Towards a framework for intuitive programming of cellular automata

by

Sami Torbey

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
December 2007

Abstract

The ability to obtain complex global behaviour from simple local rules makes cellular automata an interesting platform for massively parallel computation. However, manually designing a cellular automaton to perform a given computation can be extremely tedious, and automated design techniques such as genetic programming have their limitations because of the absence of human intuition. In this thesis, we propose elements of a framework whose goal is to make the manual synthesis of cellular automata rules exhibiting desired global characteristics more programmer-friendly, while maintaining the simplicity of local processing elements. We also demonstrate the power of that framework by using it to provide intuitive yet effective solutions to the two-dimensional majority classification problem, the convex hull of disconnected points problem, and various problems pertaining to node placement in wireless sensor networks.

Acknowledgements

This may be the hardest section to write in my entire thesis, as it is difficult for me to thank the people mentioned here enough for their tremendous impact on my studies and my life in general.

I will start by thanking my supervisor, Dr. Selim Akl, whose enthusiasm, knowledge, resourcefulness and ability to see the big picture and simplify difficult concepts deeply inspired my work as a graduate student - not to mention his occasional cracking-of-the-whip which helped me finish writing this thesis on time.

I want to thank Queen's University for welcoming me, funding me, and providing me with a great academic and extra-curricular learning environment during both my undergraduate and graduate studies.

I would also like to thank the members of my examining committee, Drs. Kai Salomaa, Robin Dawes, Ahmed Safwat and Gang Wu, as well as the members of my research group, the Parallel Computation Group at Queen's University, for thoroughly reading my thesis, attentively listening to my presentations, and offering me their invaluable comments and feedback.

Throughout my university career, several professors at the American University of Beirut (AUB) and at Queen's University have helped make computer science and mathematics fun for me. In particular, I would like to thank Dr. Louay Bazzi at AUB

through whom I discovered my passion for algorithms and the theory of computation.

Earlier this year, I started my first company, and on several occasions I felt like it was going to sidetrack me from completing my master's degree. I am especially grateful to my friend and business partner Mike Kulesza for preventing this from happening by understanding and supporting my decision to take some time off the company to write this thesis.

I would like to say a big "thank you" to all of my close friends and family in Lebanon, Canada and around the world, for the emotional support, camaraderie, entertainment, and care they provided. I want to thank Ramy Torbey for teaching me much of what I know, and for being a mentor and a friend in addition to being a brother, and Maya-Maria Torbey for being the best sister I could hope for.

Finally, my greatest thanks go to my parents Aziz and Marie Torbey. They bore me, raised me, supported me, taught me, and loved me. They pushed me to be my best when I was in doubt, and were the voice of reason when I got too carried away. It is to them that I dedicate this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cellular automata can be generally described as systems consisting of a large number of locally-interacting simple cells. This architecture is easily implementable and if harnessed appropriately, it can make a cellular automaton a formidable computation engine.

However, implementing desired high-level behaviour into low-level local cellular automata rules is a difficult problem. Our focus in this thesis is to add a powerful framework to the arsenal of cellular automata designers, allowing them to more simply and intuitively design cellular automata that perform some desired behaviour. We show that some ideas previously seen as "enhancements" to the traditional cellular automaton model are actually part of it; they are just a different, more designer-friendly way of thinking about its behaviour.

Having provided the main ideas behind our framework, we use it to design automata that solve three open problems: the majority classification problem in two dimensions, the convex hull of discrete points problem, and various problems pertaining to node placement in wireless sensor networks. These problems are simple

to describe but difficult to solve using cellular automata because of their requirement for global behaviour, despite the limitation of cellular automata to strictly local connections.

Finally, we learn from these solutions that computation in cellular automata is only in the eye of the beholder and cannot be isolated from the perspective from which it is meant to be seen; we also learn that simple "dull" cellular automata can indeed be "interesting" and better at solving some problems than their universal counterparts. To conclude, we provide some direction for future research in the area.

1.1 Contributions

In this thesis, our original results can coexist with previous work within the same chapter (although whenever possible we try to clearly demarcate our original work). We have therefore opted to add this section in order to help in the task of distinguishing our contributions from those of our predecessors. The thesis is organized as follows:

Chapter 2 mostly consists of background information which we had no role in creating although we present it in a manner to fit the purposes of this thesis. However, the chapter contains some glimpses of original ideas (usually preceded by the expression "we believe" or something of the like) which are revisited in other chapters; furthermore, the ideas for the quick universality proof of Rule 110 are ours.

Chapter 3 is also mostly based on the literature. Nevertheless, the direct comparison of cellular automata against Parallel Random Access Machines and Artificial Neural Networks is our original work. We also contradict Gutowitz and Langton's "interestingness" hypothesis [18], and show the limitations of both automated and

manual existing rule generation approaches.

Chapter 4 consists mainly of original work. Although some of the enhancements described (such as hierarchical and probabilistic cellular automata) have been previously presented in the literature under different forms, they have never been combined into such a complete framework designed to facilitate the design of cellular automata. In addition, they have never been shown to be capable of being implemented under the simple traditional cellular automaton model.

Chapter 5 is also mostly original work. We designed the convex hull solver from the ground up, although the second (simple) stage is partially based on Adamatzky's work [3]. In addition, the two-dimensional GKL adaptation, the non-local automaton approach and the two-dimensional perfect majority classifier based on Rule 184 are entirely our work. The remaining automata are merely presented to show previous results in the area. We briefly presented the results of Chapter 4 and the majority classification and convex hull results of Chapter 5 in [37]. The wireless sensor network node placement strategy and simulation also entirely consist of original work.

Finally, the recapitulation, open questions and philosophical discussion in Chapter 6 are original. However, the views that computation is in the eye of the beholder and that non-universal cellular automata are capable of computing were previously hinted at by Capcarrere [7].

Chapter 2

Background

2.1 Basic questions

2.1.1 What is a cellular automaton?

Generally speaking, a cellular automaton is a system consisting of a large number of simple processing elements (cells) locally interacting among themselves. The emphasis here is on the simplicity of individual elements, their connectivity and the absence of global control. This is a basic definition retaining only the elements essential for a system to be considered "cellular". However, cellular automata are often defined in far more detail, with many variations proposed since they were first introduced in the middle of the twentieth century. While we will contribute several such variations throughout this thesis, we will always refer to the definition above whenever we need to provide a basic sanity check of whether a proposed system can still be considered a cellular automaton.

Cellular automata are commonly seen as consisting of a "cellular space" and a set

of "transition rules" [28]. The cellular space is a set of cells, often shown in a given geometric configuration with respect to each other (usually a grid). Each one of these cells is a finite state machine in one of a constant number of possible states, evolving synchronously at discrete time units in parallel with all of the other cells in the system. The state of a cell at the next time unit is determined by a set of transition rules which are functions of the current states of cells in its neighbourhood (a finite set of cells connected to it, usually in its geometric vicinity). This neighbourhood often also contains the cell itself. Traditionally, all cells in the automaton have the same transition rules. However, there is an increased interest in non-uniform automata (see [34] for example).

In this thesis, we will mostly use finite grid configurations. This means that we will consider the number of cells to be finite but still large enough to clearly display complex behaviour (this will be defined in later sections), and we will show cells as small squares juxtaposed to each other in a row for one-dimensional grids or in a rectangle (usually a square) for two-dimensional grids. We now need to define two particularities of finite grid configurations: the neighbourhood radius and the boundary conditions.

Neighbourhood radius

The radius is a short-hand method of specifying the neighbourhood for common neighbourhood configurations. For example, in a one-dimensional cellular automaton of radius $r = 1$, each cell's neighbourhood consists of itself, one cell to its immediate left and one cell to its immediate right. In two dimensions, two configurations (shown in Figure 2.1) are commonly used: the von Neumann neighbourhood and the Moore

neighbourhood, differing in how they treat diagonally-connected cells. In von Neumann neighbourhoods, they are considered to be two distance units (one vertical and one horizontal) away from the main cell, whereas in Moore neighbourhoods they are directly connected to it (one distance unit apart). In other words, a two-dimensional von Neumann neighbourhood with $r = 1$ and including the cell itself contains five cells, while a Moore neighbourhood with the same specifications contains nine cells. Defined this way, the neighbourhood of a cell consists of all cells that are within a distance smaller or equal to r from it.

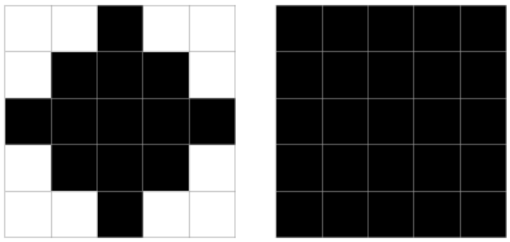

Figure 2.1: von Neumann (left) and Moore (right) neighbourhoods with radii 2.

Boundary conditions

When the grids are finite, boundary conditions become essential. They are meant to answer questions such as: "what is the left neighbour of the leftmost cell?" Periodic (also called "cyclic") boundary conditions are commonly used. They turn one-dimensional rows into circles (where the leftmost cell is connected to the rightmost cell), and two-dimensional rectangular grids into toroids (by connecting the leftmost column to the rightmost column and the topmost row to the bottommost row). An example of a simple cellular automaton with periodic boundary conditions is shown in Figure 2.2. Static boundary conditions are also common; in such a scheme, the boundaries are fictional cells that remain in a given state throughout the computation.

For example, in a two-dimensional cellular automaton with two states 0 and 1, static boundary conditions can dictate that there are two additional columns respectively to the left and to the right of the grid, and two additional rows respectively above and below the grid, where all the cells are in the state 0 throughout the computation. This means they participate in the computation only as passive neighbours to other cells without being themselves affected by the computation. Throughout this thesis, we will follow the convention of representing cellular automata with two states in black and white with white representing 0 cells and black representing 1 cells.

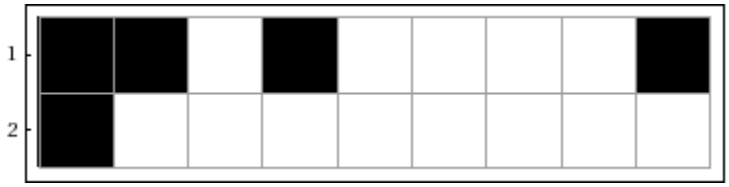

Figure 2.2: An interation of Wolfram Rule 128 on an example configuration of a one-dimensional cellular automaton with periodic boundary conditions. The Wolfram Rule notation will be defined in more detail in Section 2.2.1; for now, the reader just needs to know that under this transition rule, a cell becomes black if and only if it is itself black and both of its left and right neighbours are also black. The only cell satisfying this condition in the example above is the leftmost cell (because of the periodic boundary conditions). As such, all of the other cells become white.

2.1.2 Why study cellular automata?

Ilachinski presents four partially-overlapping motivations for studying cellular automata [19]:

Powerful computational engines

Cellular automata are inherently parallel. This makes their implementation in hardware interesting, with large numbers of simple locally-connected cells serving as a

faster alternative to complicated sequential processors. Several such implementations have been made, most notably the Cellular Automata Machines (CAM) series led by Norman Margolus at the MIT Information Mechanics Group. Despite sacrificing some of the performance benefits for programmability, Margolus showed in multiple instances (such as [26]) that CAM computation speeds are both theoretically and practically several orders of magnitude larger than their contemporary supercomputers. While we do not focus on hardware implementations of cellular automata, their use as an alternate computer architecture will be our main interest in this thesis.

Discrete dynamical system simulators

Cellular automata allow systematic investigation of complex phenomena by embodying any number of desirable physical properties. They enable the correlation of macroscopic continuous behaviour with microscopic discrete behaviour. Toffoli, a prominent researcher in this field, puts it best in [36]:

> *The features that one usually associates with physics (forces, pressure, temperature, magnetization, etc.) are not present as primitive features in the fine-grained recipe, which just deals with simple discrete tokens. They are derived features, which emerge at a macroscopic level when we start counting, averaging, blurring; in sum, looking at populations rather than individuals.*

Toffoli also outlines the special role played by reversible cellular automata (detailed later in this chapter) in this area: they allow a more accurate simulation of physical processes, which are assumed to be reversible.

Conceptual vehicles for studying pattern formation and complexity

Cellular automata are also used as a theoretical tool for studying complexity and emergence. Emergence (or complexity) can be defined as the capability of a system to display global behaviour greater than the sum of its parts. Certain cellular automata display an apparent remarkable capability to self-organize and generate complex behaviour starting from very simple rules. Wolfram has dedicated decades of work to classifying automata into different subjective complexity categories [39]. Other researchers such as Langton have attempted to objectively quantify overall system complexity [23]. Many have also been studying the relationship between microscopical dynamics and global emergent behaviour.

Original models of fundamental physics

Cellular automata allow studies of radically new discrete dynamical approaches to microscopic physics, exploring the possibility that elements of nature locally and discretely process their own future states. As explained in the previous section, some simple cellular automata are capable of arbitrarily complex behaviour. For example, Toffoli mentions the existence of discrete cellular automata capable of simulating continuous fluid flow despite the automata rules having never heard of the Navier-Stokes equations [36]. Some researchers such as Zuse [42] and Rucker [33] even go as far as suggesting that the entire universe is a giant cellular automaton.

2.1.3 What does it mean for a cellular automaton to compute?

It is generally agreed upon that a cellular automaton is considered to have terminated a computation when it reaches an appropriate termination condition (also called acceptance condition) as specified by the automaton designer. However, no single such condition applies to all automata. Therefore, the choice of a reasonable termination condition is usually left to the automaton designer. Sutner provides several examples of plausible conditions [35]:

- All cells are in a special state at some time t_0
- All cells are in a special state at all times $t \geq t_0$
- One particular cell is in a special state at some time t_0
- One particular cell is in a special state at all times $t \geq t_0$

All of these example conditions are suitable for detection by both machines and human observers. However, the first two are generally preferred for human observers, while the last two are easier for machines. In fact, many if not most computing cellular automata throughout history have been designed for human observers. The role of the automaton in such cases is to reorganize or modify the input information to make the desired result stand out for a human observer. The second and fourth example conditions assume that the observer knows when the automaton has converged, which may not be obvious at all. Capcarrere goes even further to argue that computation only happens in the eye of the beholder [7]. We agree with this statement given that emergence, considered the holy grail of computation in cellular automata, can only be

perceived by an observer with a certain point of view. The automaton cells themselves only see local microscopic information, even after the acceptance condition is reached. We will argue this point further when we explain the majority problem.

2.2 Classification

There are many classification systems attempting to group together cellular automata behaving similarly. We will detail the Wolfram classification [39], arguably the most well-known system. We will also explain its limitations and briefly mention some of the alternative classifications. However, we must first define elementary cellular automata, the simplest non-trivial automata which were used by Wolfram to illustrate his four behaviour classes.

2.2.1 Elementary cellular automata

Elementary cellular automata are one-dimensional two-state automata with periodic boundary conditions and a neighbourhood radius of 1.

Each cell can be in either one of the two states, leading up to $2^3 = 8$ possible input configurations for a neighbourhood of three cells (including the cell itself). For each of these configurations, the result cell can also be in any one of the two states. This means that there are $2^8 = 256$ possible sets of transition rules to the system.

Wolfram notation

Elementary cellular automata can then be described unambiguously based on the rules they follow, using a notation coined by Wolfram. As shown in Figure 2.3 depicting

an example rule, the rule number describing an elementary cellular automaton can be obtained as such: after placing all possible input configurations in decreasing order, the rule number is the concatenated binary value of the outputs yielded by each one of these configurations.

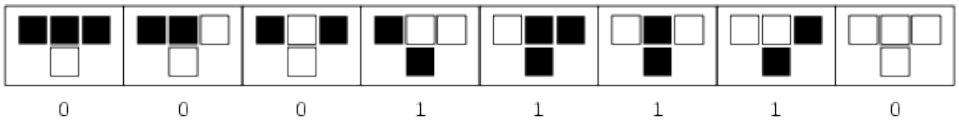

Figure 2.3: Transition rules of a Rule 30 elementary cellular automaton. It is called "Rule 30" because "11110" in binary notation is "30" in decimal notation. As a convention throughout this thesis, we will use a black cell to represent a "1" and a white cell to represent a "0" in two-state cellular automata.

2.2.2 Wolfram classes

After thousands of hours simulating countless selected and random rules on disordered initial configurations, Wolfram divided cellular automata into four classes depending on their prevalent practically observed long-term behaviour. While this classification is purely phenomenological and very qualitative, its simplicity and relevance makes it the most widely-adopted and well-known classification for cellular automata [17, 39].

Class 1

In cellular automata belonging to the first class, evolution on a random initial configuration leads to a homogenous state. Formally, every finite initial configuration evolves to a stable configuration in finitely many steps. Such automata exhibit repetitive behaviour, as shown in the example of Figure 2.4.

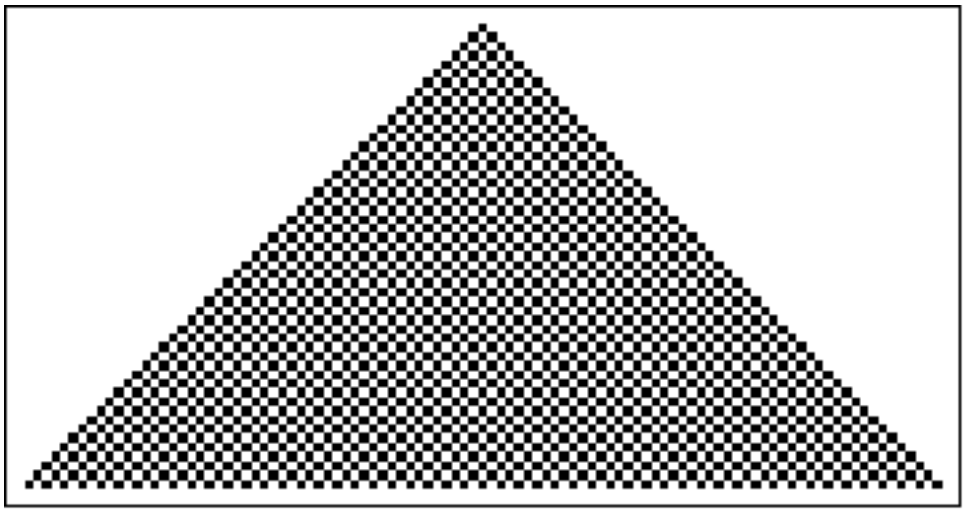

Figure 2.4: The Rule 250 elementary cellular automaton exhibits an infinitely-growing chequerboard pattern. Therefore, it belongs to Class 1. Note that one-dimensional cellular automata are often presented in two dimensions, with time on the vertical axis. Here, we show 50 successive iterations of Rule 250, beginning from the top row.

Class 2

In the second class, evolution on a random initial configuration leads to simple separated periodic structures (Figure 2.5); every finite initial configuration evolves to a periodic configuration in finitely many steps. This periodicity can also involve nesting structures. Class 1 and Class 2 automata are common particularly in elementary cellular automata where they constitute the majority [17].

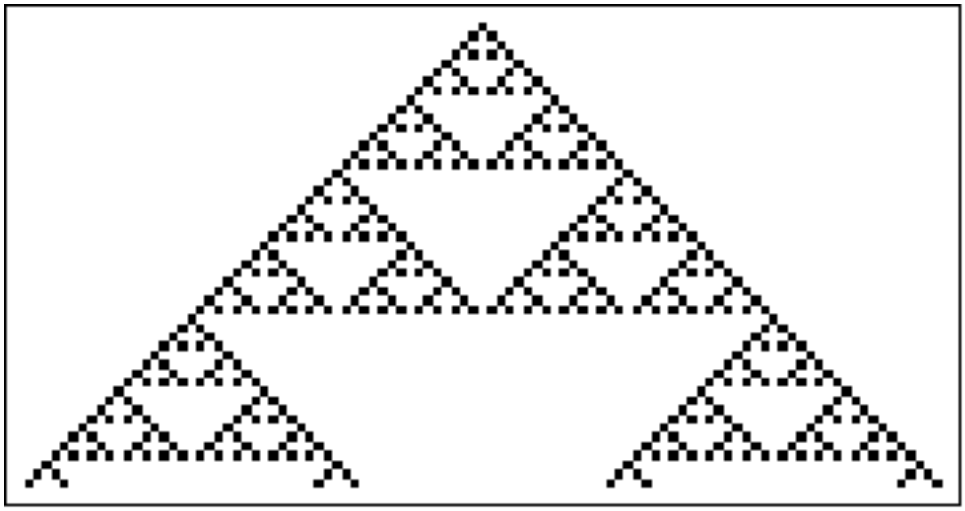

Figure 2.5: The Rule 90 automaton presents a fractal-like structure that is completely periodic. This is typical of Class 2 cellular automata.

Class 3

In cellular automata of Class 3, evolution on a random initial configuration leads to chaotic aperiodic patterns. An example of such automata is shown in Figure 2.6. This behaviour is similar to the chaotic behaviour found with strange attractors in dynamical systems. Put more simply, the behaviour of such automata is seemingly random; in fact, they are often used as random number generators. Class 3 is estimated to be the largest among the four classes (more than half of arbitrary automata but only 25% of elementary cellular automata).

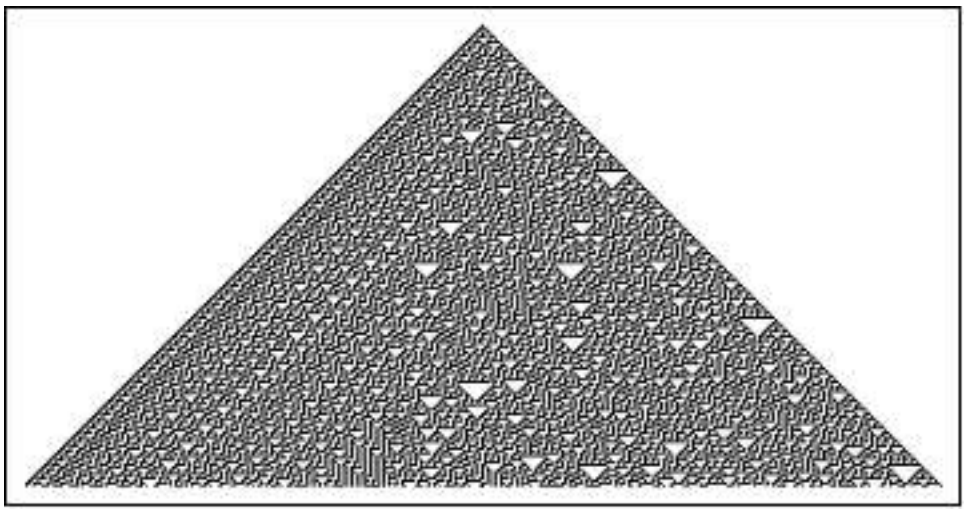

Figure 2.6: 200 iterations of the Rule 30 cellular automaton. Despite some regularity around the edges, its overall behaviour is seemingly random. As such, it belongs to Class 3.

Class 4

In the fourth class, evolution on a random initial configuration leads to complex patterns of localized structures (Figure 2.7). No similar behaviour has been found in continuous dynamical systems. Few automata belong to this class (less than 6% of various simple cellular automata as shown in [17]), but in general they are considered to be the most interesting ones to study.

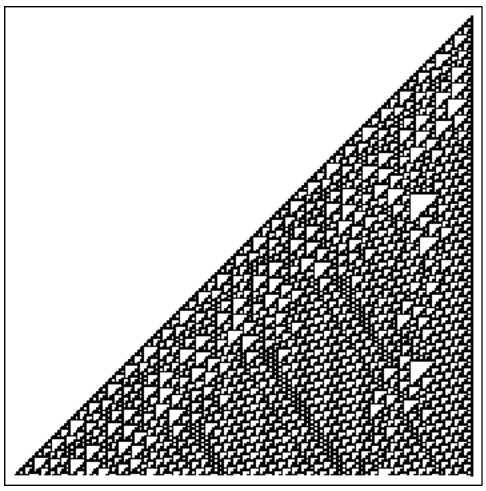

Figure 2.7: 200 iterations of Rule 110, which belongs to Class 4

Wolfram's classification has been criticized for being heuristic and not providing an objective quantification of "interestingness". It also does not differentiate between areas within an automaton. For example, some parts (such as the left side) of the diagram in Figure 2.7 representing the behaviour of Rule 110 seem to belong to a Class 2 rule.

2.2.3 Alternate classifications

These limitations have led other researchers to offer alternate classification systems, each with their own shortcomings. We present some of them in this section, which is largely based on [28].

Edge of Chaos

For example, Langton studied the relationship between the "average" dynamical behaviour (most likely behaviour obtained from a random initial sequence) of cellular automata and λ, a particular statistic of their rule tables defined as the fraction of non-quiescent states in the output (where one of the states is arbitrarily chosen to be the quiescent state) [23]. For example, in two-state cellular automata λ would be the number of rules whose output is a 1 divided by the total number of rules.

Langton performed numerous simulations intended to capture the "average" behaviour (most likely behaviour for randomly-chosen initial configurations) of a large number of automata belonging to each λ-space (sub-interval of λ values within the $\left[0; \frac{1}{2}\right[$ interval). Based on these simulations, he claimed that as λ increases, the average behaviour of cellular automata undergoes a "phase transition" from ordered behaviour (Wolfram Classes 1 and 2) to chaotic behaviour (Wolfram Class 3). Cellular automata having λ close to a critical value λ_c situated between orderly and chaotic behaviour tend to exhibit long-lived "complex" (non-periodic but non-random) patterns. Langton proposed that such automata roughly correspond to Wolfram's Class 4 and can be considered to be at the "edge of chaos". To quote Toffoli [36]:

> *Thus, between the immense area of boring grayness (chaos) and the small areas of dependable but not so exciting performance (e.g. waves),*

> *there are vanishingly thin areas in which something interesting happens where we had no "right" to expect it. There, an explanation for the existence of macroscopic texture will have to be sought case by case, and the challenge is to devise general rules to deal with the exceptions.*

Mitchell et al. re-examine the edge of chaos concept and discuss its problems in [29]. For example, they show that there is no single λ_c; instead, it depends on the particular computation desired.

Intrinsic computation

Crutchfield and Hanson took a different approach in [10], pointing out that attempts (like Wolfram's and Langton's) to classify cellular automata rules in terms of their "generic" behaviour are problematic, in that for many rules there is no generic behaviour either across initial configurations or even for the same initial configuration (there can be different dynamics going on in different parts of an automaton).

Instead, they developed techniques for classifying the different patterns that show up in cellular automata space-time behaviour. Their idea was to discover an appropriate "pattern basis" for each cellular automaton. This can be thought of as a regular expression specifying the background configuration of the automaton against which coherent structures evolve. Once the parts conforming to the pattern basis are removed from the general space-time configuration, if such coherent structures exist, they would be considered as the "interesting" part of the automaton.

An important aspect of this approach (which Crutchfield and Hanson call the "intrinsic computation" of a cellular automaton) is that it works for both ordered and chaotic configurations. For example, the Wolfram Rule 18 automaton which was

considered by Wolfram to be chaotic (Class 3) does in fact display coherent structures once the appropriate pattern basis is removed [28].

2.3 Universality

Although the Turing machine has been proven to be non-universal in [5], we follow the traditional definition of computation universality (more aptly referred to as "Turing universality") as being the ability of an automaton to simulate an arbitrary Turing machine. One common way to prove universality is using von Neumann's approach for the "universal copier and constructor": construction universality [17].

2.3.1 Construction universality

Construction universality requires the existence of a configuration c^0 with the following two properties [17]:

- c^0 is self-reproducing: that is, if at time $t = 0$ the initial configuration is c^0, then at some later time there will be two disjoint copies of c^0 in the current configuration of the host universe
- Upon being given another configuration x in its neighbourhood, c^0 will build a disjoint copy of x

Intuitively, c^0 can be seen as the equivalent of a logic gate. More specifically, it is a universal logic gate such as the NAND gate. In addition, c^0 needs to be self-reproducing so that it can create enough universal gates to implement any action required by an arbitrary algorithm. It should also be able to build a disjoint copy of one of its inputs; this means generating the output of the gate.

However, construction universality is not a necessary condition for universality. In fact, it is considered responsible for much of the complexity in von Neumann's original automaton. Perrier et al. show how a simple implementable self-reproducing cellular automaton can be capable of universal computation when the need for construction universality is forfeited [31].

2.3.2 Universal cellular automata

Researchers have not been able to prove general conjectures about universality in cellular automata. However, many automata have been individually proven to be universal. For example, Rule 110, one of the simplest non-trivial automata was shown to be universal by Cook in 2004 [9].

This and other universality proofs have laid the groundwork for more general statements. For example, it is now believed that most Class 4 cellular automata are universal, while there are no universal automata in other classes. This emanates from the fact that Class 1 and Class 2 automata converge to some known repetitive or periodic state irrespective of their input, i.e. after some transition period their output is mostly not a function of their input. On the other hand, Class 3 cellular automata exhibit pseudo-random behaviour no matter what their input is. Therefore, Class 4 automata are the most promising for universality; this can include automata such as Rule 18 which were thought by Wolfram to be in Class 3 but were later shown to have some of the Class 4 characterestics.

The Rule 110 universality proof also reinforced earlier beliefs that adding finite complexity beyond a certain point does not make a cellular automaton more powerful.

Conway's Game of Life

The Game of Life is widely credited for bringing cellular automata to the mainstream. It was invented by John H. Conway and popularized in the early 1970's by Martin Gardner in his "Mathematical Games" column in *Scientific American*.

It is a two-dimensional semi-totalistic cellular automaton with a unitary-radius Moore neighbourhood. In totalistic cellular automata, the state of a cell depends only on the average state of neighbouring cells, and not on their individual states. This means that the position of such cells relative to the output cell does not matter; all that matters is an aggregate number (usually the sum or average) of their states [39]. Semi-totalistic cellular automata are a variation on this where the state of the output cell itself in the previous iteration is considered towards the calculation of that aggregate; that is, the cell is in its own neighbourhood. The Game of Life has the following rules [17]:

- Any live cell with fewer than two live neighbours dies, as if by loneliness
- Any live cell with more than three live neighbours dies, as if by overcrowding
- Therefore, a live cell remains alive if and only if it has either two or three live neighbours
- Any dead cell comes to life if it has exactly three live neighbours

The key to the Game of Life's universality is a multi-cell structure called "glider". As their name implies, gliders are structures that move on the two-dimensional space through translation. They are considered to be the "bits" of the Turing machine within the Game of Life. Their behaviour becomes more interesting when they collide and thus interact; depending on how the collision happens, they can form "glider

guns" shown in Figure 2.8 (first discovered by Bill Gosper at the Massachusetts Institute of Technology) which emit gliders periodically until disrupted by an external object. The gliders emitted by the glider guns can themselves collide and form other glider guns. This proof idea makes sense given its similarity to construction universality. A more detailed universality proof sketch describing logic gates and wires is provided in [19].

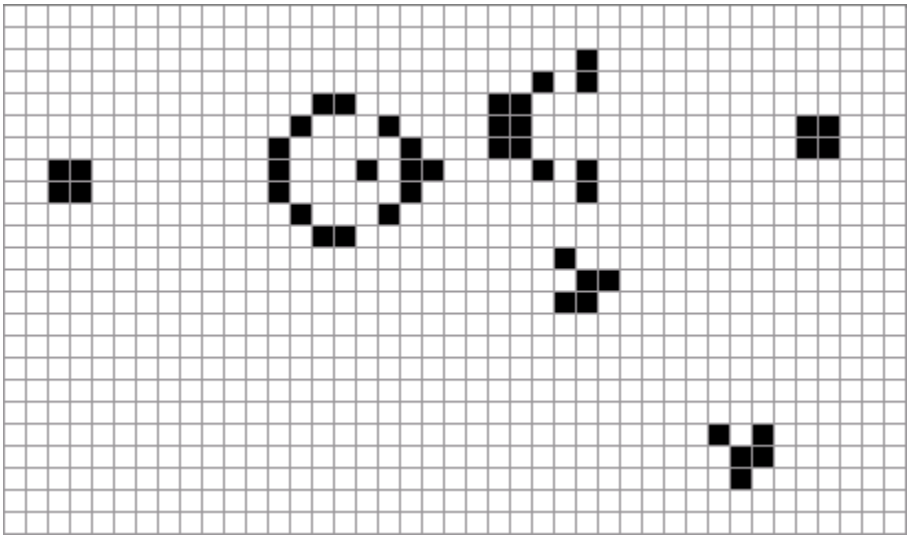

Figure 2.8: Game of Life configuration showing Gosper's glider gun with two emitted gliders (the two bottommost five-cell structures)

The Game of Life is very interesting from a theoretical standpoint. However, despite its name its behaviour does not closely mimic that of a population in real life (this may not have been Conway's original purpose). Some of our colleagues have made attempts at tweaking the Game of Life in order to better simulate real life by modeling human interaction [13] and enabling multiple populations [11].

Rule 110

Despite its simplicity, the Rule 110 cellular automaton can simulate an arbitrary Turing machine. Cook showed that it is universal by proving it equivalent to cyclic tag systems (Cook's rather complicated proof is described in [9]). Here we provide the beginning of a potential proof that may end up being simpler than Cook's. One way to show that a system is universal is to show that it supports a universal set of logic gates, wires (connections), bridges (allowing wires to cross each other without intersecting), and some method to hold an input for an arbitrary number of cycles [19]. We notice that Rule 110 is equivalent to the following boolean expression:

$$\rho\left(x, y, z\right) = \left(\overline{x} \wedge y\right) \vee \left(y \oplus z\right)$$

We can easily see that if we set $y = 1$ and apply De Morgan's theorem, ρ becomes a NAND gate having x and z for inputs. The NAND gate is known to be a universal logic gate, and as such it can simulate a NOT gate. By applying an even number of NOT gates on an input, we can hold it for arbitrarily many cycles. With this quick proof sketch, we have shown that Rule 110 satisfies two of the conditions listed above. We leave to the reader to prove that it can support wires and bridges.

The universality of the Game of Life and Rule 110 proves that additional complexity in cellular automata (such as more states or larger neighbourhoods) does not necessarily make them more powerful.

2.4 Reversibility

A cellular automaton is said to be reversible if for every current configuration of the cellular automaton there is exactly one past configuration. If one thinks of a reversible automaton as a function mapping configurations to configurations, this function must be bijective [17].

Cellular automata do not have to be reversible in order to be computationally universal. In fact, the Game of Life is irreversible. A more basic example is the NAND logic gate, which is universal but not reversible. For universal and reversible computing, check the Fredkin gate and the billiard-ball computer mentioned in Fredkin and Toffoli's seminal 1982 paper [14].

For one-dimensional cellular automata, there are known algorithms for finding past configurations, and any rule can be proven reversible or irreversible. For cellular automata of two or more dimensions, Kari showed using Wang tiles that the reversibility is undecidable for arbitrary rules [20].

For finite cellular automata that are not reversible, there must exist patterns for which there are no previous states [17]. These patterns are called Gardens of Eden, an example of which is shown in Figure 2.9.

Figure 2.9: The smallest known Garden of Eden in the Game of Life [1]

Several techniques are available to explicitely construct reversible cellular automata with known inverses. These are used by researchers such as Toffoli to design

cellular automata simulating physical rules, since such rules are assumed reversible. Another interesting property of reversible cellular automata is that they can be considered to have infinite memory (of the previous states). However, the further the desired state, the more time is needed to retrieve it.

Chapter 3

Computing with cellular automata

In the previous chapter, we have defined cellular automata in general along with some of the relationships between their dynamics and computation. However, we will see that while these relationships are interesting to study, they are not necessarily efficient ways to compute using cellular automata. Instead, our main focus will be on how cellular automata can help generate fast, practical, parallel computers.

3.1 Computational aspects of cellular automata

Capcarrere presents four ways to view the computational aspects of cellular automata (see [7] for references and examples):

3.1.1 Abstract computing systems

Cellular automata can be studied as abstract computing systems, similarly to Turing machines or finite state automata; the focus here on what languages they can accept, time and space complexity, undecidable problems, and other mathematical aspects.

3.1.2 Simulation

Another approach is to develop certain structures inside the cellular automata to allow them to simulate existing universal computers such as the Turing machine. Both the Game of Life and the Rule 110 cellular automata fall under this section if used as universal computers. These simulations are interesting from a theoretical standpoint but not for practical computation using cellular automata. In fact, even real-time Turing machine simulations - where one step of the Turing machine is simulated by only one cycle of the automaton (unlike in the Game of Life) - are considered inefficient because of the large number of parallel processors involved to simulate one sequential device.

3.1.3 Computing black boxes

Cellular automata can also be seen as computing black boxes whose input data is their initial configuration and whose output is given in the form of some spatial configuration after a certain number of time steps. In such systems, computation occurs on collision of particles carrying information. Therefore, it is called particle or collision-based computing.

Despite the similarity of rules governing all cells, Capcarrere writes that computation can often be seen as happening only in some specific cells of the system, while other cells are considered particle transmitters; this means that parallelism is not exploited as such. However, there are exceptions where every cell in an automaton is part of the computation. We believe that this is only a matter of perspective, because some information transmittal is necessary for global behaviour to occur. For example, we challenge the reader to come up with a local automaton to compute an

exact convex hull of discrete points in two dimensions - as presented in Chapter 5 - in less than $\sqrt{n}$ cycles (where n is the number of cells in the automaton). This $\sqrt{n}$ is the time it takes for information to be transmitted vertically or horizontally across the automaton. Whether the cells in-between points are considered to be merely passing information or whether they are seen to be computing if they belong to the convex hull or not is in the eye of the beholder. However, we understand Capcarrere's view in the light of the cells which have already decided with certainty whether they belong to the convex hull. These cells can be seen as quiescent or inactive for the rest of the computation.

This is the form of computation where a cellular automaton is tailored to solve a specific problem, allowing it to achieve large performance gains as well as to present its output in human-readable form. For these reasons, it will be our main focus throughout this thesis.

3.1.4 Computational mechanics

A fourth way of studying computation in cellular automata is to consider their computational mechanics. This is detailed in the second chapter of this thesis under the "Classification" section, and concentrates on regularities, particles and exceptions arising in the spatial configuration of the cellular automata considered through time. This research really studies the dynamical global behaviour of the system without considering any particular computation.

3.2 Cellular automata compared to other parallel computational models

Following our detailed overview of cellular automata, we deemed appropriate a comparision of their characteristics compared to other parallel computational models. Having found no such direct comparision in the literature (except briefly in [19]), most of the conclusions in this section are ours.

3.2.1 PRAM and other conventional parallel computers

The Parallel Random Access Machine (PRAM) is considered the most powerful among conventional parallel computers. For more background information regarding this section, the reader is encouraged to consult [4].

One difference between cellular automata and conventional parallel computers is that in cellular automata, everything is integrated - the processor is the memory, the input and the output. The input is merely the initial state of the whole system, and the output the final state. Conventional parallel computers can be seen as a large number of Turing machines connected together. They retain the Turing machines' property of separating between the structural part (fixed) and the data (variable) [36]. Therefore, unlike cellular automata, they are unable to extend themselves or construct separate copies of themselves, which was the original purpose of cellular automata when invented by von Neumann.

Another difference is in the way algorithms are designed for cellular automata compared to conventional parallel computers. To quote Akl in [4]:

Given a problem to be solved in parallel, a parallel algorithm defines

> *how the problem can be solved on the given parallel computer, that is, how the problem is divided into sub-problems, how the processors communicate, and how the partial solutions are combined to produce the final answer.*

From this statement, we can understand that each one of the processors *globally* solves a part of the problem in question before the partial global solutions are combined. On the other hand, cellular automata require a completely different approach to ensure that emergent behaviour is observed from strictly *local* rules which can appear unrelated to the desired global behaviour. Therefore, in conventional parallel computational models complex global behaviour comes from complex building blocks, while such behaviour comes from very simple parts in cellular automata.

Even in interconnection networks - a conventional parallel computation model where each processor has its own memory - there is a certain expected "flow" of information. On the other hand, the cells in a cellular automaton do not know of such flow and execute the same rules with the same neighbours at every time unit.

3.2.2 Artificial Neural Networks

The general definition given in the beginning of this thesis applies equally well to cellular automata and Artificial Neural Networks (ANN or neural networks for short) - see [21] for more information on Artificial Neural Networks. Both systems consist of a number of locally-interacting simple elements aiming to generate complex global behaviour. Therefore, strictly speaking, neural networks can be seen simply as biologically-inspired cellular automata.

However, there are practical differences in how the two are generally used which

we will detail below. Recurrent ANN of one layer are virtually identical to totalistic cellular automata if such automata are constrained to only threshold-based rules and the neural networks are constrained to uniform input weights (ANN frequently use threshold rules with weighted inputs which are not common for totalistic cellular automata but can be simulated using non-totalistic rules). In addition, many neural network constructions are feed-forward (having non-recurrent components) and consist of several layers. These are conceptually similar to cellular automata with time-varying rules and neighbourhoods which are not very common, although we will show in Chapter 4 that they can be simulated with traditional cellular automata.

To illustrate this point further, Minsky and Papert showed in [27] that single-layer perceptrons - a type of neural network - are unable to perform linearly-inseparable tasks such as the XOR function (see [12] for an introduction to linear separability). We can prove that this is a limitation of the single threshold normally used in ANN by showing that even certain elementary totalistic cellular automata are capable of performing the XOR function. For example, Rule 90 is effectively an XOR of the left and right neighbours and can be seen as a totalistic rule which outputs a 1 if and only if the sum of the left and right neighbours is a 1; otherwise, it outputs a 0 when the sum is 0 or 2. This requires two thresholds; a feat not possible using single-layer perceptrons.

Neural networks also usually have designated input and output neurons, while every cell in a cellular automaton is the input in the first cycle and an output in every cycle (especially when the termination condition is reached). However, as with many of the other differences described here, this one is mainly a matter of perspective as the user cannot be prevented from only considering a few of the cells while disregarding

the rest.

Finally, neural networks are often instructed to "learn" (by varying thresholds or weights) during a computation. On the other hand, the rules of a cellular automaton are generally derived before it is run.

3.3 Generating "interesting" cellular automata

3.3.1 What do we consider "interesting"?

We use the term "interesting" after Gutowitz and Langton in [18]. They define "interestingness" as the ability of cells in an automaton to transmit information between them in a useful manner. They argue that all Class 1 and 2 cellular automata are "dull" because no such information is transmitted. They also consider most of Class 3 automata to be dull because although information is transmitted in them, it is not transmitted in a useful form.

We agree with Gutowitz and Langton's definition of interestingness and elaborate on it by considering any automaton that simulates a non-trivial system or performs some desired computation to be interesting. However, we disagree with their conclusion and show that even Class 2 automata can be interesting. In fact, we present in Chapter 5 a Class 2 cellular automaton that can perfectly solve the majority classification problem. A similar result was also presented by Capcarrere in [7]. This automaton solves the problem by sorting its input, and sorting is generally considered a computation and therefore interesting. However, the automaton reaches a periodic state after sorting which is expected given that the computation is over.

We believe that Gutowitz and Langton confuse computation with universality. A

computing device does not have to be universal to perform a desired computation. While we agree with the quasi-universal belief that only Class 4 and certain Class 3 automata (such as Wolfram Rule 18) - which should really be in Class 4 but were misclassified because their Class 4 behaviour was harder to identify - can be universal, we disagree with the opinion that no other cellular automata can be interesting.

In this thesis, we are mainly concerned with the design of cellular automata that behave as "computing black boxes" by performing some desired computation in an efficient way and present an easily-readable output. Devising local rules exhibiting a required global behaviour can be extremely tedious. Therefore, several approaches (both automatic and manual) have been created to make the designers' job easier. We present some of them along with their shortcomings in the following sections, after which we propose our novel approach aimed at mitigating some of these issues.

3.3.2 Automatic approach

Many researchers have worked on algorithms to automatically derive cellular automata rules exhibiting desired global behaviour. Such systems require as an input the automaton in several starting conditions and their corresponding desired termination conditions after a certain number of steps. Most algorithms also require additional input parameters such as the desired number of states and neighbourhood size, as well as the number of steps in which the computation should be completed.

One of these researchers is Andrew Adamatzky who wrote a book on the subject [2]. In his book, Adamatzky devises several theorems regarding the complexity of the automatic identification of desired cellular automata rules. For example:

Theorem Let U be a deterministic synchronous stationary d-dimensional cellular

> automaton without memory, consisting of n cells, each of the cells having q states and k neighbours. Then U can be identified completely by a Turing machine M in $\Omega\left(k^{1+\frac{1}{d}}q^{2k}\right)$ steps.

This is the simplest of the theorems in [2]. Like all other ones, it involves at least exponential running time in the neighbourhood size.

To mitigate this impracticality, other researchers proposed approximate methods with faster running times such as genetic algorithms (computational models of adaptation and evolution based on natural selection) championed by Mitchell [28] and Sipper [34] among others, and some deterministic approaches such as the one used by Rosin in [32] - which is essentially modified hill-climbing: the sequential floating forward search.

Despite being tractable, these approaches still require large amounts of processing power and deliver less accurate results than their exponential-time counterparts. However, this is not their main limitation. Instead, we believe their key shortcoming is the lack of human intuition: these methods only consider one accepting condition, their work is based on example inputs and outputs (some special cases might be missing), and they often require the neighbourhood size, the number of states and the number of cycles needed to reach the desired terminating condition to be known in advance.

These limitations mean that such methods are excellent for the cases where a solution is known to exist and details about it are known but writing every possible transition rule is tedious. On the other hand, in the initial stages where problems still require human intuition, these approaches fail to provide adequate solutions. For example, as we will explain in Chapter 5, the majority classification problem

with the traditional accepting condition is not perfectly solvable [22]. Gacs et al. manually came up with an approximate solution for it achieving 81.6% accuracy [16]. It took more than a decade of genetic programming attempts to obtain marginally better accuracy at 82.3% [6]. However, the problem becomes perfectly solvable with a different terminating condition as shown in [7] and [15]. Another example is the convex hull problem. Adamatzky used his automatic identification approach to find a perfect solution for it provided that the points are connected (the solution converts a concave polygon to a convex polygon instead of forming a polygon from a set of discrete points) [3]. Rosin automatically found an approximate solution also provided one starts from a polygon [32]. In this thesis we provide the first solution - which we found manually in just a few days - to the problem of finding the convex hull of *discrete* points in a cellular automaton.

3.3.3 Manual approach

An alternate approach is to come up with a manual method that allows devising local rules in a way that makes more sense to designers. This is the approach we take in this thesis, although we consider our work to be very different from existing work in the area. Existing work generally focuses on making a particular function easier to implement subsequently: a set of rules allowing for some input parameters is engineered with considerable difficulty the first time it is needed; when similar rules are required for other automata, all one has to do is modify the input parameters in a straightforward way for rules behaving according to specifications to be automatically generated. The problem with this approach is that no set of functions, no matter how large, is enough to characterize any arbitrary desired behaviour. If an undefined

behaviour (or some unexpected parameter change) is needed, the designer is left to start from scratch by devising every transition rule required.

Most previous research - such as work done by Turner et al. [38] and Capcarrere [7] - adopts a multi-layered approach, for instance a "surface" or "mobile" or "action" layer and an "environmental" layer. The two layers can communicate within one cycle and the environment layer helps the surface layer make the right decision. For example, if the function represents a number of moving objects interacting in a room according to Newtonian mechanics, designers can use the surface layer to easily explain the desired behaviour (e.g. "the ball moves vertically at two cells per cycle") while the environment layer takes care of settling conflicts (hitting the ceiling for example) by instantly messaging the surface layer, therefore allowing it to change its movement within the same cycle. This high-level informal specification is then converted automatically to a set of pre-specified conventional low-level cellular automata rules. However, problems occur when the required behaviour is not pre-specified and no function exists for it; in the above example, what if the designer wants to take gravity into account? The only solution is to create the desired function and its low-level equivalent, which is very hard. To quote Turner et al. [38]:

> *"The low-level system components cannot be systematically derived from the system specification. The components are fundamentally different from the overall system, and cannot be described using the same language concepts."*

This approach is akin to creating libraries of functions in traditional programming languages. However, if some required function is not part of the library, the designer is left with the only option of writing it in machine language. On the other hand,

our approach - described in the next chapter - can be compared to a higher-level programming language such as C allowing the designer to implement any desired function more easily without losing sight of the low-level details.

Both existing automatic and manual approaches do not leave a significant place for human intuition, and as such suffer from problems similar to those described in Gödel's Incompleteness Theorem: one cannot possibly have a complete set of functions or an automated device that generates every possible desired rule. Therefore, human intuition is always needed and our work is meant to facilitate the process of applying it to deriving cellular automata rules while still allowing the use of existing automatic and manual (function libraries) approaches whenever they are suitable.

Chapter 4

Proposed framework

4.1 Description

The framework we propose can be assimilated to a higher-level programming language for cellular automata that does not completely abstract the low-level details. Rules are there to stay, whether they are generated automatically, using function libraries, or manually. However, our framework effectively divides those rules into several parts that are each easily understandable and devisable by humans, although they can be complicated when taken as a whole. It then combines these parts automatically to form the complete rule understandable by a machine.

While many of the ideas we propose in the next section have been mentioned before in the literature, we group them together as a framework and customize or reframe each one of them to fit our purpose. More importantly, we show how every single one of them can be implemented using simple cellular automata by adding more information to the state and the transition rule. This implementation is done automatically by the framework's "compiler", allowing the user to focus only on the

general idea of the automaton where human intuition is still required. The goal from this implementation is to allow running any algorithm on a generic cellular automata machine in order to benefit from the performance gains provided by such hardware implementation without having to design specific hardware for every desired automaton (we briefly go over potential hardware implementations in Chapter 6). For example, we believe that we are the first to show that automata with variable rules or probabilistic automata can be indeed implemented using the simple traditional description of cellular automata without the need for external information.

In the next section, we detail some major elements of the framework (we call them "enhancements" because they have traditionally been considered to be outside the scope of simple cellular automata) while explaining some of their benefits and their implementation using traditional cellular automata, as well as providing a reality check on whether they can still be considered to belong to the general definition of cellular automata as provided in the beginning of this thesis. We then go over some improvements which we considered but decided to leave out of the framework because they violate some of the conditions established by that general definition. Finally, we briefly mention some usability recommendations that do not inflict any changes on the physical automata but would be beneficial to designers if implemented in some cellular automata development environment.

4.2 Major enhancements

Most of the features we describe in this section can be implemented using hierarchical automata. We therefore start by describing hierarchical cellular automata and explain how they can themselves be implemented using traditional automata. It is important

to note that such implementation (as described here) is not always the minimal one, but is fully usable and relatively efficient. It can be optimized at a later stage if enough information is known about the hardware it is designed to run on.

4.2.1 Hierarchical cellular automata and variable rules

Adamatzky defines hierarchical cellular automata in [2] as a finite sequence of one-dimensional deterministic cellular automata $U_0, U_1, \ldots, U_p$ where automaton U_i is controlled by $U_{i+1}$for $i = 1, \ldots, p-1$. U_p has no supervisor and evolves in the usual way for a cellular automaton. The transition vector is the juxtaposition of the rule outputs for neighbourhoods ordered in decreasing order similarly to the Wolfram notation; for example, the transition vector for Rule 30 is "00011110". In Adamatzky's scheme, the transition vector of automaton U_i is given by the current states of the cells in automaton U_{i+1}.

We provide a different definition which is broader yet simpler and easier to implement. In our framework, cells in U_i simply have cells in U_{i+1} as part of their neighbourhood - with U_i and U_{i+1} generally evolving under different rules. This allows U_{i+1} to dictate any rule to U_i (as in Adamatzky's model) but also enables it to take a lower profile when needed where cells in U_{i+1} are equally (or even less) important than cells in U_i. We also allow two-way communication (U_i can be in the neighbourhood of U_{i+1}), similarly to what Adamatzky defines as "implantable cellular automata". While only one-way communication is needed in most cases, there are certain instances such as cellular automata with memory (described later) where two-way communication helps. We still call these hierarchical cellular automata because conceptually U_{i+1} would still be governing U_i despite needing some feedback from it.

Usage

We consider hierarchical cellular automata to be the most important feature of our framework. Along with enabling other framework features, they have unlimited uses; one of these is to allow variable rules. For example, they can allow an automaton to completely change its rules after some time t_0 (time-variable rules). The way this is done is by having a progress-bar-like automaton such as Rule 238 (shown in Figure 4.1) as the parent automaton and connecting all cells in the child automaton to its t_0^{th} cell. If this cell is a "1", then $t \geq t_0$ and the new rules are used.

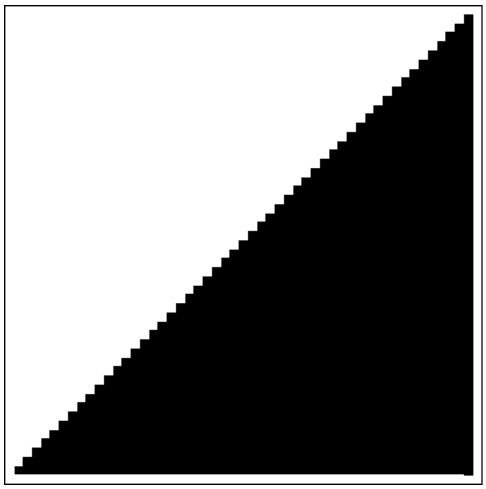

Figure 4.1: 50 iterations of Rule 238 showing its progress-bar-like behaviour

The rules can also change based on some other parameter: for example, when some kind of constant-size structure is reached in the child automaton; we stress the necessity of the constant size of the structure because otherwise the automaton would be considered non-local and would therefore violate the general definition. This is one of the cases which would require two-way communication.

The danger of having all cells connected to one cell is that this might seem to violate the locality condition. However, the neighbourhood size of every one of these

cells would still be constant (only one additional cell added to the neighbourhood) and not a function of the automaton size. In addition, they would all be making decisions based on their neighbourhood. Although connecting this one cell to every other cell might be complicated to implement in practice, we believe that this is not a major issue as long as the number of such cells is constant compared to the automaton size. In our opinion, the locality condition is only violated when a cell needs to read some value outside of its fixed constant neighbourhood in order to make a decision. We still consider this scheme to be local because many cells are reading one cell and not vice-versa.

Implementation

Hierarchical cellular automata can be easily implemented within one simple cellular automaton by taking the largest automaton in the hierarchy and embedding the rest within it by adding their state bits and transition rules to it. This means that two binary-state automata can be embedded within one four-state automaton.

4.2.2 Clocking

Another timing-related rule change is clocking, which has been previously used in Quantum-dot Cellular Automata (refer to [24] for more information on this topic). In such a scheme, all cells in the automaton do not update at the same time. Instead, different areas do so sequentially in order to allow some messages to be passed without being influenced by pre-existing conditions. This concept was also considered by Rosin [32] who proposed the idea of even-odd rules (different rules for even and odd cycles) without providing a means to implement such rules using traditional cellular

automata. Our framework can be used for this purpose and is not limited to two different cycles. We can have an arbitrary number of cycles n using a parent Rule 2 automaton (shown in Figure 4.2) having n cells and periodic boundary conditions. We recommend limited use of clocking rules because they curtail some of the parallelism benefits of cellular automata.

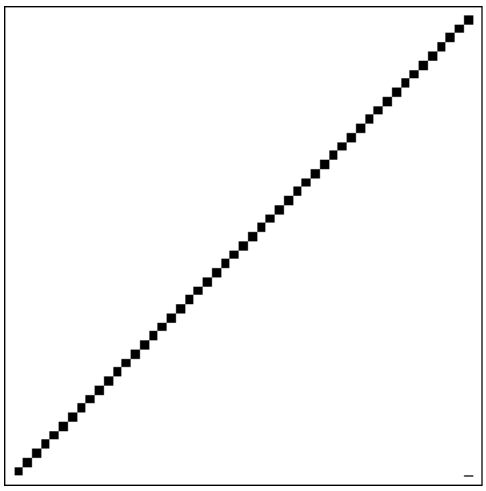

Figure 4.2: 50 iterations of Rule 2 provide 50 different clock cycles

4.2.3 Probabilistic rules

Probabilistic rules are yet another feature that makes use of hierarchical automata. Adamatzky defines probabilistic cellular automata in [2] by putting a certain probability on a set of transition rules for a given neighbourhood (instead of having just one transition rule per neighbourhood). He also proposes several classes of probabilistic cellular automata differing in the details.

We propose a simpler definition that is also more flexible and easier to implement. For the purposes of our framework, probabilistic rules are ones where at least one neighbourhood cell is in fact a coin flip (unbiased for simplicity, but this can differ in

the implementation). Using a number of such coins, we can implement any desired probability to an arbitrary precision and as such, simulate Adamatzky's system.

Usage

Probabilistic rules can be used similarly to their probabilistic algorithm counterparts. For example, they can provide adequate approximate solutions in less time than deterministic rules, or they can help escape local optima or deadlocks.

Implementation

We do not need an external coin or access to an external random variable. The coin flip can be implemented using hierarchical cellular automata by having a Wolfram Class 3 automaton as a parent and selecting a number of its cells to be in the neighbourhood of the cells in the child automaton. This works because Class 3 automata such as Rule 30 are chaotic and as such, they behave like pseudo-random number generators. In fact, Rule 30 is used to generate random numbers in Wolfram's famous software package Mathematica.

4.2.4 Cellular automata with memory

Also called multi-cycle automata or automata with history, they are considered by various researchers including Adamatzky [2]. In such cellular automata, the next state of a cell depends not only on the current state of its neighbours, but also on the previous states (up to a constant number of previous cycles) of some of its neighbours.

Interestingly, reversible cellular automata do not need to have their previous states stored. They can be retrieved from any current state, however the further in time the

desired state, the longer it takes to retrieve it. As such, reversible automata can be said to have unbounded memory.

Usage

Memory in cellular automata has many uses. Among them is estimating certain behaviour over time (derivatives such as speed or acceleration), or detecting deadlocks and backtracking, as we will show in one majority classification solution example in Chapter 5.

Implementation

Automata with memory can be easily implemented using hierarchical automata, which themselves can be implemented within one simple traditional automaton. In that case, the parent automaton is the previous state of the child automaton. This needs two-way communication and a very simple rule having a radius zero neighbourhood (only the cell itself).

4.2.5 Static and private state information

We propose allowing a part of a cell's state to be constant throughout a computation (such information is to be provided before the beginning of the computation). We also propose allowing a part of a state to be considered private (not part of the output) in order to facilitate discerning the solution for human observers.

Usage

The static part of a cell's state is used to give a cell more information about itself and its role in the computation. For example, this can be the cell's position within the cellular automaton (or relative position for periodic boundary conditions) which can help in many cases, such as when designing an automaton to find a centroid. The concept of static state information is closely tied to non-uniform automata detailed in the next section.

Keeping some of the state information private is important to allow better perception of the solution and to discern between temporary variables and ouput variables. For example, when attempting to compute the convex hull of a set of points, the observer only needs to know whether a given cell belongs to the convex hull or not (one state bit). However, the cells often require more information during a computation (more state bits) in order to be able to reach such a decision - it is this information that we desire to keep private. For example, one way to keep it private is to assign only two colours to a four-state automaton when visually displaying it.

Implementation

Both of these conditions are very easy to implement. Static information can be considered as variable information that is not modified by any of the transition rules, while certain state bits can be ignored in the output in order to keep them private. Static information can also be implemented using static parent automata (with idle transition rules) such as Rule 204 shown in Figure 4.3.

Figure 4.3: 50 iterations of Rule 204

4.2.6 Non-uniform automata

Non-uniform cellular automata are automata where some cells follow different transition rules from each other. They are widely documented in the literature; for example, Sipper provides several methods to automatically design such automata efficiently using genetic programming [34]. We believe that non-uniform automata fit within the general definition of cellular automata since the individual elements are still simple and can often be easily reprogrammed to perform different functions when different automata are being run on the same hardware (we provide an example hardware implementation in Chapter 6). However, we also believe that it is practically hard to

design an automaton where every cell has different transition rules, unless these rules differ only by a certain value (for example a threshold in totalistic cellular automata) that can be automatically programmed.

Usage

A more practical application is having several blocks of cells each with their own rules. For example, when designing an automaton that simulates a bouncing ball inside a maze, the walls which are fixed cells need to follow different rules than the free-space cells where the ball is allowed to pass.

Implementation

Non-uniform automata can be simply implemented using uniform automata with static state information telling the cell which set of rules it should follow. They can also be directly implemented in hardware if this is supported.

4.3 Forgone enhancements

We also considered one other enhancement that would add additional power and convenience to the design of traditional cellular automata. However, we have decided to leave it out of the framework because it clearly violates the locality condition of the general definition of cellular automata.

4.3.1 Non-local automata

Non-local automata have been widely studied (reference [25] provides more information). These automata contain some cells having either a large number of neighbours (a function of the automaton size), or a "pointer" cell in their neighbourhood. Pointer cells are cells which contain the location of another cell (the one they point to) and return its content when read. The transition rules of pointer cells specify the location of the cell they point to, allowing it to change during the computation depending on their neighbourhood. Effectively, pointer cells allow the re-routing of a small number of connections and as such, allow any cell to have any other cell as its neighbour during the computation without needing a large neighbourhood size.

While such a scheme might have practical uses such as the simulation of some natural systems which are believed to be non-local, we believe it violates one of the key conditions of cellularity. It is also difficult to implement in hardware unless some hardware implementation allowing instant re-routing of connections without major compromises is developed. As such, it does not fit the goals of our framework.

4.4 Usability recommendations

4.4.1 Pseudo-totalistic rules

One thing we found very useful in developing the example automata mentioned in Chapter 5 is the introduction of pseudo-totalistic rules. While they do not modify simple automata in any way, they would be a welcome addition to a cellular automata development environment. We define pseudo-totalistic rules as logical conjunctions of partial totalistic rules. They aim to reduce the total number of rules required to

describe a system. For example, a three-state automaton with a neighbourhood of size 9 presents $3^9 = 19683$ possible neighbourhood states and 3^{19683} possible rules. In most cases, the rules follow similar patterns and not all possible neighbourhood states should be specified individually. Pseudo-totalistic rules allow specifying the system more simply by enabling the designer to provide statements such as "if the sum of the top 3 cells is larger than the sum of the bottom 3 cells and the sum of the middle 3 cells is 0 then the next state is 2". The system can then easily fill-in the blanks and derive the low-level rules by executing such statements against all possible input conditions.

4.4.2 Automatic identifiers and function libraries

Another useful feature allows easy integration of automatic rule identifiers and function libraries such as those defined in Chapter 3. These can be used to fill-in the details after the work requiring human intuition is done. For example, in the "gravity" automaton we describe to solve the majority classification problem in Chapter 5, we could have simply used a pre-defined function implementing Newtonian mechanics to define the rules of the automaton instead of having to take every possibility into account (which can be difficult especially when the next state of one cell influences the next state of another - this requires enlarging the neighbourhood size and adding many duplicate rules).

Chapter 5

Examples

In this chapter, we present three problems that although simple in their description, are considered complicated for cellular automata given their requirement for emergent global behaviour. We explain existing solutions to these problems as well as provide our own improved solutions reached using our framework.

5.1 Majority

The majority (also known as density classification) problem is arguably the most studied problem in cellular automata theory. It can be described as follows: given a two-state (black and white) cellular automaton, does it contain more black or white cells? The accepting condition can vary as long as it provides a clear-cut answer. However, the most frequently used (and the first proposed, in [16]) accepting condition requires that the whole automaton converges to the more dense colour, i.e. all cells become black if black cells initially outnumber white cells, and vice versa.

Land and Belew proved in [22] that a two-state automaton with local neighbourhoods can never converge 100% accurately to such an accepting condition. However, the search is still on for the rule with the highest accuracy, and at the time of writing, such a rule is believed to be a soon-to-be-published rule discovered through genetic programming in 2007 by Wolz and de Oliveira [40]. This rule achieves an 89% accuracy for a large number of random initial conditions on a one-dimensional lattice of size 149 with a neighbourhood of radius 3 (by convention these are the conditions normally used to easily compare the efficiency of such algorithms; larger lattice sizes generally reduce the accuracy). In two dimensions, the best rule at the time of writing achieves 83% accuracy [40]. Later in this section, we show that a change in the accepting condition makes a 100% accurate convergence possible.

We now present a few approaches to solving the density classification problem:

5.1.1 Normal threshold

This is considered the most obvious solution; however, it rarely works since the automaton does not usually converge. In such a solution, the next state of a cell is determined simply by the majority of the cells in its neighbourhood. Traditional two-way neighbourhoods are used. In elementary cellular automata, such a solution would be Rule 232. In two dimensions, a Moore neighbourhood of radius 1 (and including the cell itself) is usually taken in which a cell becomes black if and only if five or more of its neighbours are black.

We believe that the deadlock reached by such automata (to some intermediate state that looks like the pattern on a cow's skin) is largely due to the mutual influence exercised by neighbours on each other. For example, in a Moore neighbourhood with

$r = 1$, each cell can have up to six out of nine neighbours in common with its neighbours. All cells also follow the same threshold rule which is reinforced in the next states with more and more cells stabilizing on one colour.

5.1.2 Vichniac rule

The Vichniac rule is a simple rule aiming to overcome that deadlock. It is effective in many cases and works by switching the behaviour at boundary conditions. For example, in a two-dimensional Moore neighbourhood of radius 1 this means that a cell becomes black if and only if 4, 6, 7, 8 or 9 of its neighbours are black; it becomes white otherwise.

5.1.3 GKL

A seminal 1978 paper by Gacs et al. presented a breakthrough in solving the majority problem [16]. That breakthrough which became known as the GKL rule (after the last names of its authors) is a one-dimensional automaton of radius 3 with quasi-totalistic rules. It was shown 81.6% effective on a large number of random initial configurations of a one-dimensional automaton of size (number of cells) 149. This test became the standard against which the effectiveness of other automata (such as the genetic improvements presented in the next section) is measured.

Under the GKL rule, if a cell is black, its neighbourhood consists of the cell itself, the cell immediately to its left and the cell three spaces away from it also to the left (Figure 5.1). Then a simple totalistic rule is applied where the majority in the neighbourhood determines the next state of the cell. On the other hand, if the cell is white, then its neighbourhood has the same configuration except that this time only

the cells to its right are taken into account.

Figure 5.1: The black cells in this figure represent the middle cell's neighbourhood when it is black. Its next state is the same as the majority among them.

We believe that the key to GKL's success is the minimal overlap between neighbouring cells' neighbourhoods. This makes information more likely to flow inside the system instead of having neighbours highjack each other's next states, forcing the automaton into a deadlock.

5.1.4 Genetic improvements

Ever since the GKL rule was introduced, researchers have been trying to devise rules that are more effective at classifying random initial configurations. Most of these attempts are automated and use genetic algorithms to select the best rules from a very large number of possibilities. At the time of writing, the best such rule is presented in [40]. It has a neighbourhood of radius 3 and is 89% effective on initial configurations of size 149.

We argue that while better rules can be found using such approaches, the manual approach can be more effective if more insight is gained into the long-term behaviour of cellular automata. For example, the best known rule introduced almost two decades and many researcher and computer hours after GKL was only 82.3% correct [6], a marginal improvement. In addition, such genetic algorithms use GKL as their basis; rules are generally considered "better" and thus preserved by the genetic algorithms when they are more similar to GKL. This makes reaching a breakthrough with such

automated methods not possible.

5.1.5 Two-dimensional GKL adaptation

We experimented with adapting GKL to bi-dimensional cellular automata. We used a radius 3 von Neumann neighbourhood - although a Moore neighbourhood can be considered by adding diagonals for probably better results - and added to the traditional one-dimensional GKL black cell's neighbourhood the cells immediately above it and three steps above it (for a total neighbourhood size of 5). We did the same to the neighbourhood of white cells by considering the cells below them as well (Figure 5.2).

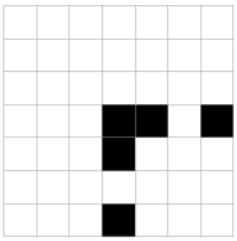

Figure 5.2: Neighbourhood of the middle cell when it is white

Reverse annealing

While the two-dimensional GKL adaptation behaved very similarly to the one dimensional GKL rule by having large masses of chequerboard-like patterns erase black or white islands (clusters of cells), it frequently did not converge and got stuck in a deadlock of period 2. We resorted to the proposed framework to reduce the likeliness of that deadlock by adding memory capability that effectively allowed the system to

behave conversely to simulated annealing: each cell checks a constant number (parameter of the system) of previous cycles of itself and its neighbours for deadlock patterns. For each previous cycle that exhibits such patterns, a coin toss is added to the cell's neighbourhood; this means that the more a cell considers itself to be in a deadlock, the less it relies on its traditional neighbourhood by adding more and more random cells to its neighbourhood.

This method - which we call "reverse annealing" - allowed us to increase convergence and success rates significantly. Table 5.1 shows the results of a preliminary study we did on a 100×100 cellular automaton with twenty initial configurations for each of the two-dimensional GKL rule and the two-dimensional GKL rule with reverse annealing.

	Standard 2D GKL	2D GKL with reverse annealing
Correct	65%	85%
Incorrect	15%	10%
Did not converge	20%	5%

Table 5.1: Results of a preliminary study comparing two-dimensional GKL results respectively without and with reverse annealing

While twenty initial configurations are not statistically significant, these numbers provide an idea of the results to be expected in a larger-scale study.

5.1.6 Non-local automaton approach

GKL and its derivative rules are effective on random initial configurations. However, they are all vulnerable to specific patterns of initial configurations where their success rates can drop to below 50%. Here we propose a method to overcome that vulnerability using non-local cellular automata. The key is to provide randomization

somewhere in the process. In this case the neighbourhoods are chosen at random at every step, making a result manipulation using specific initial configurations impossible. The rate of effectiveness of this method is always above 50%; it increases predictably with the size of the neighbourhood chosen and the discrepancy between black and white cells in the initial configuration.

In this method, each cell randomly chooses a constant odd-size neighbourhood at every step; this neighbourhood determines the cell's next state using a simple majority rule. While non-local automata are not part of the framework described, in this particular case they can be easily and reasonably implemented using local automata. This is done by noticing that only a very small number of iterations is needed before the accepting condition is reached. This number is determined by the closeness of the densities of black and white cells, which can be related to the automaton size in general. For example, in a 1,000-cell automaton, the most difficult initial configuration is the one having 501 cells in one state and 499 in the other. In a 10,000-cell automaton, this is the configuration with 5,001 cells in one state and 4,999 in the other. This means that in the first case, the difference is $\frac{2}{1{,}000}$ cells while in the second it is $\frac{2}{10{,}000}$. The good news is that for a fixed neighbourhood size of more than 10 cells for both cases, the second automaton is expected to need only one extra cycle before it reaches its accepting condition. Therefore the number of cycles required for convergence can be seen as growing at most logarithmically with the size of the automaton.

This allows simple implementation of that particular non-local automaton in a local automaton. This implementation is done by selecting at random a neighbourhood of size the expected number of cycles times the neighbourhood size in the non-local

automaton. For example, if a non-local automaton with a neighbourhood size of 5 is expected to converge in 3 cycles, the neighbourhood size of the local automaton should be 15. Then, using changing rules as defined in this framework (hierarchical automata), each cell can select 5 neighbours among the 15 at every iteration. In fact we do not even need to multiply by the expected number of cycles. A constant multiple should be enough regardless of the automaton size since the discrepancy between the numbers of black and white cells is greatly increased at each iteration, making the task of classification much simplified after a couple of iterations. Each cell can select its neighbours at random among a multiple of the non-local neighbourhood size at every iteration, and some overlap between iterations after the first few cycles rarely influences the outcome.

The key however remains in non-uniform neighbourhoods (von Neumann, Moore and other neighbourhoods that apply to every cell cannot be used) where each cell's neighbourhood structure is different from the others and picked at random when the automaton is defined. This means that while all cells have equal neighbourhood sizes, the position of such neighbours relatively to the cells vary for each cell and are picked randomly. Then each cell randomly chooses an odd number of neighbours that influence its next state at every iteration, and follows a simple majority rule to decide on that next state.

5.1.7 Rule 184

Although Land and Belew proved that a two-state cellular automaton cannot perfectly perform the majority classification task under the all black or all white accepting condition [22], their proof does not necessarily hold under other accepting conditions.

For example, Capcarrere showed that an elementary cellular automaton (Rule 184) of radius 1 was perfectly capable of performing that task under a different termination condition [7]: if there are two consecutive black cells anywhere in the automaton after n cycles - where n is the lattice size i.e. the total number of cells - then black cells have the majority. Conversely if there are two consecutive white cells then white cells have the majority. If none of these conditions is true (the automaton is a perfect chequerboard) then both colours are in equal numbers. Capcarrere also showed that under Rule 184, having both consecutive black and consecutive white cells in the automaton after n cycles is impossible. In Figure 5.3, we show an example of several iterations of Rule 184 leading up to its accepting condition.

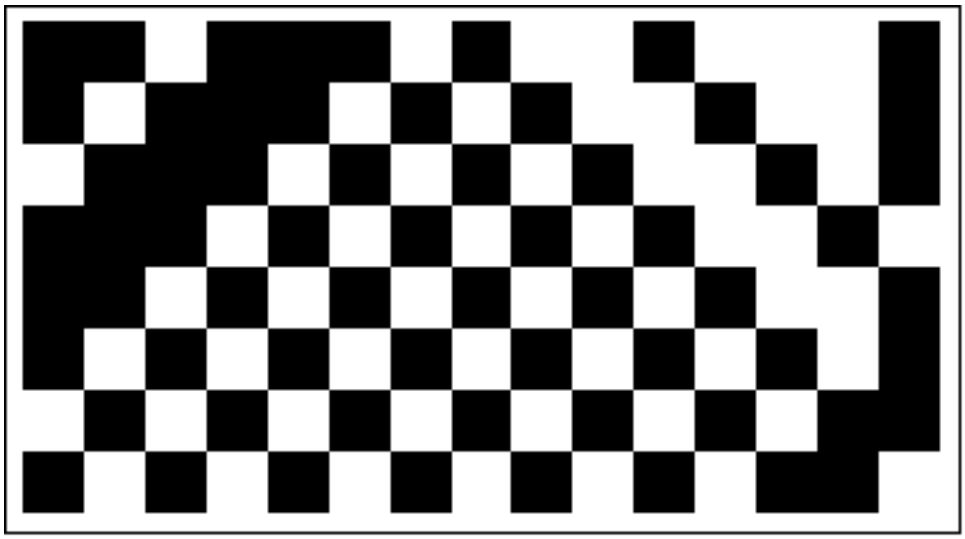

Figure 5.3: 7 iterations of Rule 184 highlighting an initial majority of black cells

The key behind Rule 184's success is that it is conservative: this means that the numbers of black cells and white cells do not change throughout the computation; they are merely reordered. In fact, it makes intuitive sense that non-conservative two-state automata are incapable of performing majority classification with a constant

neighbourhood size (not a function of the lattice size). The accepting condition chosen by Capcarrere is perfectly valid; it can be easily checked by both humans and machines. This reinforces the point we made earlier: computation is in the eye of the beholder. Whether this simple sorting can be seen as a computation or not is only a matter of perspective. By using human intuition to change an accepting condition, a problem that was previously unsolvable becomes very easily solvable. This also shows that universality and "interestingness" are two separate ideas: Rule 184 belongs to Wolfram's Class 1 or 2 (there is no clear taxonomy of classes), yet it still perfectly solves a desired problem.

Fuks later extended Capcarrere's idea to show that even under the all black or all white accepting condition, a perfect classification is possible using a two-state automaton [15]. The trick is to allow a rule change from Rule 184 to Rule 232 after $\frac{n}{2}$ cycles; the automaton then converges to the desired accepting condition after no more than n cycles. Rule 232 is a simple totalistic local majority automaton; it only becomes effective for global majority after Rule 184 ensures that no consecutive black or white cells exist unless they are the global majority.

We believe that a slight modification of Capcarrere's idea can make it even more powerful: we propose replacing the periodic boundary conditions with static boundary conditions; a black cell to the right and a white cell to the left. Under these conditions, Rule 184 stacks all black cells to the right of the automaton, and leaves all white cells on the left. It can be seen as if there were a force (such as gravity) attracting all black cells to the right. We expand on this concept in the next section to provide the first automaton capable of performing majority classification perfectly in two dimensions. For now however, we notice that by changing the boundary conditions only one cell

(or two if the lattice size is even) needs to be checked to provide an answer to the majority question - under Capcarrere's approach all cells need to be checked. This simplifies spotting the accepting condition for both humans and machines. It also allows checking for other percentages of cell densities: it makes answering questions such as "do black cells constitute more than 30% of the total number of cells?" possible, while this cannot be done under periodic boundary conditions.

5.1.8 Two-dimensional perfect majority classification

After presenting his solution to the one-dimensional majority classification problem, Capcarrere presents an open problem [7]: how can this solution be extended to two dimensions, and can the two-dimensional majority classification problem be completed in $O(\sqrt{n})$ cycles (assuming a square shape where n is the number of cells in the automaton)?

We devised a solution based on Rule 184 that performs in expected $O(\sqrt{n})$ time given a random initial configuration. This solution, which we call the "gravity automaton" has static vertical boundary conditions and periodic horizontal boundary conditions: the top row is all white, the bottom row is all black, and the left and right columns are connected to each other. The gravity automaton can therefore be seen as an upright cylinder. All cells have a uniform neighbourhood shown in Figure 5.4: it is a Moore neighbourhood of radius 1 with one additional cell immediately to the right of the middle row (this can also be seen as a von Neumann neighbourhood of radius 2 where three cells are not used).

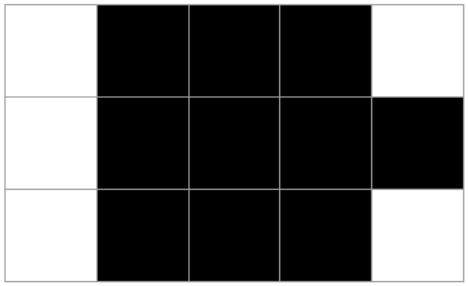

Figure 5.4: Representation of the neighbourhood in the gravity automaton

Transition rules

Since this a conservative cellular automaton, we explain its behaviour in terms of particle motion where black cells are considered particles and white cells are considered free space. We believe such a description to be much more intuitive than one describing every transition rule in detail. The reason why we call this system the "gravity automaton" becomes more apparent with this description: The top white row boundary condition is just additional free space above all particles, while the bottom black row is the ground, beyond which particles cannot fall. The particles then behave as follows:

1. If a particle has nothing under it (the cell below it is white), it simply falls by one cell

2. If the particle has other particles immediately below it and nothing above it (it is the top particle in a column of particles), it moves one cell to the left (the gravity vector is not fully vertical; it leans slightly to the left)

3. If the particle has other particles immediately below it but also has particles immediately above it (it is part of a column), it moves one cell to the right; here

the particle on top of the column (the one which satisfies rule number 2) can be seen as having with its left movement pushed all particles below it to the right.

All rules 1 to 3 are in priority sequence. This means that under rule 2, the particle also checks if there is a particle diagonally to its top left. If so it does not move in order to avoid a collision with that particle since it would be falling into the same spot. The same check is done to the top right under rule 3 to avoid a collision with rule 1. A particle satisfying the conditions of rule 3 also avoids a collision with another cell under rule 2 by checking the cell two positions to its right; if that cell contains a particle then it is given priority and the checking particle does not move to the right. Figure 5.5 presents a small example illustrating these rules.

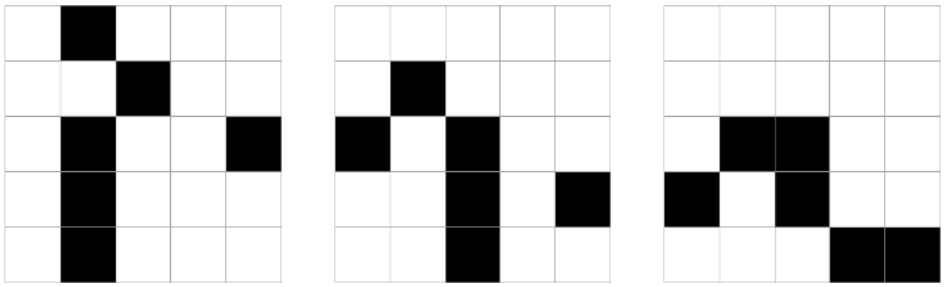

Figure 5.5: Three consecutive iterations of a small example gravity automaton. Notice how the two rightmost particles and the top particle simply fall (following the first rule). The top particle in the column moves to the left (rule 2), and the lower two particles move to the right (rule 3) - there are no conflicts in the first transition. In the second transition, the two uppermost particles in the column face a conflict (the first one under rule 2 and the second one under rule 3) and therefore do not move.

Terminating condition

The accepting condition is reached when as many rows as needed are completely full. Figure 5.6 illustrates this condition for the example automaton presented in Figure

5.5. The behaviour of the topmost row is similar to Rule 184. Thus to solve the majority problem, if the number of rows is even, one only needs to check if there is any particle on the upper middle row after a certain number of iterations; if so, then black has the majority. Otherwise, it is white or the initial numbers of black and white cells are equal; if the latter case is a concern one can check the lower middle row for any gaps that, if present, would confirm that white has a strict majority (if no such gaps exist then there would be no majority). If the number of rows is odd, one only needs to check if there are any consecutive black or white cells in the middle row. If so, whichever colour is present in two or more consecutive cells is the colour with the majority; there cannot be both consecutive black and consecutive white cells in the middle row at the same time after convergence. This accepting condition can also be easily adapted to check for thresholds other than 50% up to a $O(\sqrt{n})$ resolution; changing boundary conditions or other tricks would be needed for an $O(n)$ resolution.

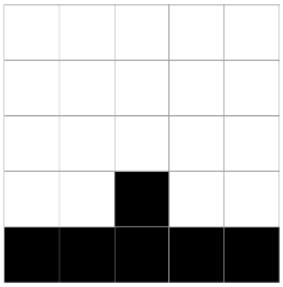

Figure 5.6: Terminating condition of the example automaton reached two transitions later

As with the one-dimensional case, we can reach an all-black or all-white terminating condition if desired by marking the middle row (private static information under our framework) and changing the rules to propagate its majority colour across the rest of the automaton.

Analysis

By giving priority to the first rule, we ensure that particles keep falling while they can. If they cannot, they move left and right until they find an empty spot where they can fall. This simple strategy ensures that after a certain number of iterations, as many bottom rows as possible are guaranteed to be filled with particles. Then when the top row is partially full, the particles in it cannot follow the first and third rules and as such they always follow the second rule which mimicks Rule 184 exactly (but in the reverse direction). This means that the rules are correct and the accepting condition is guaranteed to be reached after some number of cycles.

The question that remains to be answered is: how many cycles are needed? A large number of practical trials with randomly-generated initial conditions and varying concentrations of black and white cells (obtained by flipping a biased coin to decide on the colour of each cell) has shown that the number of cycles before the accepting condition is reached falls between $\sqrt{n}$ and $2\sqrt{n}$, and is never practically above $3\sqrt{n}$. We can explain this running time by dissecting the vertical (1) and the horizontal (2 and 3) rules individually.

All particles in a column need no more than $\sqrt{n}$ time before they cannot fall anymore. This is illustrated by two cases: the individual particle and the column of particles. An individual particle in the worst case (topmost row) falls to the ground in $\sqrt{n}$ time. On the other hand, in a column of particles, only the lowest particle can fall at one time. The particles above it have to wait (we are disregarding horizontal motion for now). k successive particles take $k-1$ cycles to be separated from each other, with their number shrinking by one particle at a time (the lowest particle separates itself from the column at each time unit). At this point, they can be seen

as individual particles since they can all fall at the same time. The time for their fall to be completed is the time required for the topmost particle to reach the top of the column when the column is completely affixed to the ground. For a column of size k, the top of the column is situated at k cells above the ground. Therefore, assuming the column is initially located at the topmost position, the particle needs to fall $\sqrt{n}-k+1$ cells. The total running time for the fall becomes $k-1+\sqrt{n}-k+1=\sqrt{n}$.

In the horizontal case once the column has reached the ground, the particles are separated from it at the rate of one to two particles per cycle (one to the left under rule 2 and one to the right under rule 3 if it finds a gap to fall into). There could be some conflicts slowing this movement down but generally it remains within $O(\sqrt{n})$. Combining these two movements yields an $O(\sqrt{n})$ total running time; this is especially the case when the initial conditions are generated at random, which means that there are no large discrepancies in the column sizes. Even if there were such discrepancies, this set of rules helps resolve them both while the particles are in the air and when they touch the ground.

Special case

Although every random practical test we did on the gravity automaton terminated in $O(\sqrt{n})$, we were able to artificially generate a special case where $O(n)$ time is required for it to reach its accepting condition. This is the case of a pyramid-like structure - depicted in Figure 5.6 - of size (number of cells) $\omega(\sqrt{n})$, where $\omega(\sqrt{n})$ is $\Omega(\sqrt{n})-\theta(\sqrt{n})$. The pyramid itself does not need to be in the initial condition; any structure that can lead to a pyramid structure over time is sufficient. Some examples include an upside-down pyramid or an hourglass. A superset of these structures is

the set of structures where, in a rectangle, the middle column contains the largest number of black cells, followed by one column to its left and one column to its right which contain one less black cell, etc; this continues until there are no black cells left so that if all black cells in a column are stacked on top of each other in the botton of the rectangle, a shape similar to the one depicted in Figure 5.7 is obtained.

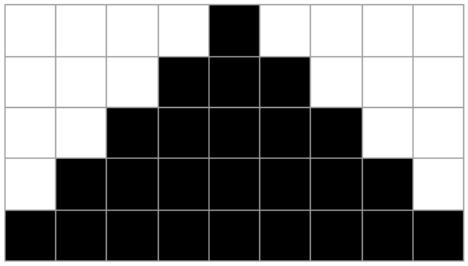

Figure 5.7: Worst case structure for the gravity automaton

We believe this to be the only special case; however we leave finding other cases as an exercise to the reader. In the pyramid case, particles fall one by one to the left starting with the topmost particle, and as such a time proportional to the size of the pyramid is required for termination. Practically, the probability of generating such a structure with size $O(n)$ from random initial conditions is extremely small. However, we avoid calculating that probability because there is no clear-cut boundary between what terminates in $O(\sqrt{n})$ time and what terminates in $O(n)$ time. The running time required to flatten any structure is relative, and depends on how much that structure is similar to a pyramid. Exact pyramids require $O(n)$ time, while typical randomly-generated structures require $O(\sqrt{n})$ time. All other structures fall somewhere between these two running times, depending on their degree of similarity to exact pyramids.

5.2 Convex hull

The problem of finding a two-dimensional convex hull can be described as follows: given a set of planar points, what is the smallest convex polygon containing all of them? This problem is illustrated by Figure 5.8. While there is a large number of algorithms answering this question using traditional computers, no solution exists yet using cellular automata. The two partial solutions we found in the literature take a non-convex polygon as an input, but not a set of discrete points. This means that the points have to be connected together before being fed into the cellular automaton whose job is to form a convex polygon encompassing them and their connections.

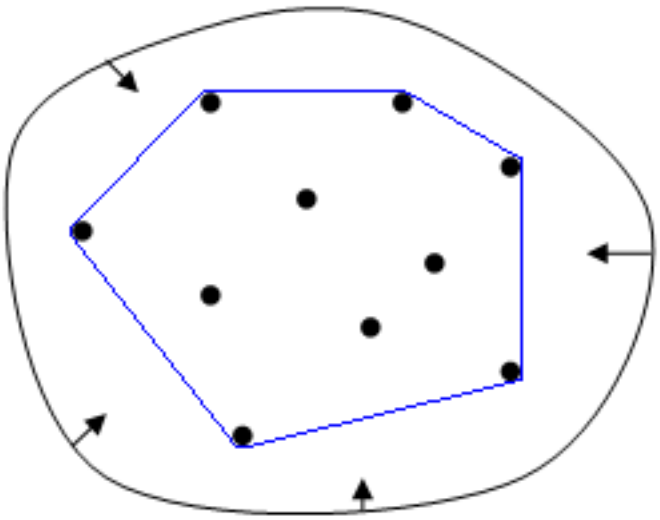

Figure 5.8: Convex hull of a set of points with the elastic band analogy

One of these solutions (Figure 5.9) is given by Rosin who uses a cellular automaton to find an approximate convex polygon containing the input concave polygon [32]. Rosin later enhances his results (while still being approximate) by adding memory capability to his cellular automaton.

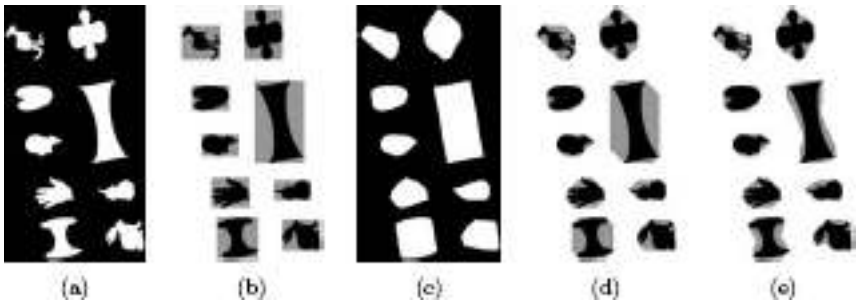

Figure 5.9: Rosin's solution to the convex hull problem [32]. (a) initial conditions, (b) cellular automaton creating a box surrounding the initial conditions, (c) exact solution (not reached using a cellular automaton), (d) Rosin's one-cycle automaton approximately solving the problem, (e) Rosin's two-cycle (memory) automaton approximately solving the problem. These solutions suffer from being approximations and from their requirement for connected initial conditions.

The other solution is given by Adamatzky in [3] and depicted in Figure 5.10. It is a simple totalistic cellular automaton capable of finding an exact solution to the 45-convex hull problem given a concave polygon. The 45-convex hull is a convex hull that is restricted to vertical, horizontal, and perfect diagonals; this means that the angles are always a multiple of 45 degrees, and there are four possible line directions. We will use this definition of the convex hull since it is the least ambiguous in a grid situation such as our cellular automaton.

Intuitively, the proposed cellular automaton can find an exact convex hull for a set of discrete, disconnected input points. It behaves like an elastic band expanded around a set of pins (the input points) and then released to form the tightest polygon around them, as shown in Figure 5.8. However, the band is overly elastic so before forming the tightest convex polygon, it goes in even tighter to form a concave polygon before it expands into the desired convex polygon.

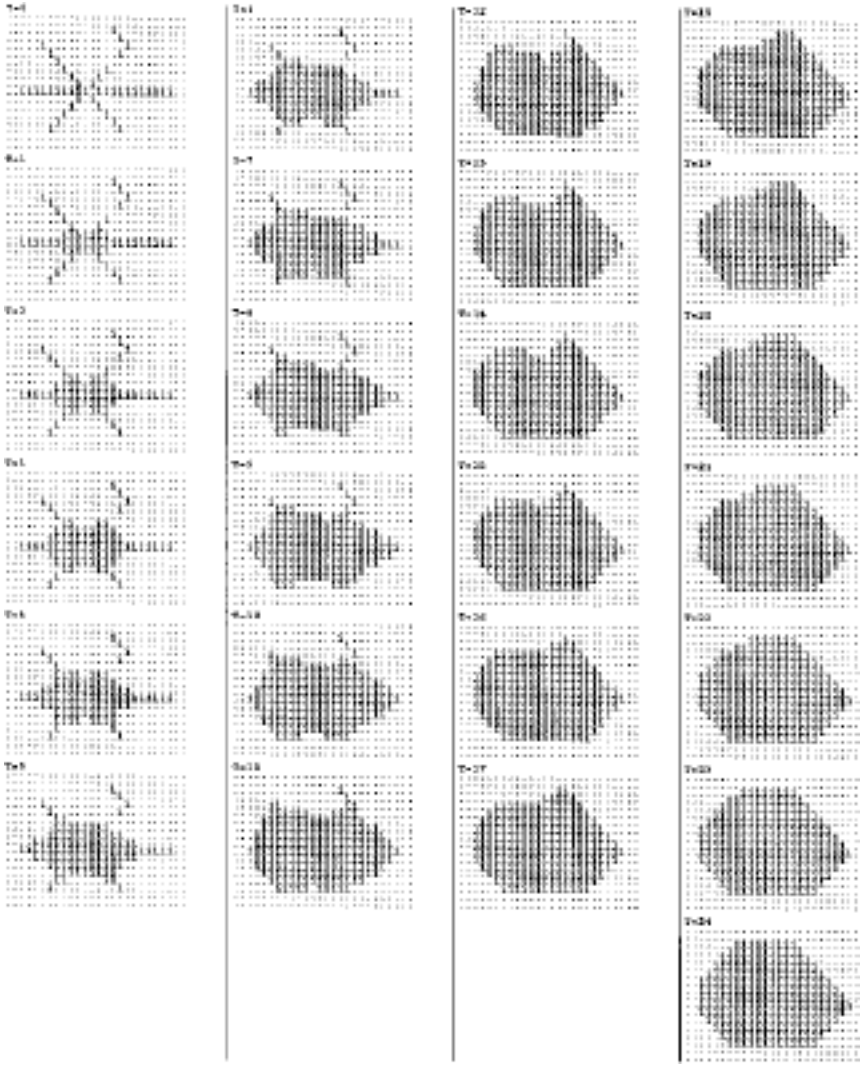

Figure 5.10: Figure taken from [3] showing 25 iterations of Adamatzky's cellular automaton. Notice the connectedness in the initial conditions.

5.2.1 Rule description

The automaton therefore uses two stages: one for tightening the elastic band and the other for re-expanding it. It has three states: black (2), white (0) and gray (1). Black is reserved for points that are certainly part of the convex hull. White is the colour of cells which are not part of the convex hull, although a cell coloured as white can be re-coloured as black in the second stage when the convex hull expands into it. Gray describes the points which may or may not belong to the convex hull; a decision regarding them has not been reached yet. By the time the automaton reaches its accepting condition, there are no gray points left.

All cells in the automaton have a Moore neighbourhood of radius 1. The boundary conditions are static and consist of white cells.

First stage

The automaton starts with the input points in black and all other points in gray (Figure 5.11). Then the gray area begins to tighten itself around the black points, increasing the number of white cells surrounding it in the process (Figure 5.12).

When the rubber band reaches a black point, it is progressively straightened by that point and prevented from shrinking (Figure 5.13). It is also prevented from shrinking under another condition: when it risks cutting itself off, i.e. when it becomes just a thin strip and removing any points from it would cut the connection between the black points. Figure 5.14 shows the end of the first stage, when the rubber band can no longer shrink.

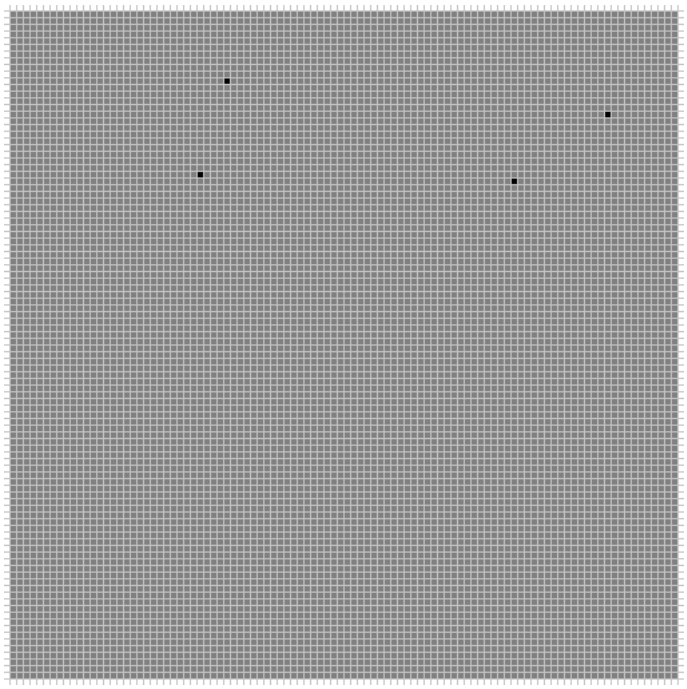

Figure 5.11: Initial state of a 100×100 convex hull cellular automaton shown with four input points and the boundary conditions

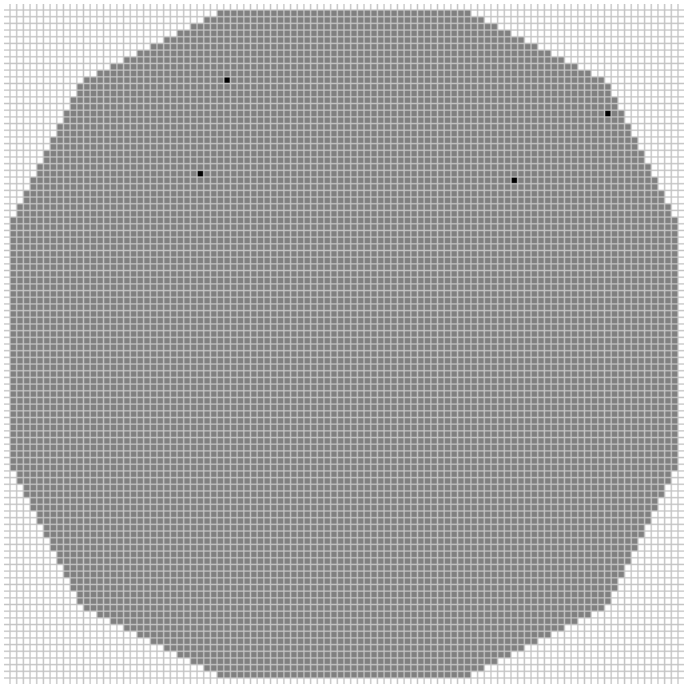

Figure 5.12: State of the same automaton after 31 iterations

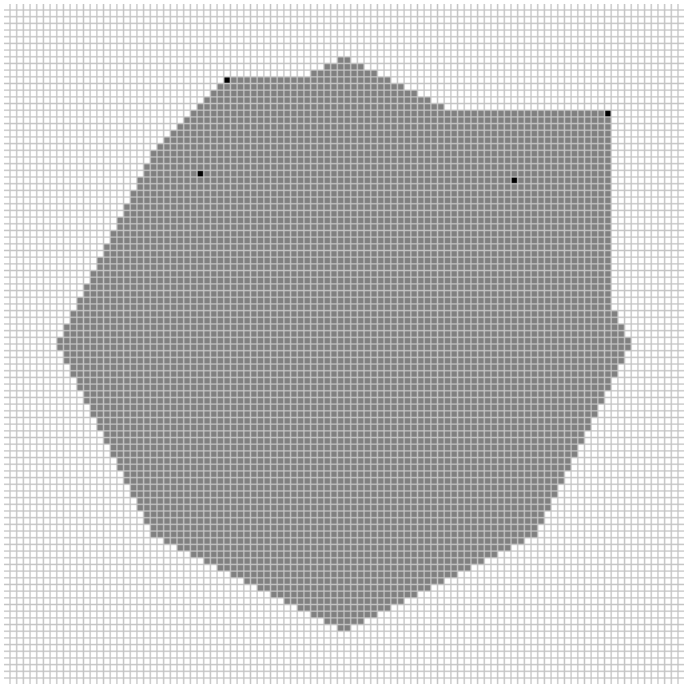

Figure 5.13: Notice the effect of the two topmost points on the shape of the polygon

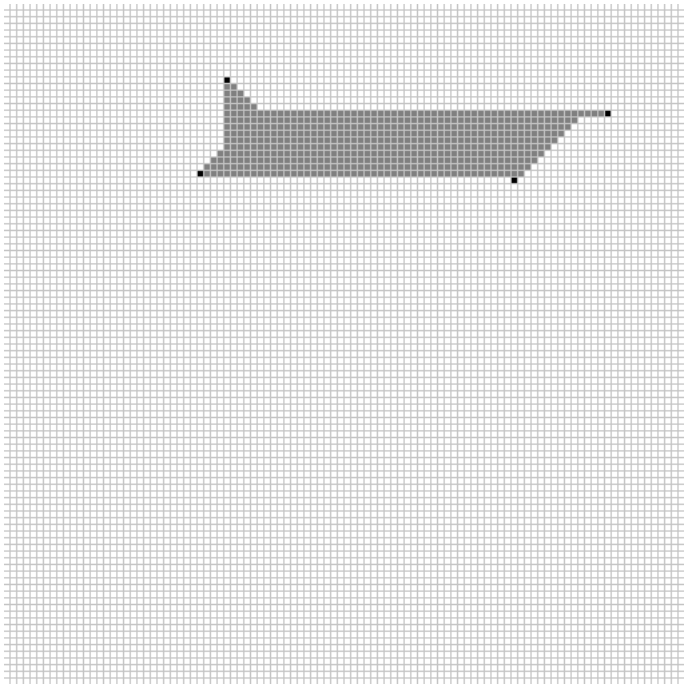

Figure 5.14: Same automaton at the end of the first stage. By now, all gray points are guaranteed to be part of the convex hull. Notice how the rightmost black point is being kept connected to the rest of the polygon by a thin gray strip.

We will try to explain the main pseudo-totalistic rules as simply as possible. First, we begin by defining the neighbourhood areas under consideration (Figure 5.15).

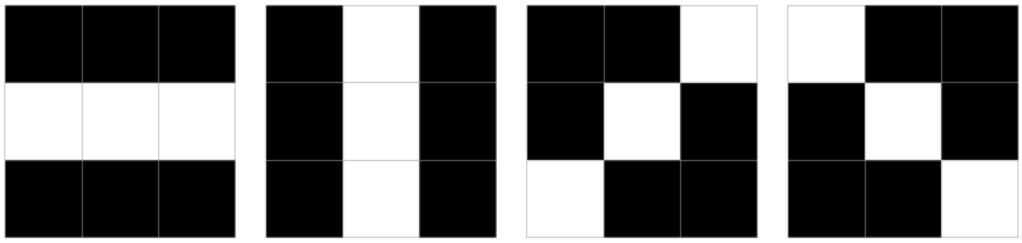

Figure 5.15: The first picture represents the top row, the middle row and the bottom row. Under the rules below a row is considered as the sum of the states of the cells it contains (a white cell is a 0, a gray cell is a 1 and a black cell is a 2). The second picture shows the left column, the middle column and the right column. The third picture represents the top left corner, the slash and the bottom right corner. Similarly the fourth picture shows the top right corner, the backslash and the bottom left corner. The total is the sum of the states of all cells in the neighbourhood, while the number of non-quiescent cells is the number of gray or black cells in the neighbourhood.

In this phase, only gray cells can change their colours. Black cells remain black and white cells remain white throughout the computation. Gray cells become white if:

- The total is smaller than or equal to 4 and in all of the four pictures shown in Figure 5.15 the sum of the white cells is smaller than or equal to the sum of the highlighted cells (e.g. for the first picture the middle row is smaller or equal to the sum of the top and bottom rows, etc.). This rule removes cells with few neighbours (i.e. cells on the edge) while making sure that removing them does not cause any cut-off.

- *Or* in any of the four pictures in Figure 5.15 one of the black clusters of cells is greater than or equal to 4 and its opposed cluster is equal to 0 (e.g. for the first picture the bottom row is all white while the top row has at least one black

cell and two gray cells, or at least two black cells, and vice-versa). This rule removes the cells that are just one point outside of the convex hull which would not be removed under the rule above because they have a powerful one-sided neighbourhood.

- *Or* if the total is smaller than or equal to 2 (there is only one other non-quiescent cell in the neighbourhood and it is gray).

- *Or* if one of the black clusters is empty and the sum of both black clusters corresponding to its 90-degree rotation is smaller than or equal to 1 plus the number of black cells in the neighbourhood (e.g. for the first figure the top row is empty and the sum of the left and right columns is smaller than or equal to 1 plus the total number of black cells in the neighbourhood). The last two rules are both meant for the cases where the number of input points is very small (one to three points in general); they overcome the first two rules to make sure that the gray area is shrunk as tightly as possible around these points.

Second stage

Once the elastic band is done from tightening itself around the points according to the rules above, the system comes to a stop until the rules are changed to allow it to expand back into a convex polygon. The second set of rules is very simple and is based on [3]:

- A gray cell always becomes black. Therefore, after the first cycle of the second stage there are no more gray cells.

- A white cell becomes black only if:

– The sum of its neighbourhood is larger than 6, which means that it has four or more black neighbours; this rule is the one described by Adamatzky in [3]

– The sum of its neighbourhood is equal to 6 but it is made of a cornerless "L" shape on any of the four corners (Figure 5.16); considering 90-degree rotations and mirroring, there are eight possible such shapes. This rule was added to overcome the special case where the first stage leaves some 30-degree lines in the system, which the first rule is incapable of handling.

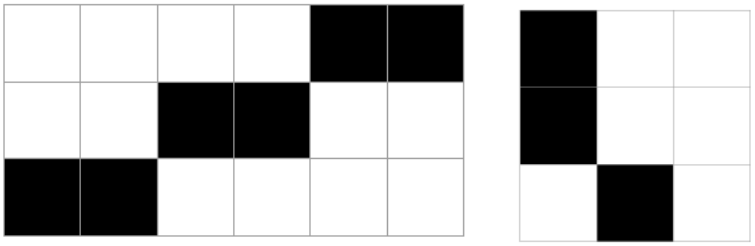

Figure 5.16: The special case and the cornerless L shape neighbourhood

These rules are enough to perfectly expand the concave polygon into a 45-convex hull, as shown in Figure 5.17. Figure 5.18 shows another example with more points at the end of the first stage and at the accepting condition.

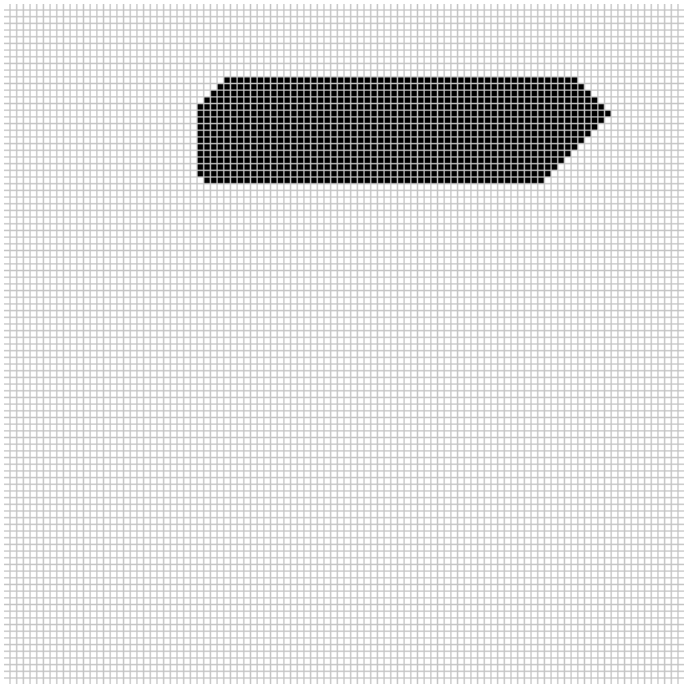

Figure 5.17: The 45-convex hull reached from the initial conditions in Figure 5.11

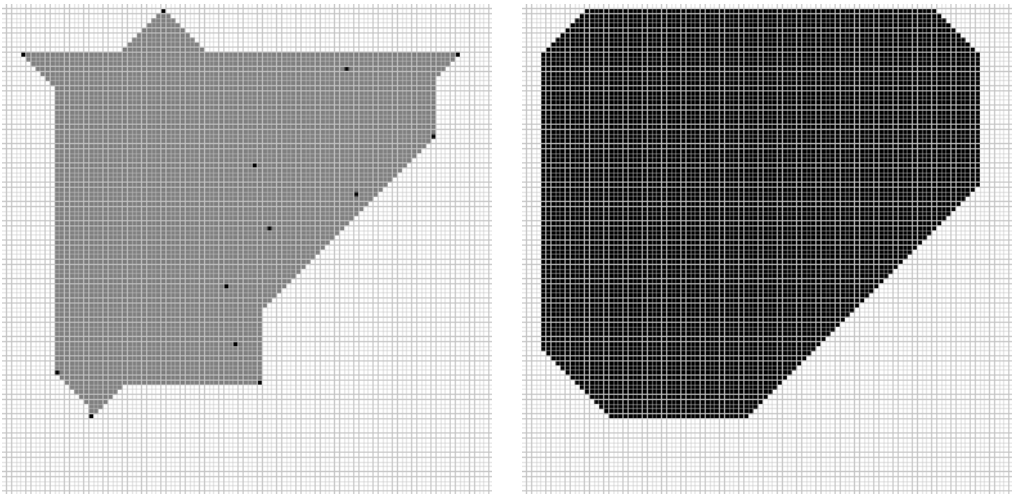

Figure 5.18: End of the first and second stages for an example where not all points belong on the convex hull

5.2.2 Analysis

One question that remains to be answered is: when should the rules change from those of the first stage to those of the second stage? Our practical trials have shown that the first stage is always completed before $2.5\sqrt{n}$ cycles. This reflects our theoretical results too: the first stage can be seen as consisting of two parts:

1. The elastic band shrinking uniformly throughout (with an octagon shape)

2. The elastic band straightening after having touched one of the points

The first part takes $1.5\sqrt{n}$ cycles; this can be seen by having empty initial conditions (no points on the convex hull) and checking how long it takes for all the gray area to disappear, i.e. for the octagon to shrink to the middle of the automaton from all eight sides. It takes $\frac{\sqrt{n}}{2}$ cycles for the octagon to be created (for the message to reach the middle of the automaton from opposing corners) then $\sqrt{n}$ cycles for the octagon to shrink to the middle of the automaton since it moves by one cell every two cycles. In the event where there are many points and they are all located on one side of the automaton (in one quadrant, or two consecutive quadrants), the first part can take longer since the octagon needs to shrink past the centre of the automaton; however, these extra cycles are subtracted from the second part since this means that the shape is small enough that straightening takes much less time than the maximum.

The second part clearly takes at most $\sqrt{n}$ time since the straightening message is sent at the pace of one cell per cycle, and vertically, horizontally and diagonally there are at most $\sqrt{n}$ cells between the farthest points.

This means that the rules can be changed after $2.5\sqrt{n}$ cycles according to hierarchical cellular automata as proposed in our framework. The second stage has a

$\sqrt{n}$ running time for the same reason as the second part of the first cycle. Therefore, the overall running time of the system is in $O(\sqrt{n})$. We believe this to be optimal for the problem as described (each cell represents a point in the euclidean plane) since one could always have points on opposite corners of the automaton and some communication between them would be needed; although the rules do not explicitly show communication between the points (since they are not in each other's neighbourhoods), such communication is essential and is at the centre of emergent behaviour.

For the sake of comparison, the lower bound for the running time of a sequential computer solving the convex hull for m planar points is known to be $\Omega(m \log m)$. Several optimal algorithms achieving this lower bound are known; for example, Graham's scan computes the convex hull of m planar points in $O(m \log m)$. In parallel, a constant $O(1)$ running time can be achieved on a concurrent-read concurrent-write PRAM with $O(m^2)$ processors. [4]

This solution is faster than its sequential counterpart when the number of input points is $\Omega(\sqrt{n})$. Although it cannot achieve a constant running time (due to the locality of rules), it can perform better than the best-known parallel solution (with performance defined as the product of the running time with the number of processors) when the number of input points is $\Omega\left(n^{\frac{3}{4}}\right)$. Despite these encouraging performance characteristics, we do not believe that the convex hull problem is the perfect example of a problem to be solved using cellular automata; it can be easily solved sequentially and using other parallel methods, and is certainly not optimized for local behaviour. However, we provided it as an example to show that our framework greatly simplifies the design of difficult cellular automata and that cellular automata are capable of

solving problems requiring global behaviour, sometimes even competitively with other methods.

5.3 Sensor positioning

Of the three solutions studied in this chapter, sensor positioning is the one with the most practical utility. Wireless sensor networks are systems consisting of a large number of miniaturized sensor nodes deployed to operate autonomously in unattended environments. They probe their surroundings and send the collected data to an access point either directly or through a multi-hop path. Wireless sensor networks have many applications including forest monitoring, disaster management, space exploration, factory automation, border protection and battlefield surveillance. [41]

Geometrical positioning of sensors (nodes) in sensor networks is an important research area whose aim is to optimize one or more design variables such as coverage, connectivity and energy consumption (see [8] and [41] for surveys of node placement strategies and algorithms). While the communication methods and protocols of the sensors can have an important impact on these variables, we will only deal here with sensor positions for simplicity reasons, keeping in mind that any other constraints can be added later to the resulting system. Coverage is a Quality of Service (QoS) problem whose goal is to minimize the part of the desired monitored area that is not covered by any sensor node. In other words, the coverage problem is optimally solved when every part of the area that we need to monitor is covered by at least one sensor node. Connectivity is another QoS problem aiming to make sure that every sensor node is connected either directly or indirectly (through other sensor nodes) to an access point; the information collected by the sensors is useless if it cannot be transmitted

back. Sensors are generally battery-operated; energy consumption is therefore a key performance metric because it determines the lifetime (and replacement cycle) of the sensors. Energy consumption should be minimized in order to minimize the frequency of sensor replacements.

Like some of the algorithms described in [8] and [41], we aim to optimize all three key performance metrics: coverage, connectivity and energy consumption. Energy consumption can be greatly reduced by having simple components with exclusively local decisions. It can also be seen to be inversely proportional to the node sparsity [8]; this means that we should aim to achieve the best balance between maximum coverage and connectivity and a minimal number of nodes.

The simplicity and locality of nodes in sensor networks bears a striking similarity to that of cells in cellular automata. We have therefore elected to simulate simple sensor node positioning rules on a cellular automaton.

5.3.1 System description

R_S and R_C are two widely-used characteristics of nodes in wireless sensor networks. R_S is the sensing radius; it defines the maximum distance that a point can be from a sensor while still being covered by that sensor. R_C is the communication radius, which is the maximum distance two sensors can be from each other while still being able to communicate. We will only consider $R_S \leq R_C \leq 2R_S$. This makes sense according to our objectives: if $R_C \leq R_S$ then R_S needs not be as large as it is since the necessity of connectivity guarantees that there are several sensors covering the same area (redundancy may be useful in some cases but it contradicts our sparsity requirement). On the other hand, if $R_C \geq 2R_S$ then R_C should be reduced because

the need for coverage ensures that sensors are within $2R_S$ of each other.

We simulate our wireless sensor network on a two-dimensional cellular automaton: space is therefore discretized. However, this is not perceived as a problem or limitation since many existing mathemcatical models of sensor networks also deal with discretized space. In our model, a cell in state 0 does not contain a sensor (but still needs to be monitored by a sensor). A cell in state 1 is an access point and a cell in state 2 is a sensor node. For our purposes, both access points and sensor nodes can monitor their environment and they have the same sensing and communication radii. The difference between them is that access points are capable of communicating directly with the external observer (through wired or additional powerful wireless connections); this means that access points need to be wired somehow even in the case of additional wireless connections (in this case they would be plugged into some power source because of the large power consumption). Therefore, the other difference between access points and sensor nodes is that access point positions are fixed while sensor nodes are mobile.

The mobility of the sensor nodes classifies our system as a dynamic positioning system, as opposed to a static positioning system where the sensors are assigned to fixed positions upon deployment. Sensing and communication radii are assimilated to the Moore neighbourhood radius of the cellular automaton; a direct implication of this fact is that sensing and communication radii are of constant size relatively to the size of the system. Since $R_C \leq 2R_S$, our transition rules need only focus on the communication radius; this is the case because under this restriction, the fact that two nodes can communicate means that they are collectively fully monitoring the area between them.

5.3.2 Transition rules

The nodes behave according to very simple even-odd (described in Section 4.2.2) probabilistic rules (Section 4.2.3). Therefore, the automaton can be seen as being in a cycle of two steps: nodes decide where they want to move in the first step, while they actually make that movement in the second. This division simplifies the design because the choice of where to move is partially probabilistic. As mentioned earlier, the goal of that movement is to maximize coverage, communication and sparsity.

Even cycles

The first step is when the nodes announce their intention to move. The decision to move is made very simply; it is a random decision based on the number of other nodes in a sensor's neighbourhood weighed by the distance of these nodes from the sensor. For example, for a neighbourhood of size 4 every sensor node calculates a number k as follows:

$$k = 4N_1 + 3N_2 + 2N_3 + 1N_4$$

In this formula N_1 is the number of nodes within a distance of 1 cell from the sensor in question, N_2 the number of nodes within a distance of 2 cells, etc.

k is then used to determine the probability of moving:

- For $k = 0$ or $k \geq 9$ the node has a 50% chance of moving
- For $k = 1$ or $7 \leq k \leq 8$ the node has a 37.5% chance of moving
- For $k = 2$ or $5 \leq k \leq 6$ the node has a 25% chance of moving

- For $3 \leq k \leq 4$ the node has a 12.5% chance of moving

These numbers are not cast in stone; however, they have worked well in our practical experiments. They are meant to give a greater incentive for a node to move when it has too few or too many neighbours. The assumption is that a node that has too few neighbours is potentially isolated (incapable of reaching an access point either directly or indirectly) is therefore encouraged to move for the sake of connectedness. On the other hand, a node that has too many neighbours is not needed at its current location (while being probably needed somewhere else) and is hence encouraged to move for the sake of sparsity. Note that chances of moving are kept at or below 50% to provide some stability to the system.

The question that remains to be answered is: "where does a node move?" Once it has taken the decision to move, a sensor chooses at random one of its eight immediate neighbouring cells while following two conditions:

- The chosen neighbouring cell must be empty
- The chosen neighbouring cell must also be outside the reach of all other nodes (conflicts are resolved by simply preventing them from occuring in the first place)

The sensor then points to the cell it has randomly chosen by changing its state to a number from 3 to 10 reflecting the chosen direction. If on the other hand it decides not to move, it remains in state 2.

Odd cycles

The rules for odd cycles are very simple:

- A cell in state 1 or 2 does not change its state
- A cell in state 0 changes its state to 2 only if there is a cell in its immediate neighbourhood pointing in its direction (having the right state number greater or equal to 3)
- A cell in any of the states 3 to 10 changes its state to 0

Thus, the moves designated in the previous cycle would be complete. The cycle then repeats itself.

5.3.3 Testing

This algorithm assumes that the "right" number of sensors is deployed in the first place. Too few sensors cause coverage and connectivity issues, while too many sensors cause a waste of resources and energy (violating the sparsity requirement). But what is that number?

The good news is that ideal placement (with the minimal number of sensors) is possible given R_C and R_S. The problem with such placement is that it is static and extremely vulnerable to any minor position change or sensor failure. However, we can use the ideal placement as a benchmark against which to compare our placement algorithm. Taking $R_C = R_S = 3$, the ideal placement (shown in Figure 5.19) for a cellular automaton with periodic boundaries requires one sensor for every 18 cells.

Starting from $\frac{10{,}000}{18} \simeq 556$ we performed several tests on a cellular automaton with varying parameters. The average results of these tests are given in Table 5.2.

From these tests, we see that deploying 50% more sensors than the minimum yields excellent results with no disconnections and almost complete coverage (Figure

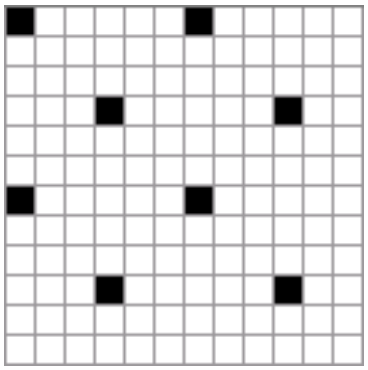

Figure 5.19: Ideal static placement of sensors for a small cellular automaton with periodic boundaries and $R_C = R_S = 3$

Number of sensors (as a multiple of the minimum)	R_S	R_C	uncovered area	disconnected sensors
1.1	3	3	2%	3.1%
1.1	3	4	2%	0.2%
1.3	3	4	1%	0%
1.5	3	4	0.3%	0%
2.0	3	4	0.05%	0%

Table 5.2: Performance of the proposed algorithm under various parameters

5.20). Note that in our system the area covered changes between cycles (the numbers displayed in Table 5.2 are average values in any given cycle). Thus, the small areas missed by sensors in one cycle are covered in subsequent cycles, unlike with static placement algorithms. We also notice that a communication radius slightly larger than the sensing radius dramatically reduces disconnection rates. However, a significantly larger communication radius is not necessary since the large number of deployed sensors (for coverage purposes) would prevent it from having any effect.

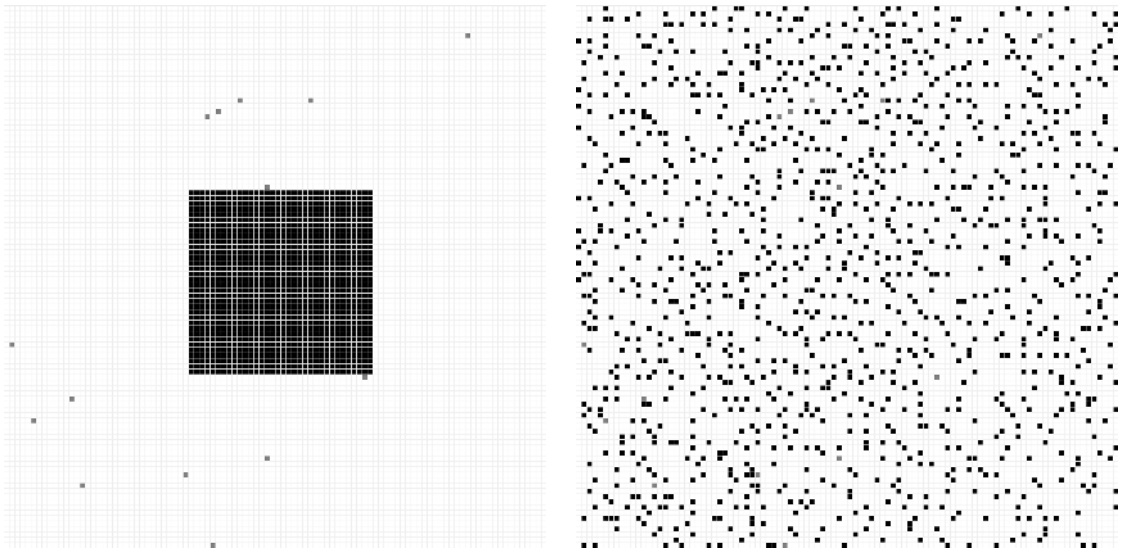

Figure 5.20: Initial and desired states of an automaton with 10,000 cells and 729 sensors. Note that despite the simplicity of the rules, the emergent behaviour is clear: it strives for sparsity while maintaining coverage and connectivity. Regardless of the initial state, the desired state is always reached and it looks roughly the same.

5.3.4 Analysis

We have shown how our system can achieve its objectives of maximizing coverage and connectivity while aiming for sparsity, provided the right number of sensors is initially deployed. We have also shown how we can quickly estimate that number. What remains is an analysis of the other benefits and side effects of this algorithm. We will base this analysis on the open problems described in [8], and show how we can solve many of them.

Sensors with irregular sensing or communication ranges

Since our transition rules are simply based on the number of other sensors every individual node can locally see within its communication range, this problem is inherently taken care of. Sensors with irregular sensing and communication ranges can

be simulated in cellular automata using non-uniform automata (described in Section 4.2.6 as part of our proposed framework).

Coverage solutions for mobile sensor networks

Mobility is at the core of the presented system. It enables desired initial positioning as well as fault tolerance when changes in the environment (or problems with individual sensors) cause reduced coverage or connectivity. In fact, fault tolerance is presented as a separate open problem in [8]; in our case, it is simply a consequence of mobility and the simplicity of transition rules.

Other energy conservation methods

Chen and Koutsoukos propose communication range reduction as an example measure aimed at energy conservation; while this measure is not part of our system, it can be easily accomodated by locally adjusting the communication range probabilistically depending on the number of other nodes in that range.

It is worth mentioning here that energy conservation seems to be the only potential major weakness of our system: constant movement significantly contributes to depleting node batteries. Our proposed solution to this problem resides in repeated cycles of simulated annealing: the probability of movement is decreased with time (for example 50% becomes 10% and 12.5% becomes 2.5%) in order to keep the sensors in place after they have found good positions. However, to maintain the fault tolerance benefits sensors should be given the chance to move again after potential changes in their environment have occured; this is accommodated by briefly raising the probability of movement periodically, and then decreasing it again.

This raises another issue: our algorithm needs an adequate number of nodes in order to function as desired; what happens when some nodes die (when their batteries are depleted for example)? We propose measuring the average node lifetime, and periodically deploying additional nodes. For example if 100 nodes are needed and the average node lifetime is of 10 years, 10 nodes should be deployed every year after the initial deployment. If the nodes are inexpensive and unintrusive, they can just die in-situ when their batteries are depleted; if this is not the case, being mobile the nodes can be instructed to move to a charging station when their battery levels reach critical values.

Chapter 6

Concluding thoughts

We have presented a framework conceived to simplify the process of designing complex cellular automata. It is not designed to replace the existing manual and automated methods we described in Chapter 3. Rather, it is meant to complete them and we have shown how allowing for human intuition can move problems such as the majority classification problem from unsolvable to easily and perfectly solvable. On the other hand, the greatest difficulty we had in designing the perfect majority classification solver was in making sure that the details of the rules match the required particle-like behaviour. We described the rules in terms of particles; however, actual low-level cellular automata rules are more complicated; for a particle to move from one spot to the other, two rules are needed: one to make it disappear from the first spot and the other to make it appear in the second spot. This is where using particle-based high-level manual design methods such as the one described in [38] would have helped us complete our task more efficiently. Genetic programming could have also been used to help in the design of rules that overcome some of the special cases reached in solving the convex hull problem. We did not use these approaches because although

described theoretically in the literature, they are not formulated in a readily-usable software framework.

In addition to providing the initial framework and solving two open problems using it, we tried to answer some more general questions regarding cellular automata. For example, we showed that computation or emergent behaviour is only a matter of perspective. We also showed that universality and computation are two separate concepts; in fact, simple non-universal automata can be "interesting" and can perform computations perfectly. Finally, we showed that human intuition still plays a central part in the design of cellular automata.

We believe that there are three directions for future work based on this thesis:

1. Further developing the proposed theoretical framework by adding language elements and expanding or detailing existing elements

2. Designing solutions for difficult problems in various fields using cellular automata in general and this framework in particular

3. Devising a suitable cellular automata hardware implementation that takes the framework into consideration and easily enables porting a wide variety of problems from it.

Regarding this last point, we propose the chaotic computer described by Munakata et al. [30] as a starting point. Its ability to instantly "change the rules" (by using a parameter that defines logic gate behaviour) closely matches our framework and allows the implementation of large systems using a significantly smaller number of elements.

Finally, no treatment of cellular automata is complete without some broader philosophical questions: is the universe a giant cellular automaton? We believe that anything can be seen as a cellular automaton provided a large enough neighbourhood and set of states (which may defeat the purpose); in fact, this stems directly from our belief in the laws of physics and cause and effect: any event occuring at time t is a direct consequence of one or more events or conditions at times prior to time t - a perfect description of transition rules in cellular automata. This also brings about the question of whether the universe operates in discrete or continuous time; we are in no position to answer this question but we can point out that some adaptations of cellular automata behaving in continuous time have been designed, and are able to cover that concern. We have shown that a large variety of behaviours (memory, probability, changing rules, etc.) can all be implemented using standard cellular automata given enough states and connections. Given this information, we believe that a cellular automaton with real-valued (infinite) states - as described by Rucker in [33], a large (inifinite?) number of cells and a large (infinite?) number of connections, as well as a fine-grained (continuous?) update time would have no problem in simulating the universe.

Bibliography

[1] Wikipedia: Garden of eden pattern. Available at `http://en.wikipedia.org/wiki/Garden_of_Eden_pattern`.

[2] Andrew Adamatzky. *Identification of Cellular Automata*. Taylor and Francis, London, Bristol, 1994.

[3] Andrew Adamatzky. Automatic programming of cellular automata: identification approach. *Kybernetes: The International Journal of Systems & Cybernetics*, 26(2):126–135, 1997.

[4] Selim G. Akl. *Parallel computation: models and methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[5] Selim G. Akl. Three counterexamples to dispel the myth of the universal computer. *Parallel Processing Letters*, 16(3):381–403, September 2006.

[6] David Andre, Forrest H. Bennett III, and John R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, Cambridge, MA, 1996. The MIT Press.

[7] Mathieu Capcarrere. *Cellular Automata and other Cellular Systems: Design & Evolution*. PhD dissertation, Swiss Federal Institute of Technology Lausanne, March 2002.

[8] Jie Chen and Xenofon Koutsoukos. Survey on coverage problems in wireless ad hoc sensor networks. In *IEEE SouthEastCon*, Richmond, VA, March 2007.

[9] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[10] James P. Crutchfield and James E. Hanson. Turbulent pattern bases for cellular automata. *Physica D*, 69(3/4):279, 1993.

[11] Sina Dengler. Multiple populations: A game of life variation. Queen's University Undergraduate Thesis, April 2007.

[12] David Elizondo. The linear separability problem: Some testing methods. *IEEE Transactions on Neural Networks*, 17(2):330–344, March 2006.

[13] Kent Fenwick. Exploring the world of simple systems : From the game of life to real life. Queen's University Undergraduate Thesis, April 2007.

[14] Edward Fredkin and Tommaso Toffoli. Conservative logic. In *Collision-based computing*, pages 47–81. Springer-Verlag, London, UK, 2002.

[15] Henryk Fuks. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55:2081R, 1997.

[16] Peter Gacs, Georgii L. Kurdyumov, and Leonid A. Levin. One-dimensional homogenuous media dissolving finite islands. *Problems of Information Transmission*, 14(3):92–96, 1978.

[17] Max Garzon. *Models of massive parallelism: analysis of cellular automata and neural networks*. Springer-Verlag, London, UK, 1995.

[18] Howard A. Gutowitz and Chris G. Langton. Methods for designing cellular automata with "interesting" behavior. Available at `www.santafe.edu/~hag/interesting/interesting.html`, 1988.

[19] Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific Publishing Company, 2001.

[20] Jarkko Kari. Theory of cellular automata: a survey. *Theoretical Computer Science*, 334(1-3):3–33, 2005.

[21] Ben Krose and Patrick van der Smagt. *An introduction to Neural Networks*. The University of Amsterdam, eighth edition, November 1996.

[22] Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, Jun 1995.

[23] Chris G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. In *CNLS '89: Proceedings of the ninth annual international conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks*

on Emergent computation, pages 12–37, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co.

[24] Craig. S. Lent and Paul D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541–557, 1997.

[25] Wentian Li. Phenomenology of nonlocal cellular automata. *Journal of Statistical Physics*, 68(5/6):829, 1992.

[26] Norman Margolus. Cam-8: a computer architecture based on cellular automata. In *Pattern Formation and Lattice-Gas Automata*. American Mathematical Society, 1994.

[27] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: expanded edition*. MIT Press, Cambridge, MA, USA, 1988.

[28] Melanie Mitchell. Computation in cellular automata: A selected review. In *Non-Standard Computation*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[29] Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Dynamics, computation, and the "edge of chaos": a re-examination. pages 497–513, 1999.

[30] Toshinori Munakata, Sudeshna Sinha, and William L. Ditto. Chaos computing: implementation of fundamental logical gates by chaotic elements. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(11):1629–1633, November 2002.

[31] Jean-Yves Perrier, Moshe Sipper, and Jacques Zahnd. Toward a viable, self-reproducing universal computer. *Physica D*, 97(4):335–352, 1996.

[32] Paul L. Rosin. Training cellular automata for image processing. *IEEE Transactions on Image Processing*, 15(7):2076–2087, 2006.

[33] Rudy Rucker. *The Lifebox, the Seashell, and the Soul: What Gnarly Computation Taught Me About Ultimate Reality, the Meaning of Life, and How to Be Happy.* Thunder's Mouth Press, 2006.

[34] Moshe Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Springer-Verlag, 1997.

[35] Klaus Sutner. Computing in cellular automata. Lecture given as part of the Computational Discrete Mathematics course at Carnegie Mellon University, 2007.

[36] Tommaso Toffoli. Occam, turing, von neumann, jaynes: How much can you get for how little? (a conceptual introduction to cellular automata). In *Proceedings of the International conference on Cellular Automata for Research and Industry*, Rende, Italy, September 1994.

[37] Sami Torbey and Selim G. Akl. Towards a framework for high-level manual programming of cellular automata. In *13th International Workshop on Cellular Automata*, Toronto, Canada, August 2007. The Fields Institute.

[38] Heather Turner, Susan Stepney, and Fiona Polack. Rule migration: Exploring a design framework for emergence. *International Journal of Unconventional Computing*, 3(1):49–66, 2007.

[39] Stephen Wolfram. *A New Kind of Science.* Wolfram Media, January 2002.

[40] Dietmar Wolz and Pedro P.B. de Oliveira. Very effective evolutionary techniques for searching cellular automata rule spaces. *To appear in the Journal of Cellular Automata*, 2008.

[41] Mohamed Younis and Kemal Akkaya. Strategies and techniques for node placement in wireless sensor networks: A survey. In *Elsevier Ad Hoc Network Journal (to appear)*.

[42] Konrad Zuse. *Rechnender Raum (Calculating Space)*. Friedrich Vieweg & Sohn, Braunschweig, Deutschland, 1969.