias Wachmann

Investigating Cellular Automata in Physics: A Case Study of Diffusion and Densest Circle Packing

BACHELOR'S THESIS

Bachelor's degree programme: Software Engineering and Management

Supervisors

Limbacher Thomas Dipl.-Ing.

Institute of Theoretical Computer Science
Graz University of Technology

Hauser Andreas Assoc.Prof. Dr.phil. Dr.techn.

Institute of Experimental Physics
Graz University of Technology

Graz, April 2023

Abstract

Write Abstract

Keywords: computational physics · cellular automata (CA) · densest circle packing · rayleigh instability

Contents

List of Figures

List of Tables

1 Introduction

Cellular automata (CA), first discovered in the 1940s by John von Neumann and Stanislaw Ulam, have been employed to model a wide range of systems [1]. CA are comprised of a grid, which discretizes space into cells, and simple transition rules that are applied to each cell at every time step. Remarkably, complex behavior can emerge from these elementary rules. One of the most well-known examples of a CA is Conway's Game of Life, a zero-player game introduced by British mathematician John Horton Conway in 1970 [2].
Aside from Conway's Game of Life, CA have been used to model the behavior of complex systems in various domains, such as traffic flow [3], epidemic spreading [4], and even building evacuation [5].

In this thesis a CA is used to model the transient evolution of the densest circle packing. Using the eight direct neighbors as an analogy for the cell's total energy a probabilistic transition rule using a probability function is applied to calculate the transition of cells. The resulting CA is used to model the transient evolution of a densest circle packing. Furthermore, Brownian motion and diffusion phenomena, as well as Rayleigh instability are studied.
Several models based on CA have been used to simulate diffusion [6, 7]. Especially the deformation during solidification of polymers is an area of interest. An osmotic-driven transport mechanism was proposed by Aggeliki [8] to explain the deformation from a spherical to a hexagonal shape. Subsequently, the proposed mechanism was discarded by Koch after further experiments [9].

For the presented CA model different initial configurations – based on the densest circle packing and various other starting conditions – are assessed with different parameters governing the diffusion speed. Different physical behaviors, such as Rayleigh instability (also simulated by Hauser [10]) and Brownian motion, in addition to the formation of the hexagonal shape are observed.

A comprehensive background chapter on cellular automata and their use in physics, as well as the physical phenomena of diffusion, sets the stage for the research questions and objectives addressed in this bachelor's thesis. Thereafter, a chapter on modelling describes implementation details using densest circle packing as an example. Relevant sections of the source code are analyzed in a complexity analysis to gain insights on the time and space complexity of the CA. The results of the simulations are presented with parallels drawn to the underlying physical phenomena. Finally, the thesis concludes with a discussion of the results and a summary of the findings.

2 Background

This chapter contains background information on cellular automata (CA) and their use in physics. Furthermore, the physical emerging phenomena are described.

2.1 Cellular Automata

Cellular automata were first studied by Stanislaw Ulam and John von Neumann in the 1940s. Von Neumann introduced the concept in lectures in preparation for the Hixon Symposium and further developed this concept in a subsequent paper. [1, 11] After laying the foundation by explaining the basic concepts behind CA, this section will focus on the classification brought forward by Stephen Wolfram and highlight generalizations and analogous automata.

2.1.1 Basic Concepts

To simulate any kind of physical system space and time must be discretized. The space discretization can be implemented on a grid as shown in Figure 2.1. In this particular example the 1×5 grid represents an elementary (one-dimensional) CA with equally spaced points represented by the squares. Grids in other dimensions using cubes, hypercubes or even other space-filling shapes for discretization can also be realized. Time denoted as t, as shown in Figure 2.1, is also discretized in arbitrary equally spaced time steps. The CA given has only one transition rule:

- If the left neighbor of a cell is colored (black in Figure 2.1) $\rightarrow$ color the cell.

For every time step the transition rule is evaluated for every cell and if the rule applies, the cell is colored in the following time step. This procedure repeats until no more cells are colored (for $t \geq 5$), resulting in a steady state from which the automata will not change anymore.
The keen observer might have noticed that it is important to define boundary conditions for the automata. In theory one can define the CA to exist on an infinite canvas where all cells are blank except the ones set according to the initial state (this is the case in Figure 2.1). However, this theoretical infinite canvas cannot be recreated in practice. Therefore, one must define a finite canvas with boundary conditions governing the behavior at the edges of the simulation area, as discussed in chapter 3.

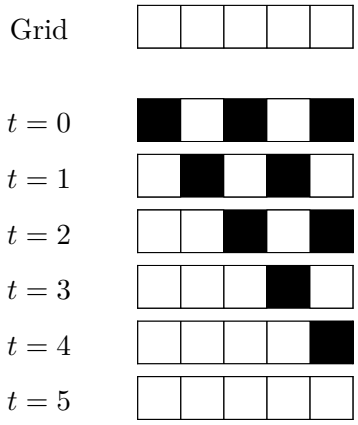


Figure 2.1: Grid showing the space and time discretization.
Grid: empty grid. Below: time (t) evolution of the grid.
The transition rule colors cells if the left neighbor of the cell is colored.

Not every CA is as trivial as the one above. One can construct automata which self replicate or evolve from their initial state. For some automata, like Conway's Game of Life, it was even shown that they are Turing complete. [12]

2.1.2 Stephen Wolfram's classification

Starting in the 1980s Stephen Wolfram put forward a qualitative classification of CA which consisted of four classes. They are categorized using heuristics describing growing complexity with the stability and growth of the automata as given below: [13]:

1. disappears after a finite number of steps – spatially homogeneous
2. evolves into a steady state with finite size – simple stable or periodic structures
3. grows at a fixed rate indefinitely – chaotic aperiodic behavior
4. grows/shrinks at a variable rate – complicated localized structures, some propagating

Wolfram introduced these classes after studying and classifying the rules of elementary cellular automata as shown in Figure 2.1. Elementary CA are one-dimensional, meaning a cell's neighborhood consists of two neighbors and the cell itself. With two different states per cell a total of eight states are possible for each neighborhood. The time evolution of the central cell is determined by its neighborhood this leads to a total of $2^8 = 256$ possible transition rules. [14] All possible states for the neighborhood are shown in Table 2.1.

Table 2.1: Possible neighborhoods and next state for a cell in an elementary CA using rule 126 (= `0b01111110`).

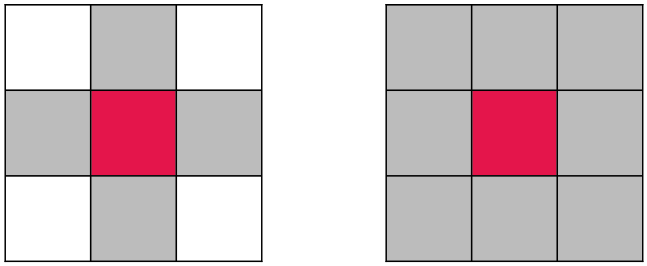
neighborhood	111	110	101	100	011	010	001	000
next state	0	1	1	1	1	1	1	0

Different transition rules yield different results as can be seen in Figure 2.2. While the class 1 automata, governed by rule 32, on the right quickly disappears after only two timestep, the class 2 CA – which is based on rule 4 – exhibits a steady-state from the onset. Rule 30 is an example of a class 3 automata exhibiting fixed rate growth with aperiodic character. Rule 110 is an example of a class 4 automata which exhibits chaotic behavior with a variable growth rate. [13]

Figure 2.2: Stephen Wolfram's four classes of CA. From left to right: class 1 using rule 32, class 2 using rule 4, class 3 using rule 126 and class 4 using rule 110. The initial state is set to 42 padded with zeros on each side. The top row represents the initial state `0b00101010` with 25 time steps shown below.

2.1.3 Von Neumann and Moore neighborhoods

For elementary CA the definition of a neighborhood is rather simple. In contrast, a higher dimensional automaton may offer different approaches to define a neighborhood. For a two-dimensional CA the two neighborhood definitions which are most commonly used are the Moore and the Von Neumann neighborhoods, as shown in Figure 2.3. While the Von Neumann neighborhood only considers direct neighbors – meaning the four cells which edges touch the center cell – the Moore neighborhood considers all eight cells surrounding the center cell. [15]



Von Neumann neighborhood Moore neighborhood

Figure 2.3: Difference between Von Neumann and Moore neighborhoods. Center cell marked in red and neighbors marked in gray.

2.1.4 Generalization & related automata

As previously mentioned, a CA can be generalized to higher dimensions and even other geometric shapes besides a simple grid. The transition rules and elementary automata from above are discretized in space and time. Moreover, the transition rules only allow the transition from one discrete state to another in a deterministic manner. However, when modeling certain problems, such as physical phenomena, transition rule that employs a continuous function may be necessary instead of a discrete mapping. Although space and time discretization remains the same, the cells are no longer restricted to binary states, as they would in elementary cellular automata. These kinds of CA are referred to as continuous cellular automaton. [15]

Besides the deterministic transition rules from elementary automata, there are also CA which employ probabilistic transition rules, so-called probabilistic cellular automata. [16] The automata presented in this thesis has the ability to act in a probabilistic, as well as deterministic manner. Furthermore, states may be continuous, as well as discrete making it a probabilistic continuous cellular automata.

2.2 Physical phenomena

There are several physical phenomena which have been modeled using various types of CA. These include high energy physics, physical chemistry, thermodynamics and even encryption, using quantum CA. [8, 17, 18, 19] The following subsections will provide an overview of physical phenomena which are relevant for the thesis.

2.2.1 Brownian Motion & Diffusion

Brownian motion, first discovered by Robert Brown in 1827, refers to a random motion of tiny particles in fluids. Although he first observed the motion in pollen swimming in water, he later confirmed with inorganic particles that the random movement is not caused by organic life. An explanation for the motion wasn't found by Brown but rather Einstein in 1905 [20]
Einstein proposed that the density distribution of the random motion of a particle starts out as a Dirac delta function at the origin of the particle and spreads out into a Gaussian distribution centered around its origin. Using statistical methods, he was able to derive the spread of the distribution using the kinetic theory of gases. He showed that the square of the expected value for a particle's distance to the origin after the time t is proportional to a diffusion constant D. Aside from particle size and the coefficient of friction of the fluid, D is directly proportional to the temperature. [21]
Coincidentally Karl Pearson coined the term Random walk, which, rather fittingly, describes the random motion of a point on a lattice in 1905 as well. Within the constraints of the lattice, a point can move to any of its neighbors per time step. However, the direction of this movement is purely random, hence the name random walk. [22] If one considers an infinitely small lattice spacing, the random walk describes Brownian motion of particles perfectly. [20]

Diffusion is a process which describes the emerging phenomena by which particles spread out in a medium if there is a concentration gradient present. As stated above, the diffusion constant D which is directly proportional to the temperature also relates to the speed at which particles diffuse.[23]

2.2.2 Rayleigh instability

As with many observed phenomena, energy minimization plays a key role for Rayleigh instability. Fluids exhibit a tendency to minimize their energy by minimizing their surface area. For a stream of fluid flowing out of a tap, one can observe that a continuous stream of fluid tends to split up into many spherical drops. Rayleigh instability describes the process by which these droplets separate from the stream. [24]
Recently, Rayleigh instability was simulated using a probabilistic CA and also observed in vitro for solid nanowire structures. The rate at which these wires break up is proportional to the temperature. Higher temperatures lead to a quicker decomposition of the original structure into various smaller circle-/elliptically shaped regions. [10, 25]

3 Modelling densest circle packing

After an introduction and a general outline of the used techniques, the algorithm is described. Subsequently, a time and space complexity analysis is presented directly followed by a short outline of comparable CA models.

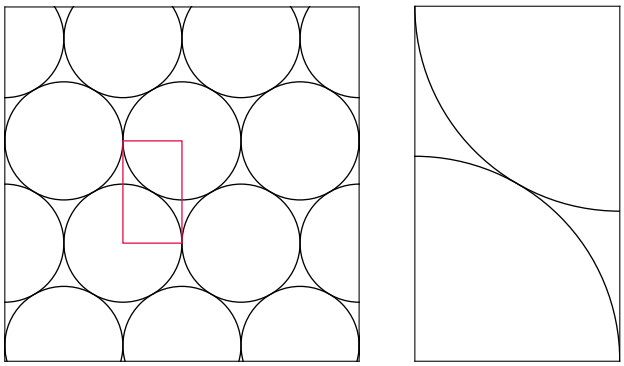
3.1 Overview of cellular automata

This section will outline design considerations during development of a CA modeling the transient evolution of the densest circle packing. First the discretization of the geometry will be explained, followed by the transition rules and the chosen boundary conditions. Finally, mass conservation and varying input parameters in the simulation will be discussed.

To motivate the usage of a CA further, a practical example in the animal kingdom is given. It is well known that honeycombs are hexagonal. This is a rather intriguing observation, in fact a hexagonal shape provides the most space to store honey while also requiring the least amount of wax to construct. Even though bees are mighty creatures, they do not build their combs in this rather complicated hexagonal shape but rather in a circular shape [26]. These cylindrical combs evolve into the well known hexagonal shape as the wax cools down and hardens.

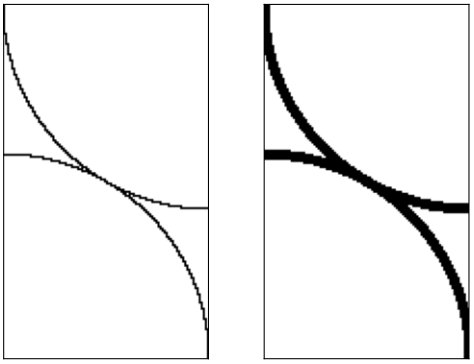
3.1.1 Discretization of the geometry

The geometry is given by the densest circle packing, as shown in Figure 3.1. Instead of simulating a larger area consisting of multiple circles, the geometry can be fully described with a smaller area, which is also shown in Figure 3.1 (b). To simulate the transient behavior of this geometry using a CA, the geometry needs to be discretized into a grid. This is done by dividing the whole area into a grid with uniform and equal grid spacing in both dimensions. Subsequently, the geometry is mapped onto this grid. A cell is considered to be part of the circle if at least one of its four corners is inside the circle and at least one is outside. For the given geometry, this will result in discretized circles with a thickness of one cell.
Further analysis in section 4.4 shows, that this will pose a problem if a CA with discrete binary states is considered, as the circles will split up rapidly. Two solutions come to mind combating this problem: either the states are made continuous – by changing



(a) (b)

Figure 3.1: Densest circle packing of a set of circles (a). The red region encapsulates the geometry of the problem fully when using mirrored boundary conditions (b).



(a) (b)

Figure 3.2: Rasterization without (a) and with (b) thickness.

the datatype of the underlying array to float or double and incorporation continuous transition rules – or the thickness of the circles is increased. The later solution is chosen and the thickness of geometric lines is added as another parameter. Circles in particular implement thickness τ in the discretization step by checking if a cell lays inside the region given by an outer radius r_{outer} and inner radius r_{inner} of the circle which is defined as follows:

$$r_{\text{outer}} = r + \tau/2 \qquad\qquad r_{\text{inner}} = r - \tau/2$$

where r is the radius of the circle. Cells which have at least one corner inside the area between r_{outer} and r_{inner} are considered to be part of the circle. An example of the discretization is given in Figure 3.2 with and without thickness.

3.1.2 Transition function

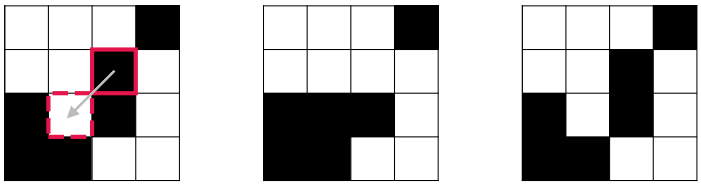
The transition function for the given CA chooses a random neighbor of the current cell and decides based upon the difference in occupancy for neighbors of the current and the randomly chosen cell if the current cell should exchange its mass with the random neighbor. In this context, a cell's mass refers to a boolean true and is colored in black. In the example given in Figure 3.3, the current cell (framed in solid red) has a mass of 1 while the randomly chosen neighbor (framed in dashed red) has a mass of 0. This means a transition of the mass from the current cell to the neighbor is possible. If instead both cells had a mass of 1 a mass transfer would increase the neighbor's mass to 2 which is not allowed in the presented CA. Additionally, just ignoring this problem by disregarding the illegally transferred mass to the neighbor is not acceptable, as this would constitute a mass loss in the system. This is why only mass transfers from cells which have mass to others which do not have any are considered.
Given the aforementioned prerequisites, the transition function is defined probabilistically using a Monte Carlo approach as shown in Algorithm 1 with the probability function p defined in Equation 3.1. Where p is a sigmoid function which depends on a constant λ and the difference in mass Δm between the Moore neighborhoods of the current cell and the neighbor chosen at random. Higher λ values decrease the probability of a mass transfer, while lower λ values increase the probability. In a physics context λ can be thought of as inversely proportional to the temperature T of the system.

Algorithm 1 Transition function of the CA

```
1: function PerformMassJump(λ, Δm)
2:     if random_uniform(0, 1) < p(λ, Δm) then
3:         return True
4:     else
5:         return False
6:     end if
7: end function
```

$$p(\lambda, \Delta m) = \frac{1}{e^{-\lambda \cdot \Delta m} + 1} \tag{3.1}$$

In the concrete example in Figure 3.3 neighbors of the current cell containing mass are counted. Using its Moore neighborhood, one obtains two neighbors with mass for the current cell and four for the randomly chosen neighbor. To avoid skewing the probability in favor of the neighboring cell, the current cell is not counted as a neighbor with mass. Calculating the difference in mass between the two Moore neighborhoods yields $\Delta m = 2$. If λ is set to 1 the transition probability p will be approximately 88 %. This practically means that the cell's mass will be transferred to the random neighbor in roughly 9 out of 10 cases.



initial state | mass transferred (88% probability) | no mass transfer (12% probability)

Figure 3.3: Probability of a mass transfer from the current cell (framed in solid red) to a randomly chosen neighbor (framed in dashed red) according to Algorithm 1 with $\lambda = 1$.

Equation 3.1 has interesting properties which will come in handy later in the simulation. To start with, the probability of a mass transfer between neighborhoods with the same number of occupied cells is independently of λ always $1/2$. Additionally, for growing λ the probability function approaches a Heaviside step function which supports formation of stable structures, as discussed in section 4.4.

It should also be noted that mirrored boundary conditions are used in the simulation. This way a cell on the edge of the simulation area has at least itself as a neighbor – as if it would look into a mirror placed at the edge of the simulation area. Cells in corners subsequently have at least 3 neighbors.

3.1.3 Conservation of mass

The total mass in a simulation is given by all cells which are set to true in the initial state. To obey the principle of mass conservation, the total number of cells with mass must not change for any time step in the simulation. Although discussed in the previous section, mass can still vanish from the system in the following way: For each time step, all cells are updated according to the transition rules described above. All updates happen independently of each other, without knowledge of the changes in the cells neighborhoods induced by other updates. Multiple cells may now transfer their mass to another cell which originally had no mass. Figure 3.4 illustrates how this leads to a loss of mass in the system, thus breaking the law of mass conservation.

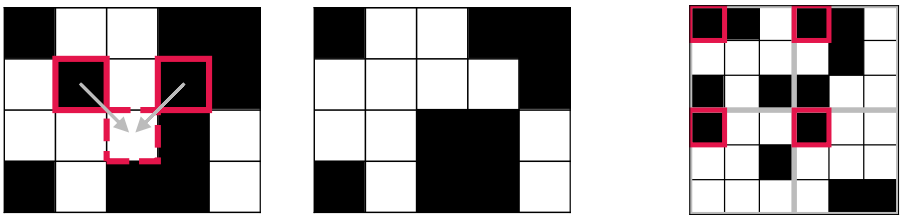


Figure 3.4: Violation of mass conservation.
Left: Initially the total mass of the system is 10; After the transition it has decreased to 9.
Right: Calculation transitions on a super grid (gray) to avoid mass loss.

This problem can be solved by calculating the transitions on a super grid as shown in Figure 3.4 on the right. A super grid cell consists of a 3×3 area of the original grid. For each time step, a random cell offset inside the 3×3 area is chosen. Figure 3.4 shows the super grid in gray cells in gray which contain the cell at the chosen offset (framed in red). Now only framed cells will be updated according to the rules. In this way it is ensured that there are at least two cells between simultaneously updated cells, this in turn guarantees that no mass is lost. In case a transition from an edge to a neighbor outside the simulation area would take place, the transition is handled – according to the mirrored boundary conditions – like a transition with the sole (Von Neumann) neighbor of the cell outside the area which lays within the valid area. In case of corners, the current and neighbor cell is thus practically the same cell.

Using this methodology in the CA there is no possibility to create additional mass, thus the super grid approach is sufficient to ensure mass conservation. Through the offset's randomness, the mean time between two updates of the same cell is 9 time steps, which means that the simulation has to be run longer to obtain the same results as with the original grid.

3.1.4 Varying input parameter

As mentioned above, the CA model consists of a transition rule influenced by a parameter λ. However, holding this parameter constant for the whole duration of the simulation is not representative of many processes. Especially if outside parameters like temperature change or certain chemical reactions are initiated or stopped λ is bound to change, to model these variations. Illustratively, the example with the honeycombs shows this: While the bees are building cylindrical shaped combs at first, temperature T is higher and thus Brownian motion and diffusion remains vigorous. After finishing one patch and leaving the immediate area, the wax cools down and solidifies, hampering Brownian

motion in the process. For a sufficiently small area (with mirrored boundary conditions) the assumption can be made, that the cooling takes place uniformly. Within these limitations, λ may be expressed by a function inspired by the fundamental solution of the heat equation, the so-called head kernel. Equation 3.2 can be used to model the cooling using two constants d and h as well as the time t as an independent variable.

$$\lambda(t) = \lambda_{max} \cdot \left(1 - \frac{1}{\sqrt{d \cdot h \cdot t}} \cdot e^{-\frac{1}{h \cdot t}}\right) \tag{3.2}$$

By lowering the constant value d the function's global minimum is lowered. Lowering h stretches the function while increasing the constants lifts the minimum and compresses the function as shown in Figure 3.5.

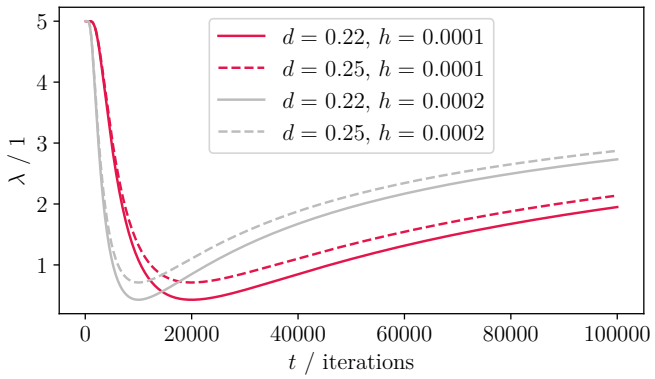


Figure 3.5: Examples for Equation 3.2 using $\lambda_{max} = 5$ and various values for the constants d and h.

3.2 Implementation details

After considering various options for the programming language – especially with regards to the performance of the CA – the decision was made to use `python`. This ultimately was due to the fact that prototyping and thus developing the first version in `python` is generally faster than in more performant languages like `C++` [27]. To speed up the simulation of the CA the `numba` [28] library was utilized for just in time (jit) compilation of certain functions. A review of different approaches to speed up the base implementation in `python` is presented in section 4.6.

3.2.1 Components

For convenience, the project is split into four main parts:

- Rasterizer: converts geometry to a rasterized grid
- Simulation: implements the CA
- Transitions & utility functions: jit compiled functions & other helper functions
- Visualizer: visualizes the simulation

The rasterizer's rasterizer method shown in Algorithm 2 takes a list of shapes and rasterizes them into a grid. Shapes must implement the *passesCell*-function which indicate whether the shape passes through a specified cell. In this way the rasterizer class is independent of the shape and can be used to rasterize any kind of class which implements the *passesCell*-function.

Algorithm 2 Rasterize method of the Rasterizer class

```
1:  function RASTERIZE(shapes : Sequence)
2:      for i ← 0 to len(self.xrange) − 1 do
3:          for j ← 0 to len(self.yrange) − 1 do
4:              x ← self.xrange[i]
5:              y ← self.yrange[j]
6:              for shape in shapes do
7:                  if shape.passesCell((x, y, self.xstep, self.ystep)) then
8:                      self.raster[i, j] ← 1
9:                      break
10:                 end if
11:             end for
12:         end for
13:     end for
14:     return self.raster
15: end function
```

xrange and *yrange* are used to specify the grid spacing, which is a uniform spacing, with distances *xstep* and *ystep* between cells. The finished raster is then returned to the Simulation class, which implements the CA.

Aside from some utility code to handle different Simulation settings, the Simulation class consists of the three methods: RUN, STEP and STOP. An iteration loop runs within the RUN method (outlined in Algorithm 3) with each iteration being a time step in the CA. For a given parameter λ – either defined via a generator or as a constant – the STEP method is called. At first a deep copy of the current grid is made and the random offset inside the 3×3 area of the super grid is chosen. Subsequently, RUN, as shown in

Algorithm 4, calls updateAllCells and wraps up by overwriting the current grid with the grid computed in updateAllCells.Finally, the stop function ends the simulation in case of a keyboard interrupt or if the last time step is reached.
The self keyword in the pseudocode references class variables which are available to the aforementioned methods.

Algorithm 3 Run method of Simulation class

```
1: function run(iterations)
2:     for i ← 0 to iterations − 1 do
3:         if self.lambda_iter exists then
4:             self.lambda ← next(self.lambda_iter)
5:         end if
6:         step( )
7:     end for
8:     stop( )
9: end function
```

Algorithm 4 Step method of Simulation class

```
1: function step
2:     next_grid ← deepcopy(self.grid)
3:     grid_offset ← random_integer(0, 3, 2)        ▷ Two random ints ∈ {0, 1, 2}
4:     i_range ← (grid_offset[0], self.max_i, 3)
5:     j_range ← (grid_offset[1], self.max_j, 3)
6:     updateAllCells(self.grid, next_grid, i_range, j_range, self.lambda,
   self.max_i, self.max_j)
7:     self.grid ← next_grid
8: end function
```

Computationally intensive functions which need to be performant like updateAllCells are outsourced and jit compiled to improve simulation speeds. During each simulation step Algorithm 5 loops over the super grid and calls neighborWeightTransfer. As shown in Algorithm 6 the latter mentioned function performs a few basic checks, gets the neighbor mass count for the current cell and the randomly chosen neighbor and finally calls performMassJump – discussed in subsection 3.1.2 – to decide whether the mass should move to the new cell or remain at its current one.

Algorithm 5 Update all cells (jit-compiled)

function updateAllCells(*grid*, *next_grid*, *i_range*, *j_range*, *lambda*,*max_i*, *max_j*)
    i_start, *i_end* ← *i_range*[0], *i_range*[1]
    j_start, *j_end* ← *j_range*[0], *j_range*[1]
    for *i* ← 0 to *i_end* − 1 **do**
        for *j* ← 0 to *j_end* − 1 **do**
            neighborWeightTransfer(*grid*, *next_grid*, *i* ∗ 3 + *i_start*, *j* ∗ 3 + *j_start*, *max_i*, *max_j*, *lambda*)
        end for
    end for
end function

Algorithm 6 Neighbor weight transfer function (jit-compiled)

function neighborWeightTransfer(*old_grid*, *new_grid*, *i*, *j*, *max_i*, *max_j*, *lambda*)
    if *old_grid*[*i*, *j*] == **False then return**
    end if
    (*i_n*, *j_n*) ← getRandomDirectNeighbor(*i*, *j*, *max_i*, *max_j*)
    if *old_grid*[*i_n*, *j_n*] == **True then return**
    end if
    initial_neighbor_mass ← getNeighborOccupancy(*old_grid*, *i*, *j*, *max_i*, *max_j*)
    old_grid[*i_n*, *j_n*] = **True**
    old_grid[*i*, *j*] = **False**
    new_neighbor_mass ← getNeighborOccupancy(*old_grid*, *i_n*, *j_n*, *max_i*, *max_j*)
    $neighbor\_mass\_diff \leftarrow new\_neighbor\_mass - initial\_neighbor\_mass$
    old_grid[*i_n*, *j_n*] = **True**
    old_grid[*i*, *j*] = **False** ▷ restore old grid to avoid influencing other transitions
    if performMassJump($neighbor\_mass\_diff$, *lambda*) == **False then**
        return False
    else
        new_grid[*i_n*, *j_n*] = **False**
        new_grid[*i*, *j*] = **True**
        return True
    end if
end function

Functions regarding plotting are implemented inside the Visualizer file and can be called directly during the simulation or plot saved data afterwards.

3.2.2 Time & space complexity

Regarding time and space complexity, the whole simulation process can be analyzed as follows:
The rasterization's time complexity is given by the number of grid points per dimensions (n_x & n_y) and the number of shapes n_s. For the two-dimensional case, the following time (denoted $\mathcal{O}_t$) and space (denoted $\mathcal{O}_s$) complexities for the rasterization stage are obtained:

$$\text{Rasterize:} \quad \mathcal{O}_t(n_x \cdot n_y \cdot n_s) \quad \mathcal{O}_s(n_x \cdot n_y)$$

Within nearly all practical use cases n_s will be negligible compared to n_x and n_y, for a finer grid with the same geometry the time complexity is bound to grow quadratically. Compared to the simulation, the rasterization takes very little time. Within the simulation there are multiple nested loops, one iterating through t_{max} time steps and two other in updateAllCells which iterate on the super grid cells ($^{n_x}/_3 \times {}^{n_y}/_3$). Random offset and Neighbor selection in step and neighborWeightTransfer has constant time complexity. In addition, all other steps in neighborWeightTransfer are only dependent on the dimensionality of the problem. getNeighborOccupancy has to check 8 neighbors in the two-dimensional case, which can be thought of as a constant leaving us with the following complexities for the simulation:

$$\text{Simulation:} \quad \mathcal{O}_t(n_x \cdot n_y \cdot t_{max}) \quad \mathcal{O}_s(n_x \cdot n_y)$$

Deep copies as well as the overwriting of the old grid in step are off the same order in terms of time complexity ($\mathcal{O}_t(n_x \cdot n_y)$) when compared to the nested loops in updateAllCells. Space complexity is comparable to the space complexity in the rasterize stage and only depends on the dimensionality of the problem.

For all practical purposes, the simulation will dominate the time of the whole process. Therefore, the time complexity is of linear order in the number of grid points and time steps. The space complexity is also linear in the number of grid points.

3.3 Comparable CA models

Various groups have experimented with similar CA and CA-inspired models. Strikingly, diffusion behavior was even shown for elementary CA (see subsection 2.1.2) by Grassberger [29]. This is particularly interesting because one would expect from the theory, that diffusion like behavior only arises stochastically, thus requiring a probabilistic CA. Other groups have used the later methods incorporating a Monte Carlo process – comparable to the one shown above – to model the same behavior. [10, 30].
These models offer a rather different approach than the established algorithms used to solve PDEs. Certainly problems of other numeric methods like propagation of rounding

off errors and problems with convergence may be avoided by using probabilistic CA methods.

4 Results & Evaluation

Starting out with the results for densest circle packing simulation, this chapter will also include additional results for Brownian motion and Rayleigh instability simulations. Subsequently, the stability of the CA dependent on different parameter settings is discussed and convergence is tested for different initial states and parameter variations. The chapter concludes by presenting a performance comparison for various optimizations in the CA-code.

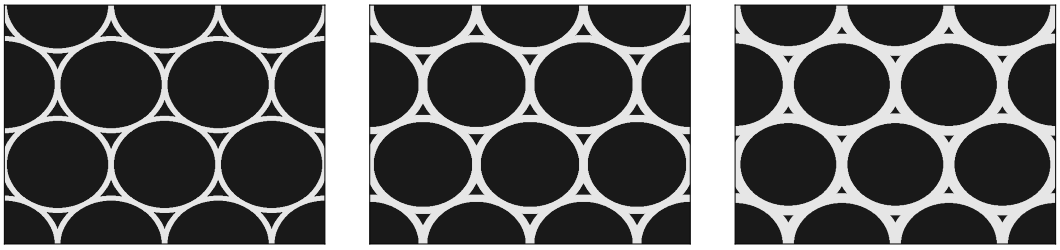
Note, that for this chapter, the cell coloring convention is flipped. From now on, white cells contain mass and black cells are empty.

4.1 Densest circle packing

As described in chapter 3 the geometry similar to the one shown in Figure 3.1 was rasterized for the simulation. Compared to the figure above, the simulation used a $6 \times 3 \cdot \sqrt{3}$ rectangular area on which mirrored boundary conditions are applicable. Using circles with radius 1 and a uniform grid spacing of 0.01 the simulated space consists of $600 \times 519 = 311400$ cells. First simulations using a constant parameter λ are presented, followed by result of simulations using a varying λ.

4.1.1 Constant λ

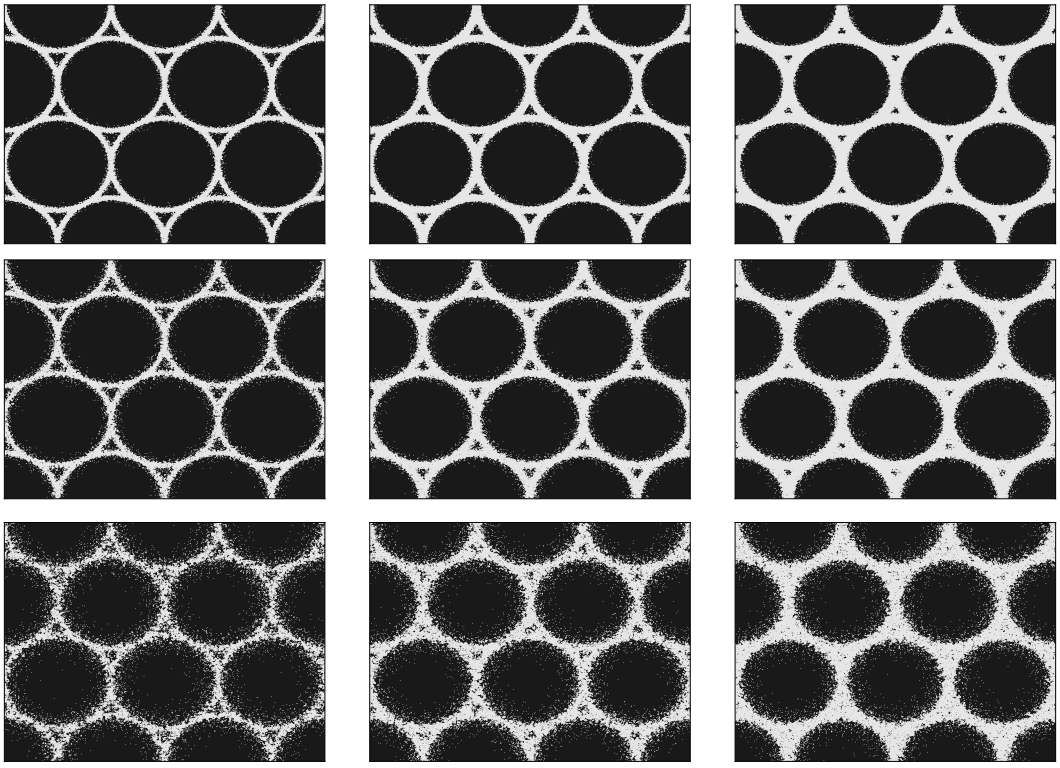
In addition to the constant λ which is initially chosen, one can also change the thickness of the circles (as defined in subsection 3.1.1) resulting in different starting conditions shown in Figure 4.1.



10 15 20

Figure 4.1: Densest circle packing different thicknesses. Each circle has a radius r of 100 cells, the thickness for each starting condition is given under the corresponding plot.

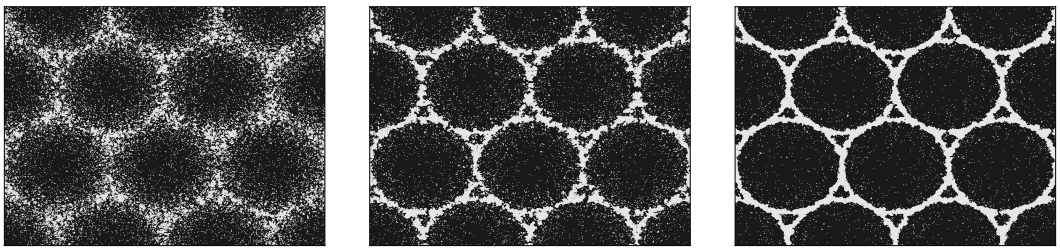
Performing the Simulation for all three test cases the following results are obtained for the sampled time steps at 500, 2000 and 10000 iterations in Figure 4.2.



10 15 20

Figure 4.2: Time evolution for constant $\lambda = 0.6$ from the given start conditions in Figure 4.1. Top row 500, middle row 2000 and bottom row 10000 iterations. The original thickness (in cells) of the circles is again given under the plots.

Changing the constant parameter λ directly influences the rate of the diffusion as shown in Figure 4.3. While a λ of 1.0 introduces lots of stray cells inside the circles, the initial structure of the circle packing is still clearly visible. Lowering λ to 0.8 drastically increases the breaking of the structure (further discussed in section 4.3). Interestingly one can not expect the same behavior – given a longer runtime – in the simulation with $\lambda = 1.0$ compared to $\lambda = 0.8$. Further discussion on the topic of stability follows in section 4.4. Naturally, setting λ to a low value predictably results in more straying cells and closes gaps formed by Rayleigh breaking which would persist for higher λ as shown on the left in Figure 4.3.

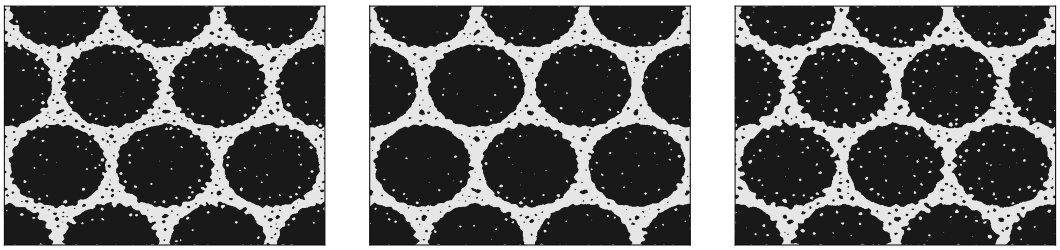


$\lambda = 0.6$ $\lambda = 0.8$ $\lambda = 1.0$

Figure 4.3: Densest circle packing evolution for different λ after 50000 iterations. Each circle starts out with an initial thickness of 10 cells.

4.1.2 Varying λ

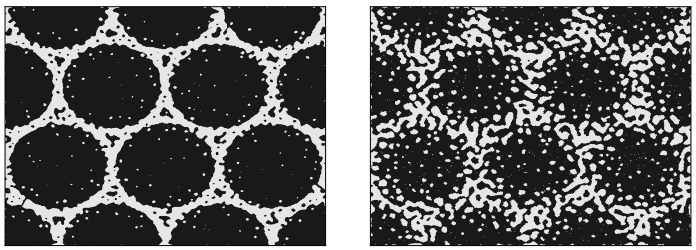
By varying the parameter λ over time, the strengths of both higher and lower values can better be utilized to obtain a more realistic model of the diffusion process under the influence of a change in temperature. Using Equation 3.2 with different inputs for h and d, different outputs (shown in Figure 4.4) can be observed.



$h = 0.0005$, $d = 0.22$ $h = 0.0005$, $d = 0.23$ $h = 0.0002$, $d = 0.23$

Figure 4.4: Densest circle packing evolution for different variable λ after 100000 iterations. Each circle starts out with an initial thickness of 20 cells. Input parameters for Equation 3.2 are given below each figure.

Changing h and d and thus altering the duration λ has a significant impact on the results depicted above. A lower d with a comparable h results in a more porous structure. Lowering h further increases the tendency of the structure to separate, particularly shown by the narrowing of the circles in the right image in Figure 4.4.
For smaller thicknesses of the circles, the simulation exhibits this tendency even more as shown in Figure 4.5. This is to be expected, as the smaller thickness makes the gap formation more probable. Simulations which linger longer at lower λ values result in a structure with mostly disconnected islands, as shown in Figure 4.5 on the right. Even in these highly disconnected structures, a hexagonal pattern is visible.



$h = 0.0005$, $d = 0.22$
Thickness: 15 cells

$h = 0.0001$, $d = 0.2$
Thickness: 20 cells

Figure 4.5: Densest circle packing evolution for different variable λ after 100000 iterations. Input parameters for Equation 3.2 and thickness are given below each figure.

4.2 Brownian motion & Diffusion

The CA explained above can also be used to simulate Brownian Motion. By changing λ to a positive (attractive) or negative (repulsive) value, the behavior of the random walk for each cell can be altered. Starting from a state with cells set to 1 and 0 randomly, different behaviors are observed.

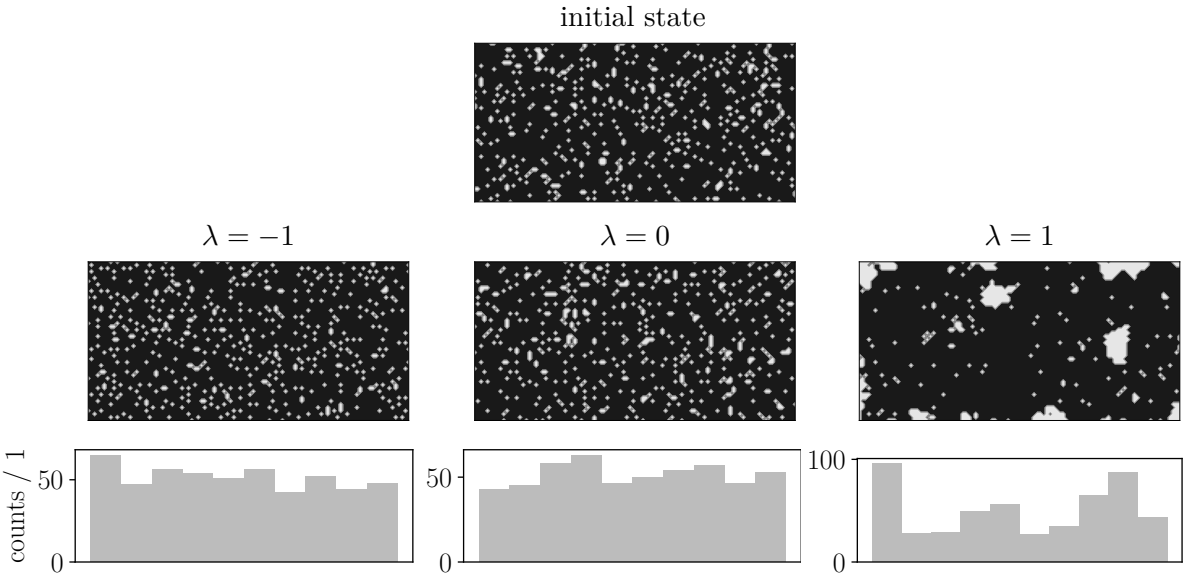


Figure 4.6: Brownian motion for randomly distributed cells on a 50×100 grid. 10 % of the cells are initially set to true at random. Top: initial state of the CA. Bottom: final state after 100000 iterations using different values for λ. Histograms: sum of true cells in the corresponding area above.

Starting from a random distribution with 10 % of the cells set to 1, λ is set to -1, 0 and 1. For a negative λ, mass transfers from a neighborhood with more neighbors to another one with fewer are more likely, resulting in a uniform distribution. This uniform distribution also persists if λ is set to 0 and is shown as a histogram in Figure 4.6 under the respective plots. Contrary to this, a positive λ quickly starts building clusters due to the higher likelihood of transitioning to a neighborhood with more neighbors. Naturally, the histogram changes accordingly.

However, not every initial state will eventually transition to a uniform distribution as shown directly above and in section 4.1. After setting the 10 leftmost cells to 1, the system will transition to a uniform distribution for certain λ. In the example in Figure 4.7 λ is set to -1 and 1 respectively. While the first system diffuses into a uniform distribution after 100000 iterations, the later one does not. Instead, most cells with mass will stay on the left hand side of the grid, while some stray cells will move to the right side following – if they do not encounter other cells in their neighborhood – a random walk.

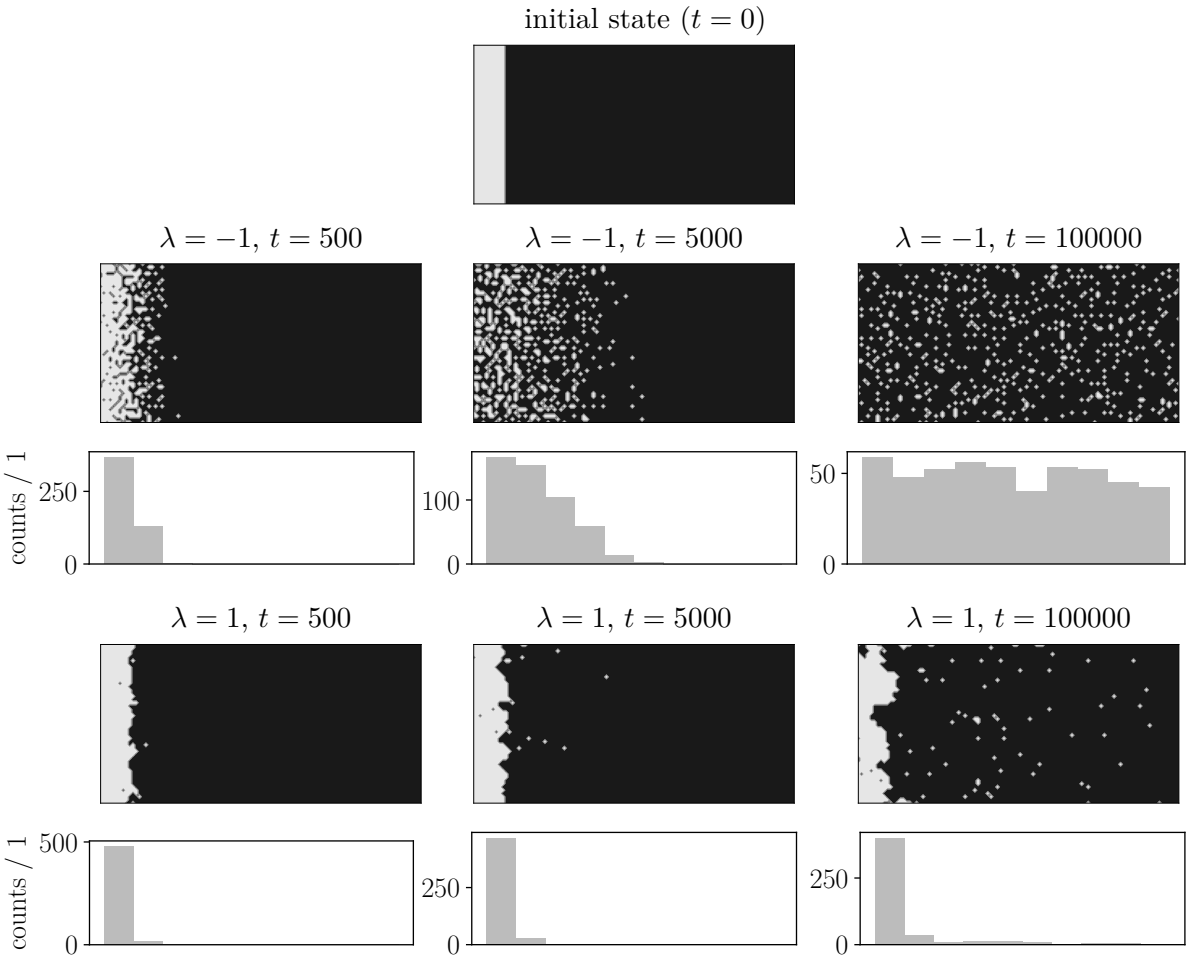


Figure 4.7: Diffusion of a bar. Initially a 9 cell wide bar is placed on the far left side. Depending on the value of λ the diffusion speed of the bar changes.

4.3 Rayleigh Instability

To further investigate the properties of the CA in question, a 2D-projection of a nanowire is simulated. The projection previously evaluated using a three-dimensional CA and also experimentally cross validated by Hauser [10] is used. Transmission electron microscopy (TEM) images of the nanowire are used to determine the initial state of the CA, which is discretized onto a 364×323 grid as shown in Figure 4.8.

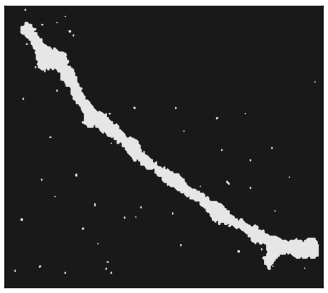


Figure 4.8: Initial state of the Nanowire. The initial state is discretized onto a 364×323 grid. (see Fig. 2 [10])

Rayleigh Instability can be observed in the results of the transient simulation with the given initial state for two different λ values in Figure 4.9.

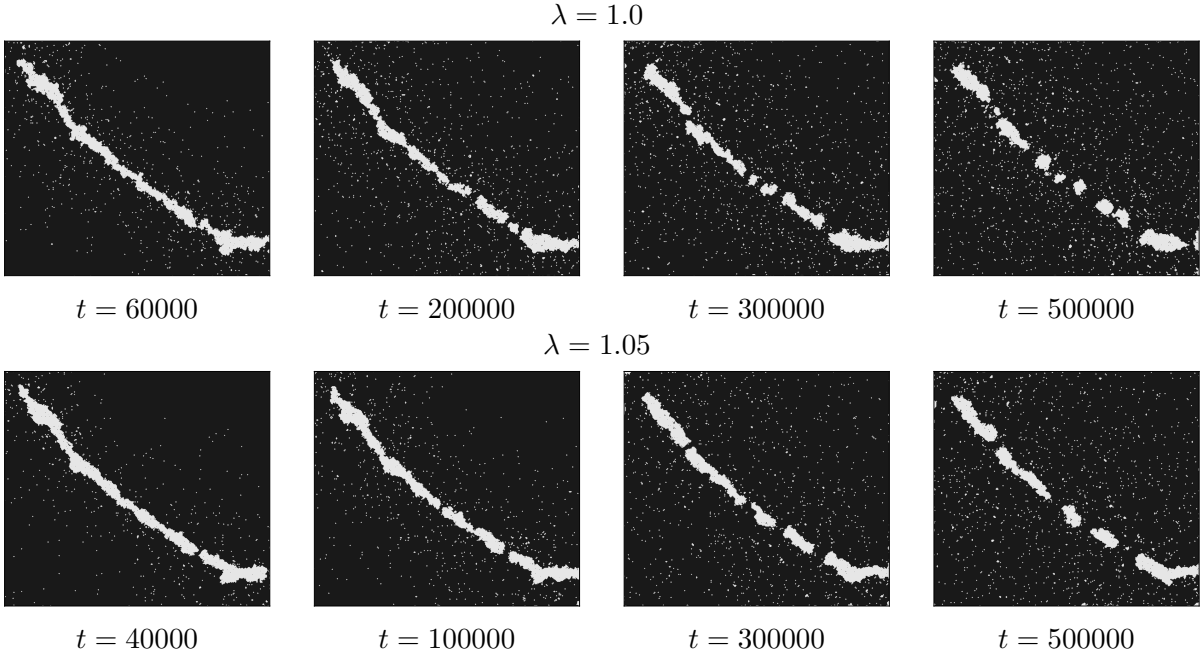
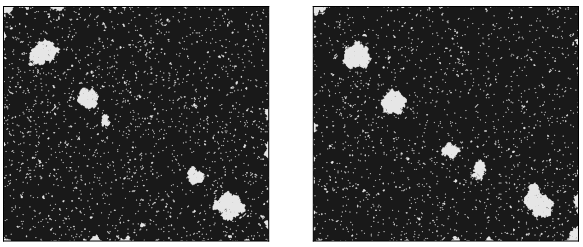


Figure 4.9: Rayleigh Instability simulation of a 2D-projection for a nanowire (see Fig. 2 [10]). The initial shape of the nanowire decomposes into multiple smaller shapes to form smaller elliptical blobs.

Surprisingly, in the simulation in Figure 4.9 the first breakup for $\lambda = 1.05$ happens at $t = 40000$ even before the first breakup for $\lambda = 1.0$ at $t = 60000$. This shows the inherent stochastic nature of the simulation, while the expected behavior – expecting faster breakups for lower λ – can be observed at time step $t = 300000$ and especially

at $t = 500000$. Here the lower λ breaks up into around 10 regions while the simulation using $\lambda = 1.05$ only splits into five.
For even longer iterations, the structure would further minimize the energy of the blobs. One or two blobs might still separate or even combine if they are very close, but overall each blob would evolve towards a more spherical shape like the ones shown in Figure 4.10.



$\lambda = 1.0$ $\lambda = 1.05$

Figure 4.10: Rayleigh Instability simulation of a 2D-projection for a nanowire (see Fig. 2 [10]). Resulting shape after a long runtime with $t = 3000000$ iterations for $\lambda = 1.0$ and $\lambda = 1.05$.

The aforementioned recombination can be observed in Figure 4.10 on the left, where various earlier unconnected areas re-merged into bigger circular shapes. Contrary, the simulation with $\lambda = 1.05$ only stabilizes from $t = 500000$ to $t = 3000000$ with no further splits or mergers.

4.4 Stability

Stability greatly depends on the value of the free parameter λ. Generally larger values benefit the stability of structures and smaller ones inhibit stable states and result in chaotic behavior, which resembles Brownian motion. Yet the analogy to Brownian motion is strictly only valid for $\lambda = 0$. This way no interaction, neither repulsive nor attractive, between different cells is present.

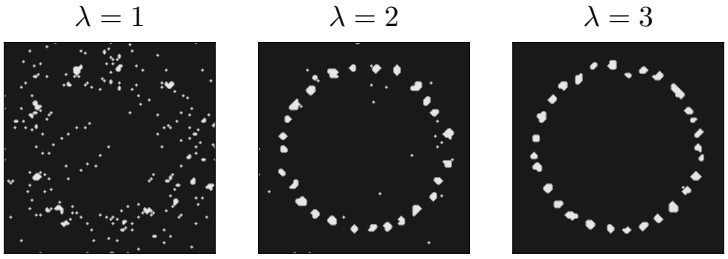


Figure 4.11: Stability of a circle with $r = 40$ cells on a 100×100 grid for different λ after $t = 5000$ iterations.

Aside from λ the geometry of the given problem may also change the stability. To begin with, as seen in section 4.3, circular shapes are more stable than elongated or thin line shapes elements. Areas with mass which are line shaped or elongated are more likely to break up into pieces due to the higher average of empty cells in the neighborhood of cells belonging to this structure. A circular shape minimizes the ratio of cells which are on the edge of the shape, compared to the total number of cells in the circle. However, this only holds true if the circular shape is filled. Otherwise, the circle will break up as shown in Figure 4.11. For $\lambda = 1$ the breakup is more chaotic and has more similarities to Brownian motion. Bigger λ values on the other hand exhibit ultimately stable breakups into smaller filled circular shapes analogous to the breakup processes of the Rayleigh instability.
Given the direct comparison for $t = 5000$ iterations and $\lambda = 1.0$ for both, Figure 4.11 and Figure 4.12 it is obvious, that the square shape evolves into a stable circular shape while the circle disintegrates.

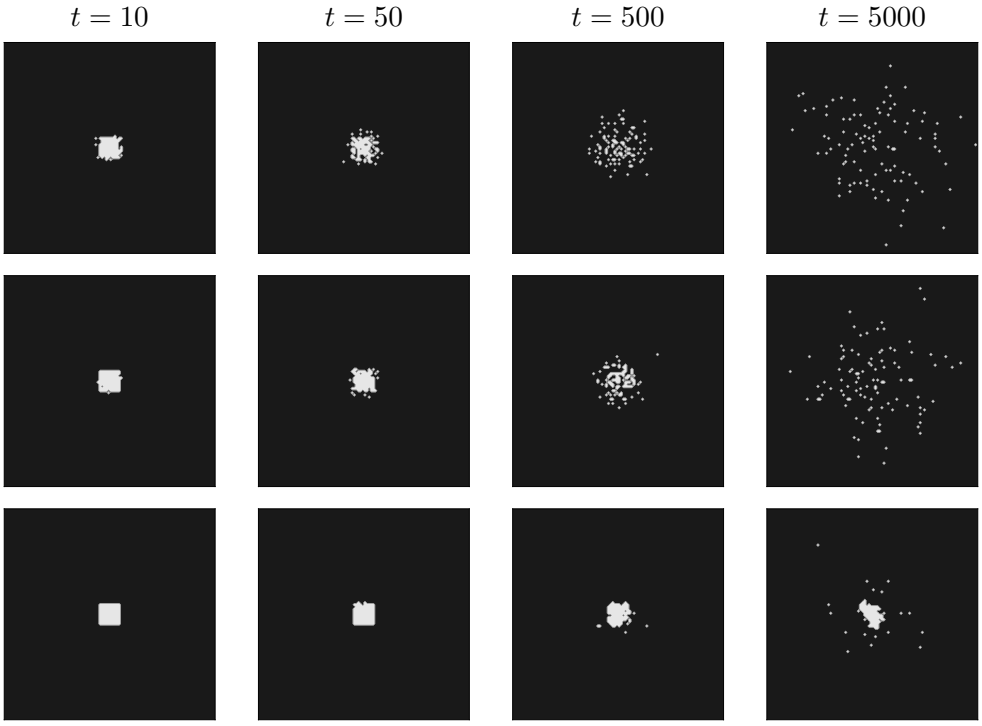


Figure 4.12: Stability of a 10×10 square on a 100×100 grid for different λ. Top row: $\lambda = -1$, middle row: $\lambda = 0$, bottom row: $\lambda = 1$. Iteration of CA is denoted above each column.

4.5 Convergence

Development of a general convergence metric for any kind of geometry is by far not trivial. One way to specify if a simulation has reached a stable state is to track the number of cells which change over time. If the rate of change of this function approaches zero, the simulation has reached a stable state. By this metric, a distinction between a stable state with structures and a stable state with Brownian motion can not be made. However, simply using the fact that there are more changes in total in a Brownian motion state than in a stable state with structures, the distinction can be drawn by comparing the total changes divided by the total number of cells with mass.
In Figure 4.13 the rate of change $\#_{\text{change}_{100}}$ is plotted against t for the example from Figure 4.6. Rate of change ($\#_{\text{change}_{100}}$) is calculated by first combining two states t_1 and $t_2 = t_1 + 100$ with an elementwise XOR operation and subsequently counting the number of true values in the resulting array. Finally, the intermediate result is divided by the total number of cells in the simulation to get the rate of change in relative terms.

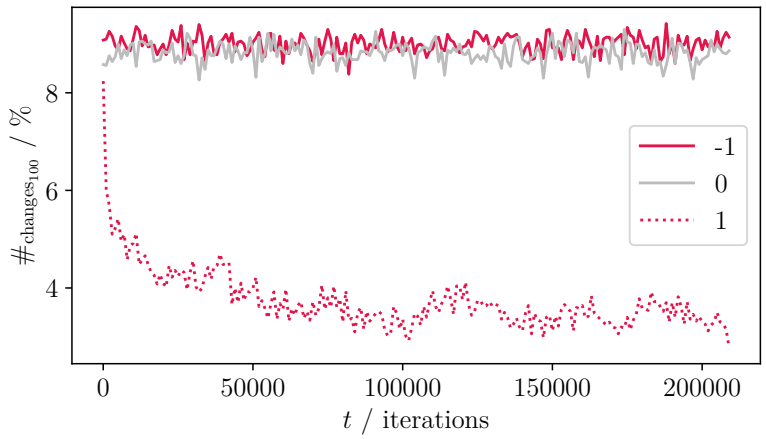


Figure 4.13: CA convergence starting from a random state. The random states are generated analogous to Figure 4.6 with 10% of the cells filled. Rate of change $\#_{\text{change}_{100}}$ is plotted against the number of iterations t.

From the curves in Figure 4.13 one can infer if a given value of λ leads to stable structures or random Brownian motion. While $\lambda = -1$ and $\lambda = 0$ remain at their initial rate of change – an indication of a stable state or random Brownian motion – $\lambda = 1$ shows a clear decrease in the rate of change. A strong indicator to suspect random Brownian motion instead of a stable state for the first two values of λ is the much higher rate of change in absolute terms in comparison to the simulation with λ set to 1. Furthermore, the decreasing rate of change for $\lambda = 1$ indicates that the simulation reaches a stable state. This hypothesis seems plausible and is in good agreement with the results from Figure 4.6 after $t = 1000000$ iterations.

With the same approach the densest circle packing geometry is tested for convergence, as shown in Figure 4.14. As expected from the results in Figure 4.2 setting $\lambda = 0.6$ erodes the structure and eventually evolves into behavior resembling Brownian motion. Contrary, a variable λ during the simulation – according to Equation 3.2 with parameters $d = 0.23$ and $h = 0.0002$ – mostly preserves the initial structures. After a peak in the rate of change at around $t = 10000$ iterations the rate of change decreases and eventually asymptotically approaches zero. This indicates that the simulation overwhelmingly consists of stable structures.

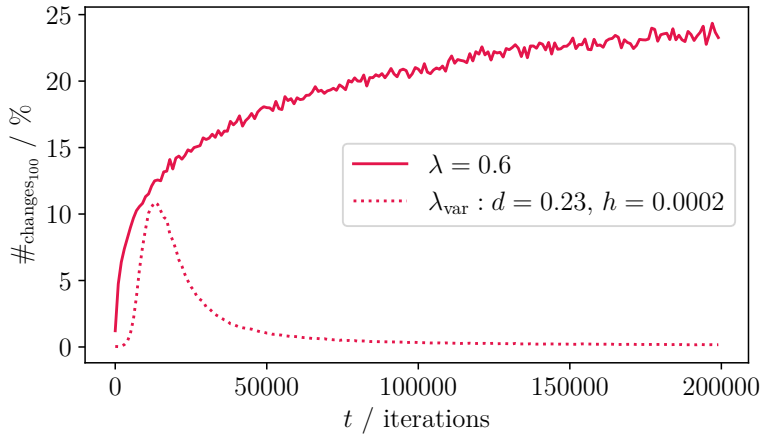


Figure 4.14: CA convergence for a densest circle packing with circle radius $r = 100$ and a circle thickness of 20 cells (comparable to the geometry in Figure 4.4). Rate of change $\#_{\text{change}_{100}}$ is plotted against the number of iterations t.

4.6 Performance comparison

Despite being regarded as a slow and memory intensive language, python remains a popular programming language, especially among the scientific community but not limited to it. Leveraging pythons vast range of different packages computation speed, as well as memory usage can be maximized and minimized respectively. [31, 32]
The CA presented in chapter 3 in and of itself is rather simple, and a single iteration can be calculated in fractions of a second using only python natively. However, for a larger number of iterations the speed, or lack there of, becomes apparent. Fortunately the CA can be parallelized, and certain functions can be just in time compiled to offer better performance.

The benchmark for the performance comparison was performed on two differently sized grids: A smaller 50×100 grid (BM (1)) and a larger 500×500 grid (BM (2)). Both were initialized randomly with 10% of the cells filled. Only the time of the RUN method – disregarding the setup and loading of the geometry – was measured up to various $t_{\max}$. The results for BM (1) are shown in Figure 4.15 and for BM (2) in Figure 4.16.
Additionally, the results from the two benchmarks are shown for all three different implementations in Table 4.1. The time for the jit compiled version is given with and without the compilation time.

While the runtime difference between a pure and a jit compiled version of the code were to be expected, NUMPY lists performing better than a jit compiled version is a bit surprising at first. This is most likely due to the fact that NUMPY uses C routines intern, which are highly optimized. Comparing the smaller to the larger test case, the observation can be

Table 4.1: Performance comparison between a bare python version and certain optimizations using NUMPY-lists and a jit compiler offered within the NUMBA package. Runtime is stated relative to the time a pure python implementation takes. Bm (1): Brownian motion on a 50×100 grid with $t_{\max} = 20000$ iterations, Bm (2): Brownian motion on a 500×500 grid with $t_{\max} = 5000$ iterations. Both measured in kilo-iterations ($= k\#$) per second. Uncertainty calculated via Student's t-distribution with 95% confidence interval.

test case / improvement	$k\#_{\text{Bm (1)}}$ / s^{-1}	$k\#_{\text{Bm (2)}}$ / s^{-1}
pure python (reference)	$4{,}24 \pm 0{,}03$	$0{,}066 \pm 0{,}005$
NUMPY lists	$19{,}8 \pm 0{,}3$	$0{,}41 \pm 0{,}03$
jit (with compile time)	$8{,}87 \pm 0{,}07$	$0{,}27 \pm 0{,}03$
jit (without compile time)	$14{,}45 \pm 0{,}17$	$0{,}29 \pm 0{,}03$

made that naturally the compile time of the jit-compiled version doesn't play a big role for larger grids.

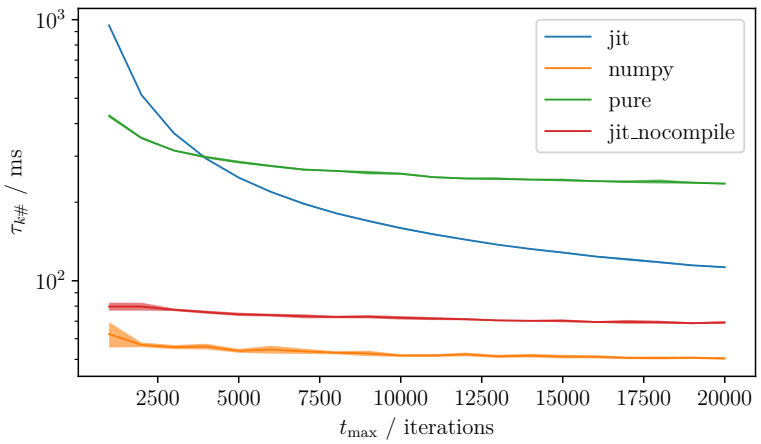


Figure 4.15: Benchamrk for different optimizations on a 50×100 grid with 10% of the cells having mass. Runtime per 1000 iterations ($\tau_{k\#}$) is plotted against the maximum number of iterations $t_{\max}$. Uncertainty is calculated per data point via Student's t-distribution ($n = 10$ observations) with 95% confidence interval. pure: using pure python, jit: jit compiled version, jit_nocompile: same as jit without compile time, numpy: using NUMPY lists

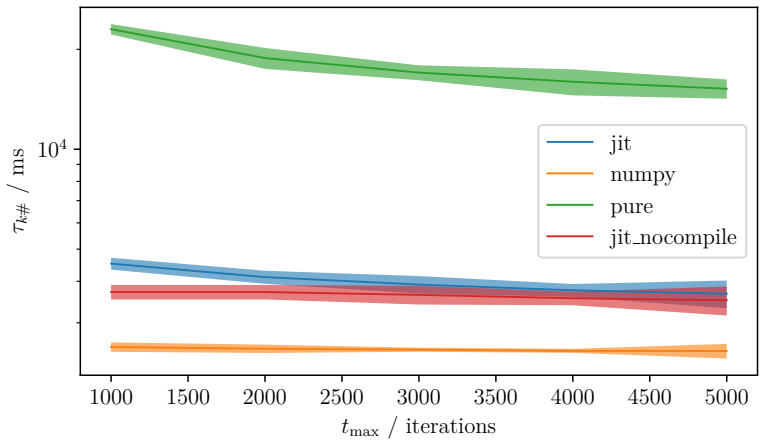


Figure 4.16: Benchamrk for different optimizations on a 500×500 grid with 10% of the cells having mass. Runtime per 1000 iterations ($\tau_{k\#}$) is plotted against the maximum number of iterations $t_{\max}$. Uncertainty is calculated per data point via Student's t-distribution ($n = 3$ observations) with 95% confidence interval. pure: using pure python, jit: jit compiled version, jit_nocompile: same as jit without compile time, numpy: using NUMPY lists

5 Discussion

Add discussion of the results

6 Conclusion

Add conclusion

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor's thesis.

Date	Signature

Bibliography

[1] A. Burks, ed. Theory of Self-Reproducing Automata. Urbana and London: University of Illinois Press, 1966. URL: https://cba.mit.edu/events/03.11.ASE/docs/VonNeumann.pdf (visited on 04/11/2023).

[2] M. Gardner. "MATHEMATICAL GAMES. The fantastic combinations of John Conway's new solitaire game "life"". In: *Scientific American 223* (Oct. 1970), pp. 120–123. URL: https://www.ibiblio.org/lifepatterns/october1970.html (visited on 04/10/2023).

[3] A. Schadschneider and M. Schreckenberg. "Cellular automation models and traffic flow". In: *Journal of Physics A: Mathematical and General* 26.15 (Aug. 1993), p. L679. ISSN: 0305-4470. DOI: 10.1088/0305-4470/26/15/011. URL: https://dx.doi.org/10.1088/0305-4470/26/15/011 (visited on 04/12/2023).

[4] S. H. White, A. M. del Rey, and G. R. Sánchez. "Modeling epidemics using cellular automata". In: *Applied Mathematics and Computation* 186.1 (2007), pp. 193–202. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2006.06.126. URL: https://www.sciencedirect.com/science/article/pii/S0096300306009295 (visited on 04/12/2023).

[5] Y. Ji et al. "Real Time Building Evacuation Modeling with an Improved Cellular Automata Method and Corresponding IoT System Implementation". In: *Buildings* 12.6 (June 2022), p. 718. ISSN: 2075-5309. DOI: 10.3390/buildings12060718. URL: https://www.mdpi.com/2075-5309/12/6/718 (visited on 04/12/2023).

[6] B. Chopard and M. Droz. "Cellular automata model for the diffusion equation". In: *Journal of Statistical Physics* 64.3 (1991), pp. 859–892. ISSN: 1572-9613. DOI: 10.1007/BF01048321. URL: https://doi.org/10.1007/BF01048321 (visited on 04/10/2023).

[7] H. Zahedi Sohi and B. Khoshandam. "Cellular automata modeling of non-catalytic gas–solid reactions". In: *Chemical Engineering Journal* 200-202 (2012), pp. 710–719. ISSN: 1385-8947. DOI: 10.1016/j.cej.2012.06.125. URL: https://www.sciencedirect.com/science/article/pii/S1385894712008558 (visited on 04/10/2023).

[8] A. Quell et al. "Creating Honeycomb Structures in Porous Polymers by Osmotic Transport". In: *ChemPhysChem* 18.5 (2017), pp. 451–454. ISSN: 1439-4235. DOI: 10.1002/cphc.201600834. URL: https://doi.org/10.1002/cphc.201600834 (visited on 04/10/2023).

[9] L. Koch, W. Drenckhan, and C. Stubenrauch. "Porous polymers via emulsion templating: pore deformation during solidification cannot be explained by an osmotic transport!" In: *Colloid and Polymer Science* 299.2 (2021), pp. 233–242. ISSN: 1435-1536. DOI: 10.1007/s00396-020-04678-5. URL: https://doi.org/10.1007/s00396-020-04678-5 (visited on 04/10/2023).

[10] A. W. Hauser, M. Schnedlitz, and W. E. Ernst. "A coarse-grained Monte Carlo approach to diffusion processes in metallic nanoparticles". In: *The European Physical Journal D* 71.6 (2017), p. 150. ISSN: 1434-6079. DOI: 10.1140/epjd/e2017-80084-y. URL: https://doi.org/10.1140/epjd/e2017-80084-y (visited on 04/10/2023).

[11] J. Von Neumann and A. W. Burks. Theory of Self-Reproducing Automata. University of Illinois Press series in automatic computation. Bibliography: p. 297-302. University of Illinois Press, 1966. URL: https://archive.org/details/theoryofselfrepr00vonn_0 (visited on 04/10/2023).

[12] P. Rendell. Turing Universality of the Game of Life. Ed. by A. Adamatzky. London: Springer London, 2002, pp. 513–539. ISBN: 978-1-4471-0129-1. DOI: 10.1007/978-1-4471-0129-1_18. URL: https://doi.org/10.1007/978-1-4471-0129-1_18 (visited on 04/11/2023).

[13] S. Wolfram. "Cellular automata as models of complexity". In: *Nature* 311.5985 (1984), p. 419. URL: https://content.wolfram.com/uploads/sites/34/2020/07/cellular-automata-models-complexity.pdf (visited on 04/11/2023).

[14] S. Wolfram. "Statistical mechanics of cellular automata". In: *Rev. Mod. Phys.* 55 (3 July 1983), pp. 601–644. DOI: 10.1103/RevModPhys.55.601. URL: https://link.aps.org/doi/10.1103/RevModPhys.55.601 (visited on 04/11/2023).

[15] S. Wolfram. A New Kind of Science. Wolfram Media, Inc., 2002. ISBN: 1-579-955008-8.

[16] L. Gray. "A Mathematician Looks at Wolfram's New Kind of Science". In: *Notices of the American Mathematical Society* 50.3 (2003), pp. 340–348. URL: https://www.ams.org/notices/200302/fea-gray.pdf (visited on 04/11/2023).

[17] T. Toffoli. "Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics". In: *Physica D: Nonlinear Phenomena* 10.1 (Jan. 1, 1984), pp. 117–127. ISSN: 0167-2789. DOI: 10.1016/0167-2789(84)90254-9. URL: https://www.sciencedirect.com/science/article/pii/0167278984902549 (visited on 04/11/2023).

[18] B. Denby. "Neural networks and cellular automata in experimental high energy physics". In: *Computer Physics Communications* 49.3 (June 1, 1988), pp. 429–448. ISSN: 0010-4655. DOI: 10.1016/0010-4655(88)90004-5. URL: https://www.sciencedirect.com/science/article/pii/0010465588900045 (visited on 04/11/2023).

[19] Y.-G. Yang et al. "Novel quantum image encryption using one-dimensional quantum cellular automata". In: *Information Sciences* 345 (June 1, 2016), pp. 257–270. ISSN: 0020-0255. DOI: 10.1016/j.ins.2016.01.078. URL: https://www.sciencedirect.com/science/article/pii/S0020025516300147 (visited on 04/11/2023).

[20] R. Feynman. “The Brownian Movement”. In: *The Feynman Lectures on Physics*. Vol. I. 1964, pp. 41–1. URL: https://www.feynmanlectures.caltech.edu/I_41.html (visited on 04/11/2023).

[21] A. Einstein. “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen”. In: *Annalen der Physik* 322.8 (1905), pp. 549–560. ISSN: 1521-3889. DOI: 10.1002/andp.19053220806. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19053220806 (visited on 04/11/2023).

[22] K. Pearson. “The Problem of the Random Walk”. In: *Nature* 72.1865 (July 1905), pp. 294–294. ISSN: 1476-4687. DOI: 10.1038/072294b0. URL: https://www.nature.com/articles/072294b0 (visited on 04/11/2023).

[23] P. Atkins and J. de Paula. Atkins’ Physical Chemistry. 9th. Oxford: Oxford University Press, 2011. ISBN: 9780199697403.

[24] L. Rayleigh. “On The Instability Of Jets”. In: *Proceedings of the London Mathematical Society* s1-10.1 (1878), pp. 4–13. ISSN: 1460-244X. DOI: 10.1112/plms/s1-10.1.4. URL: https://onlinelibrary.wiley.com/doi/abs/10.1112/plms/s1-10.1.4 (visited on 04/11/2023).

[25] P. Li et al. “Key factors affecting Rayleigh instability of ultrathin 4H hexagonal gold nanoribbons”. In: *Nanoscale Adv.* 2 (2020). This Open Access Article is licensed under a Creative Commons Attribution-Non Commercial 3.0 Unported Licence, pp. 3027–3032. DOI: 10.1039/D0NA00186D. eprint: https://doi.org/10.1039/D0NA00186D. URL: https://doi.org/10.1039/D0NA00186D (visited on 04/11/2023).

[26] B. L. Karihaloo, K. Zhang, and J. Wang. “Honeybee combs: how the circular cells transform into rounded hexagons”. In: *Journal of the Royal Society Interface* 10.86 (Sept. 6, 2013), p. 20130299. ISSN: 1742-5689. DOI: 10.1098/rsif.2013.0299. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3730681/ (visited on 04/12/2023).

[27] L. Prechelt. “An empirical comparison of seven programming languages”. In: *Computer* 33.10 (2000), pp. 23–29. DOI: 10.1109/2.876288. (Visited on 04/11/2023).

[28] Anaconda, Inc. Numba. https://numba.pydata.org/. Accessed on 2023-04-11. 2012–present.

[29] P. Grassberger. “Chaos and diffusion in deterministic cellular automata”. In: *Physica D: Nonlinear Phenomena* 10.1 (Jan. 1, 1984), pp. 52–58. ISSN: 0167-2789. DOI: 10.1016/0167-2789(84)90248-3. URL: https://www.sciencedirect.com/science/article/pii/0167278984902483 (visited on 04/13/2023).

[30] O. Bandman. “Simulating Spatial Dynamics by Probabilistic Cellular Automata”. In: *Cellular Automata*. Ed. by S. Bandini, B. Chopard, and M. Tomassini. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 10–19. ISBN: 9783540458302. DOI: 10.1007/3-540-45830-1_2.

[31] S. Overflow. 2022 Developer Survey Results: Top Paying Technologies, Other Frameworks and Libraries. URL: https://survey.stackoverflow.com/2022/#top-paying-technologies-other-frameworks-and-libraries (visited on 04/18/2023).

[32] P. S. LLC. PrimeView. URL: https://plummerssoftwarellc.github.io/PrimeView/?rc=50&rs=0&fr=&sc=dt&sd=True (visited on 04/18/2023).