NTNU

Norwegian University of Science and Technology

Evolving Cellular Automata in-Materio

Sigve Sebastian Farstad

Master of Science in Computer Science
Submission date: December 2015
Supervisor: Stefano Nichele, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Summary

Evolution-in-Materio in the context of unconventional computing is the practice of using artificial evolution techniques to search for configurations of physical material samples that allow for them to be used as practical computational devices. The motivation for Evolution-in-Materio is two-fold: it is in part an exploration of new computational substrate materials as alternatives to silicon, and in part an exploration of metaheuristically-guided evolutionary search in the design space as a design process instead of traditional top-down engineered design.

One of the biggest problems in exploiting materials for computation is finding a good computational abstraction to carry computation on-top of the underlying physical processes. This thesis looks at the possibility of implementing cellular automata as an abstraction via Evolution-in-Materio to reason about the computational capabilities of single-walled carbon nanotube and polymer composite meshes.

In this thesis, computationally stable linear and non-linear logic gates and Elementary and Sub-Elementary Cellular Automata are successfully evolved in-materio. Not all Elementary Cellular Automata are shown to be successfully evolvable, which reveals computation complexity properties of the material-under-study.

Sammendrag

I-materio-evolusjon er i en ukonvensjonell-beregningskontekst praksisen å bruke kunstig-evolusjon-teknikker til å lete etter materialkonfigurasjoner av fysiske materialer som lar de benyttes som praktiske fysiske beregningsenheter. Motivasjonen for i-materio-evolusjon er todelt: det er dels en utforskning av nye beregningssubstrater som alternativer til silisium, og dels en utforskning av bruk av metaheuristiske evolusjonsbaserte søk i designrom som en designprosess som en motpol til tradisjonell komponentisert ingeniørdrevet design.

En av de største utfordringene i å utnytte beregningskraften i fysiske materialer er å finne en god beregningsabstraksjon som kan bære beregninger over de underliggende fysiske prosessene i substratet. Denne oppgaven tar for seg muligheten å implementere cellulære tilstandsmaskiner via i-materio-evolusjon som en abstraksjon for å kunne si noe om det absolutte beregningspotensialet til et enkelvegget karbonnanorørkomposittmateriale.

Et utvalg av stabile lineære og ikke-lineære logiske porter og Elementære og Subelementære Cellulære Tilstandsmaskiner er utviklet i denne oppgaven. Ikke alle Elementære Cellulære Tilstandsmaskiner lot seg vise å være utviklbare i karbonnanorørkomposittmaterialet med den brukte i-materio-tilnærmingen, som avslører beregningskompleksistetsegenskaper ved materialet.

Acknowledgements

I express gratitude towards my supervisor Dr. Stefano Nichele for insights, guidance and support during the research and writing of this thesis. Further, I would also like to thank Odd Rune Lykkebø for technical support with the Mecobo platform.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The natural evolutionary process, in stark contrast to the traditional human top-down building-block-oriented engineering design approach, is not one of abstraction, componentization and careful manipulation based on an understanding of underlying mechanics. Rather, it is a "blind-yet-guided" metaheuristic search approach able to exploit properties to create "designs" without needing to understand them or the underlying mechanics that power them. Evolution-in-Materio is an attempt to mimic this natural evolution process in order to exploit new, interesting and not-fully-understood materials for different use cases, most notably for computational purposes.

This thesis explores the idea of using Evolution-in-Materio to exploit a single-walled carbon nanotube and polymer composite random mesh material for abstracted computation using cellular automata.

1.1 Assignment Text

The assignment text for this Master's thesis is reproduced here:

> "Computation-in-materio takes place in an unconventional fashion, e.g. cellular automata-like (instead of conventional Turing Machine / von Neumann-like computation). The way computation is performed at the physical level is based on local interactions amongst neighboring molecules without a central controller. It may be possible to abstract such a process as a "cellular-automata-in-materio". As such, it may be possible to exploit materials to perform computation as a cellular automaton, e.g. evolve cellular automata transition tables of different complexity (see cellular automata classification from Wolfram / Langton) or embody a cellular automata into the material (this can be seen as exploiting the underlying intrinsic cellular automata in the material to execute on a given external input)."

1.2 Thesis Overview

Several experiments are presented in this thesis. First, the three logic gates AND, OR and XOR are attempted evolved in-materio. The hypothesis is that Evolution-in-Materio is feasible using the experimental process detailed in Chapter 3, and that the material-under-study is capable of evolved computation at least as complex as a single non-linear logic gate.

Second, Elementary Cellular Automata Rule 54, Rule 151, and Rule 110 are attempted evolved in-materio. The hypothesis is that the material-under-study is capable of computation at least as complex as the transition tables of these three cellular automata.

Third, all 256 Elementary Cellular Automata are attempted evolved in-materio. The hypothesis is that different cellular automata should be easier or more difficult to evolve based on how computationally complex they are, and that the ease of evolution may reveal details about the computational capacity of the material-under-study.

The thesis is organized in the following manner: First this introductory chapter presents the central theme, motivation and assignment text for this thesis. Then, a background chapter presents necessary background information, definitions and context on the state-of-the-art as they relate to the topic of this thesis in five different areas: unconventional computing, evolutionary computation, Evolution-in-Materio, cellular automata and logic gates. After the background exposition, each of the three sets of experiments performed for this thesis is described and the results of each are presented. Following this, a critical analysis of the experiment results and experimental method is performed. Finally, a conclusion is drawn, and suggestions for further work are presented.

1.3 Citation Style

This thesis uses a parenthetical, in-line, numerical citation style with fragment and paragraph citation forms. This means that numbered citations that appear within a sentence cover that sentence or sentence fragment alone, and a numbered citation that appears after the final sentence of a paragraph covers that entire paragraph.

Chapter 2

Background

2.1 Unconventional Computing

The term “computation” encompasses any type of calculation or information processing using computers, but when one speaks of computation it is normally in the context of conventional computing, i.e. computation performed on transistor-based von Neumann machines implemented as silicon microchips. Unconventional computing, however, is computation performed using other techniques and methods.

The dominant computer organization in traditional conventional computing today is based on the von Neumann architecture: computers synchronously read and execute instructions from a readable and writable memory, processing data until all instructions have been completed. This is a centralized model of computing. Conceptually, only a single instruction can executed at once, and each instruction must fully complete before the next one can begin. Modern implementations use a range of advanced and sophisticated techniques to circumvent this limitation by exploiting special cases (pipelining, out-of-order execution, and so on), but the conceptual model remains the same. [43]

As the development of smaller, faster, cheaper and more energy-efficient silicon-based von Neumann computers continues to push the frontiers of efficient computation, achieving the levels of progress we have grown accustomed to becomes harder and harder [57]. There are some intrinsic hard limits on increased improvements laid down by the physics of the material that hinder further progress at certain points. The effect of this is already apparent in the industry, with the big players scrambling to come up with complex solutions to emergent problems such as the dark silicon effect [13] and quantum tunneling side-effects [39]. Unconventional computers can challenge these shortcomings in new and interesting ways.

Most modern unconventional computing work follows one of two popular approaches. The first approach is to leverage the decades of work done on well-understood computing models such as the von Neumann model, but to use alternative physical materials as the computing substrate. This approach is promising, since it addresses one of the two core issues with conventional computing today, which is that transistor miniaturization is approaching the physical limits of silicon. It is possible that there may exist other materials in which cheaper, faster, more energy-efficient or less heat-dissipant transistors, or really, any type of logical switch, may be implemented. Indeed, such an advance has already happened before, in the 1950s, when the invention of the transistor some years earlier enabled transistor-based solid-state logic switches to surpass the vacuum tube-based logical switches that were used in early electrical computers such as the ENIAC [28]. Promising alternative materials for this purpose that are actively researched today include the use of graphene [52] or gallium arsenide [56] instead of silicon in microchips, fiber-optic-based logical switches that work on photons instead of electrons as the carrier of a logical signal [51], and various materials that may support memristor implementations [4, 7, 15, 24].

One obvious advantage with this approach is that an enormous library of existing concepts, tools, and software may be relatively trivially reapplied to new unconventional computers, since the fundamental computing paradigm remains the same even though the physical layer has been replaced.

The second approach is to investigate alternative theoretical computational models for use as the fundamental abstraction in place of the traditional von Neumann model. This approach is promising because it addresses the other core issue with conventional computing, one of the most limiting factors of the conventional computational model, which is the fact that programs must be executed sequentially. Being able to surpass this limitation in an efficient fashion could yield tremendous speedups for certain classes of parallelizable problems [22]. Alternative computational models may also lend themselves to efficient implementation in traditional computational substrate materials such as silicon, which means that existing production infrastructure and knowledge may be re-used. The cellular automata model is one such promising alternative model of computation [55].

It is also possible to combine these two approaches, i.e. implementing an alternative computational model in an unconventional computational substrate. This thesis attempts to do precisely this by looking for feasible implementations of an alternative theoretical computational model (cellular automata) in an unconventional computational material (a single-walled carbon nanotube and polymer composite mesh).

2.2 Evolutionary Computation

Evolutionary computation is an umbrella term for computation that involves optimization, adaptation and discovery using iterative processes guided by some metaheuristic. One powerful evolutionary algorithm is the Genetic Algorithm, which mimics Darwinian natural selection for its search heuristic. In it, a population of individuals are simulated through iterated cycles of life, death and reproduction, favoring individuals deemed as "fit" by some evaluation function for survival and reproduction into the next cycles. If each individual is taken as a representation of a candidate solution to a problem, and the evaluation function is able to reflect a candidate solution's closeness to an optimum solution, a Genetic Algorithm search will tend to converge the population toward a group of individuals representing solutions that are good approximations to the problem. [22]

In Genetic Algorithm terminology, *selection* is the operation of choosing an individual from a population based on some fitness-aware selection metric, *crossover* is the operation of combining two individuals to create a third, simulating natural reproduction, and *mutation* is the operation of changing a small part of an individual. A *genotype* is the genetic code of an individual, and a *phenotype* is the product of a genotype as it manifests as an fitness-measurable entity in an environment, i.e. the collection of an individual's observable traits.

Evolutionary computation can be used as a tool in systems engineering, and when embraced as a design paradigm is radically different from regular constructive top-down systems engineering. In the traditional top-down approach, a system is designed as a sum of its parts, using nested components as the primary abstraction to create simplicity and tractability in the design. Thus, it is necessary to understand systems completely in order to design them, which may demand great imagination, insight and precision from the engineer. An example of a system designed using the traditional top-down constructive engineering approach pertinent to the topic of this thesis is that of a traditional computer processor. A traditional computer processor is typically composed of several high-level components, which each are again composed of several intermediate-level components, which are again composed of several even lower-level components, all the way down to some fundamental component, traditionally the logic switch.

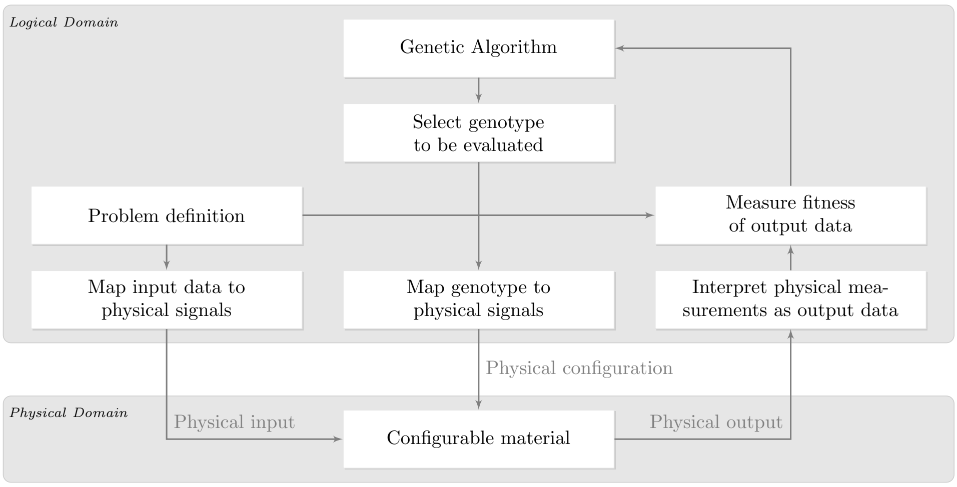
Evolutionary design, on the other hand, tends to result in complex, opaque, non-hierarchical designs that approximate solutions without having to be understood completely, or even at all. With evolutionary design, engineers specify desirable properties in the finished design, and let the stochastic evolutionary process come up with solutions that exhibit these desirable properties.

An example of a system designed using an evolutionary approach is NASA's evolved X-band antenna for use on the micro-satellites of the Space Technology 5 mission. There, an evolutionary algorithm was used to search the massive solution space of different possible antenna designs, and the algorithm was ultimately able to come up with antenna designs that were more performant than their traditionally designed counterparts. Additionally, while the initial evolutionary setup took 3 person-months to implement, new antennas could be evolved to meet new criteria in only 4 person-weeks, enabling a faster design cycle than what would be possible with a traditional design approach in the face of changing requirements, which in the end resulted in lower development costs. [21]

2.3 Evolution-in-Materio

Evolution-in-Materio is the exploitation of emergent computational behavior in physical materials through artificial evolution. The idea is that some physical processes inherent in different materials may be usefully interpreted as computation. Evolution-in-Materio attempts to harness this computational power using artificial evolution in order to discover favorable material configurations that may be efficiently exploited.

One of the earliest attempts at Evolution-in-Materio was conducted by cybernetician Gordon Pask in 1958. He attempted to create a physical signal processing device based on configurations of grown dendritic iron wires in a ferrous sulphate solution discovered through trial-and-error capable of processing sound or magnetic field signals. The device was eventually successful in discriminating between tones of 50Hz and 100Hz, a pioneer result in the then-unexplored engineering paradigm of heuristic search-driven systems design. [42]



Logical Domain
Genetic Algorithm
Select genotype to be evaluated
Problem definition
Measure fitness of output data
Map input data to physical signals
Map genotype to physical signals
Interpret physical measurements as output data
Physical configuration
Physical Domain
Physical input
Configurable material
Physical output

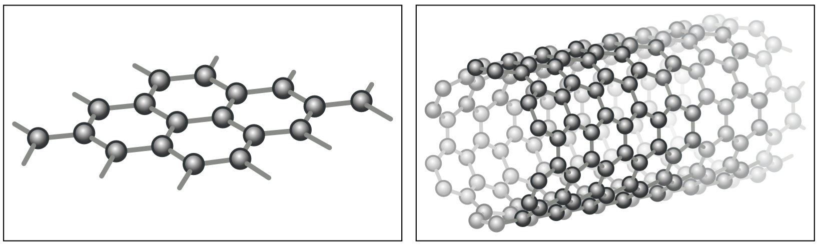
Figure 2.1: An overview of Evolution-in-Materio – artificial evolution is simulated in the logical domain, typically on a traditional computer, and fitness is evaluated by performing computations in the physical domain.

Modern Evolution-in-Materio was kicked off in 1996 when Thompson demonstrated that unconstrained evolution in a physically implemented logical system was able to exploit physical properties outside of the logical domain to improve fitness, and hence perform computation [50]. In his experiments, Thompson tried to use artificial evolution to configure a field-programmable gate array (FPGA) to perform a specific computational task: discriminate between 1kHz and 10kHz square waves. The hypothesis was that artificial evolution would be able to evolve a discrete logical circuit design capable of performing the computational task. Thompson managed to find an FPGA configuration that consistently solved the task, but upon inspection it became clear that the computation was not entirely performed in the discrete logical circuit. Instead, the computation relied on physical properties of the FPGA chip itself, outside of the discrete logical domain. This result shows that using evolution to design physically-implemented computers allows for a much larger design space than what is possible with traditional constructive engineering. This, in turn, opens the door to creating more efficient computational devices that to a greater extent exploit natural physical behaviors of the underlying computational substrate.

The term “Evolution-in-Materio” was coined in 2002 by Miller and Downing [33].

2.3.1 Current Materials

One challenge in the field of Evolution-in-Materio is discovering which materials are suitable for use as a computational substrate. A good material should preferably exhibit a number of properties that both enable computation and configuration so that evolution can be reliably performed. These properties include having a complex, practically (read: electronically) configurable semi-conducting structure in such a way that the material responds near-instantly and consistently to different inputs, as well as being robust to changes in



(a) Graphene **(b)** Carbon nanotube

Figure 2.2: An atom-scale visualization of a single-walled carbon nanotube. Each sphere represents a carbon atom, and the cylinders between them represent atomic bindings. The sheet to the left is graphene, and the cylinder to the right is graphene wrapped to form a carbon nanotube. Illustrations adapted with permission from original work by Jozef Sivek, distributed under a CC BY-SA 4.0 license.

the external environment such as lighting conditions, temperature and electromagnetic fields [41, 48]. It also helps if the material is cheap and easily available.

Some interesting candidate substrates that are currently under research are single-walled carbon nanotubes, liquid crystal matrices and silicon FPGA chips.

Single-Walled Carbon Nanotubes

Single-walled carbon nanotubes (SWCNT) are cylindrical carbon allotropes with unusual physical properties. They exhibit extraordinary electrical characteristics, which is interesting from an Evolution-in-Materio standpoint. Single-walled carbon nanotubes are engineered by wrapping a one-atom-thick sheet of graphene into a tube. The “angle” at which the nanotube is wrapped affects the electrical properties of the nanotube – some SWCNTs’ electrical conductivity show metallic conducting behavior, whilst others show different levels of semiconducting behavior. Determining how the different properties of a nanotube affects its electrical characteristics is an active area of research. [5, 9, 26, 49] An illustration of a SWCNT can be seen in Figure 2.2.

One way of using SWCNT as a substrate for computation is by arbitrarily arranging many SWCNTs in a random network and treating it as a single computational device [47]. The advantage of this is that it enables SWCNT mesh device production in large scale at the wafer level [14].

Single-walled carbon nanotube mesh devices have successfully been used as a substrate for computation in several experiments [10, 23, 36, 37, 38].

The material-under-study in the experiments presented in this thesis is a random single-walled carbon nanotube and polymer mesh material.

Liquid Crystal Matrices

Liquid crystals are matter that can exist in a mesomorphic state, carrying properties similar to both conventional liquids and solid crystal. For instance, the matter may behave like a liquid in terms of flow, all the while having molecules that are arranged in a distinct crystal-like fashion. Liquid crystals are relatively stable materials, but change their characteristics when subjected to electric fields. This makes them interesting subjects for Evolution-in-Materio, as it means that they can be configured to assume certain behaviors electronically.

Matrices of liquid crystals are today already mass-produced on a gigantic scale, because they are a core technology used in modern computer displays. This makes the material cheap and readily available, which is convenient.

Liquid Crystal Matrices have successfully been used as a substrate for computation in several experiments [17, 18, 19, 20].

Silicon Microchips

Even traditional silicon chips may be suitable for Evolution-in-Materio in certain cases. Reprogrammable silicon chips, known as Field-Programmable Gate Arrays, or FPGAs, exhibit the same characteristics that we are looking for in a good candidate for Evolution-in-Materio – they are electronically configurable and configurationally stable. One problem is that they seem to be easily affected by environmental changes when used for Evolution-in-Materio [50], which could make reliable computation difficult.

2.3.2 NASCENCE and the Mecobo Evolution-in-Materio Platform

NASCENCE, or *NAnoSCale Engineering for Novel Computation using Evolution*, is an EU-funded collaborative research project aiming to “model, understand and exploit the behavior of evolving nanosystems (e.g. networks of nanoparticles, carbon nanotubes or films of graphene) with the long-term goal to build information processing devices exploiting these architectures without reproducing individual components” [40]. One of the products that have emerged from the NASCENCE project is the Mecobo platform. It is a hardware and software platform for Evolution-in-Materio developed by Lykkebø et al. [27]. It is designed to interface with a large variety of materials, allowing Evolution-in-Materio fitness evaluation directly on a physical substrate. The Mecobo platform is used to facilitate the experiments presented in this thesis.

The Mecobo material interface works as a Voltage-time-signal sequencer and recorder that can apply and record arbitrary voltage patterns on its material-connected electrodes. In practical Evolution-in-Materio scenarios involving the Mecobo, the artificial evolution solver program is run on a traditional workstation computer. Each genotype in the population represents a candidate configuration for a given computational behavior in the material. During fitness evaluation of a genotype, the solver may want to execute some trial computations in the material using the configuration encoded in the genotype. The solver program on the workstation computer then develops the genotype-under-study into a phenotype which represents the material configuration in the physical domain, e.g. a series of static voltages that should be applied on a set of configuration electrodes into the material. The program also transforms the logical inputs to the trial computations into the physical domain as voltage pattern signals so that they can be applied on the material as well. When the configuration and computation inputs have been prepared for physical execution, they are communicated to the Mecobo, which applies the voltage signals upon its input and configuration electrodes, and performs a voltage pattern reading on its output electrodes.

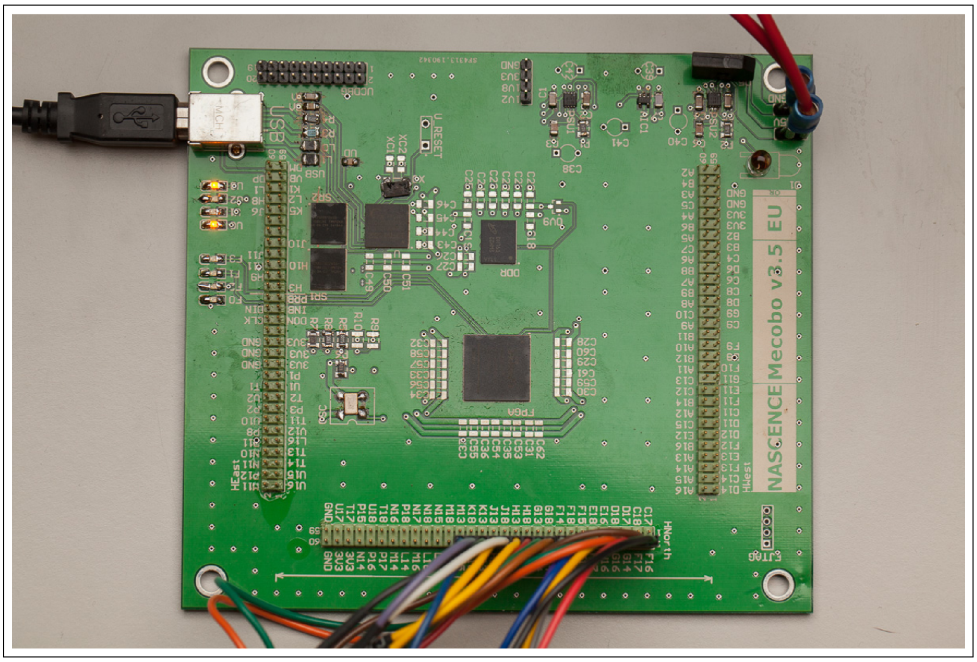


Figure 2.3: A photograph of the Mecobo. Figure adapted from [27] with permission.

The values sampled are transmitted back to the evolution solver program where they are interpreted as output in the logical domain and used together with other factors to composite a fitness score for the genotype-under-study. When used in this fashion, the Mecobo can be seen as a translation bridge between the logical and the physical world.

2.4 Cellular Automata

Cellular automata are abstract discrete n-dimensional dynamical systems that evolve over time. They consist of a graph of locally-connected nodes that each take on one of k discrete states in time step t. The state of a node n in the graph at time step t is given by the states of n's neighboring nodes at time step $t-1$. Each cellular automaton has a rule table describing how to transition from time step to time step.

2.4.1 Elementary Cellular Automata

In the simplest canonical case, a cellular automaton takes on the form of a one-dimensional array of binary cells where each cell has three neighbors: the cell immediately to the left, the cell immediately to the right, and itself, as illustrated in Figure 2.5. These specific cellular automata are called Elementary Cellular Automata [54]. There are 256 possible rule table permutations for the Elementary Cellular Automata. Of these 256 automata, 88 are fundamentally inequivalent [55, p. 57].

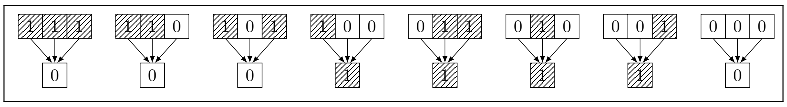


Figure 2.4: An example Elementary Cellular Automata state transition table laid out to illustrate the Wolfram Code numbering scheme. The top row of groups of three and three cells are the possible neighborhood states for a cell at time step $t-1$, and the cell beneath each group is the resulting state for that same cell at time step t. Reading the bottom row as a binary number ($00011110_2 = 30$) reveals the name of the automation: Rule 30.

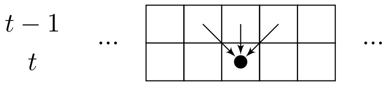


Figure 2.5: An example state-time representation of an Elementary Cellular Automaton. The state of a cell is decided by the state of its two neighboring cells and itself in the previous time step.

The Elementary Cellular Automata are given a numbering scheme known as the Wolfram Code in [54] that is rooted in binary number representation. Each of the $2^3 = 8$ possible neighborhood states for a given Elementary Cellular Automaton E are represented as each their binary number and ordered numerically. The resulting states for the next time step given from each of these neighborhood states are then taken in order as bits of a new binary number r. This number r is the number identifying E. As an example, Elementary Cellular Automaton Rule 30's rule table and corresponding Wolfram Code number is illustrated in Figure 2.4.

2.4.2 Sub-Elementary Cellular Automata

For the purpose of this thesis, it is useful to define an even simpler class of cellular automata. Consider the set of one-dimensional binary cellular automata with a brickwall neighborhood of size 2. That is, cells at every other time step are considered to be shifted 0.5 cell widths to the right, so that the state of a cell in time step t is decided by the state of the cells 0.5 cell widths at either side at time step $t-1$, as illustrated in Figure 2.6. There are 16 different cellular automata in this set. Dubbing these 16 cellular automata the Sub-Elementary Cellular Automata and numbering them analogously to the Elementary Cellular Automata, we can denote them uniquely as S0, S1, .., S15.

Using the cellular automata classification described in Section 2.4.4, S0, S8, S14 and S15 can be classified as Class I cellular automata, S1-S5, S7 and S10-S13 can be classified as Class II cellular automata, and S6 and S9 can be classified as Class III cellular automata.

2.4.3 Graphing One-Dimensional Cellular Automata

When examining cellular automata, it is useful to see how they behave over time. Conveying passage of time in complex graphs can sometimes be difficult when confined to a two-dimensional medium such as the one this thesis is written on. For the special case of one-dimensional cellular automata, only one dimension is needed for illustrating the state

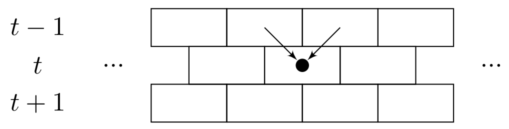


Figure 2.6: An example brickwall state-time representation of a Sub-Elementary Cellular Automaton. The cells at every other time step are considered to be shifted 0.5 cell widths to the right. The state of a cell is decided by the state of its two neighboring cells in the previous time step.

of the system in time step t. Therefore, we can use a second dimension available to us to show the passage of time. Conventionally, one-dimensional cellular automata behavior is graphed as a grid of cells where row t represents the entire state of the cellular automaton at time step t. This is called a time-space diagram [53]. All graphed cellular automatons in this thesis are graphed using time-space diagrams with randomly generated initial states.

2.4.4 Computation in Cellular Automata

The study of cellular automata is interesting due to their possible suitability as a vehicle for efficient computation. Although some cellular automata are proven to be capable of universal computation [55, pp. 644-656] [8, 11, 16, 45], perhaps the most interesting paradigm of computation is one where a cellular automaton is capable of efficiently computing a special non-general task. In this scenario, the program is embodied in the cellular automaton itself, manifested in the rule table, and possibly in the input state as well. Input to the computation is coded as an initial configuration of the cell states, and output is read from the state of the automaton at some time step $t > 0$. Cellular automata have in this fashion successfully been used to solve problems such as the Firing Squad Synchronization Problem [53], Parallel Formal-Language Recognition [34] and Parallel Arithmetic computation [34], in addition to multiple simulations of physical processes [3].

Determining which cellular automata are capable of useful computation is an active area of research [25]. Due to the chaotic nature of dynamic systems, the search space is seemingly largely unstructured, and finding useful computation is hard. Nevertheless, efforts have been made to classify cellular automata based on different criteria in order to reason about the computational properties of these classes – not a lot of conclusive results have been found, however.

Wolfram's Four Cellular Automata Classes

Wolfram attempts to classify cellular automata into four different classes based on the behaviors they seem to exhibit [55]. The class definitions are not strict in the mathematical sense, however, making it difficult to use for much else than a base for informal intuition building.

Class I cellular automata are cellular automata that tend to a stable homogeneous state. Randomness in the initial state tends to disappear as time progresses. Rule 32, which is shown in Figure 2.7a, is an example of a Class I cellular automaton.

Class II cellular automata are cellular automata that yield a sequence of simple stable or periodic structures. Randomness in the initial state is somewhat retained in periodic

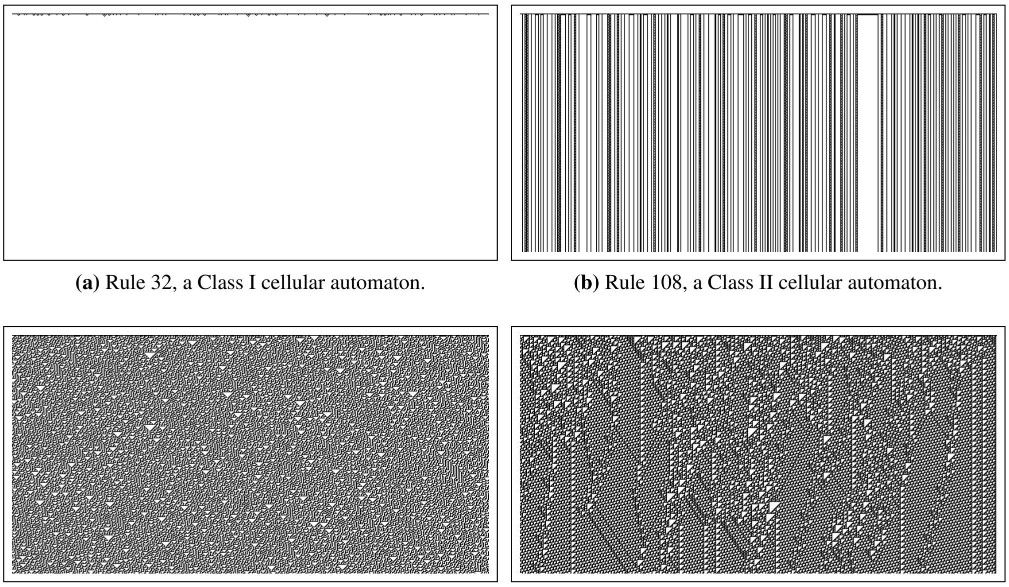

(a) Rule 32, a Class I cellular automaton.
(b) Rule 108, a Class II cellular automaton.
(c) Rule 30, a Class III cellular automaton.
(d) Rule 110, a Class IV cellular automaton.

Figure 2.7: Example Elementary Cellular Automata with random initial states.

structures. Changes made to the initial state tend to only have a local impact on the behavior of the cellular automata over time. Rule 108, which is shown in Figure 2.7b, is an example of a Class II cellular automaton.

Class III cellular automata are cellular automata that exhibit chaotic aperiodic behavior. Changes made to the initial state tend to have a global impact on the behavior of the cellular automata over time. Rule 30, which is shown in Figure 2.7c, is an example of a Class III cellular automaton.

Class IV cellular automata are cellular automata that yield complicated localized structures, some propagating. Changes made to the initial state tend to have a global impact on the behavior of the cellular automata over time. Wolfram postulates that most Class IV cellular automata are capable of general computation [55]. Examples of Class IV cellular automata include Rule 110 (shown in Figure 2.7d) and Conway's Game of Life [55].

Culik-Yu Classification

Culik et al. presented in [12] a more formal hierarchical classification based on Wolfram's four complexity classes. The classification relies on decidability theory. Informally, Class One contains all cellular automata that lead to a homogeneous state, Class Two contains all cellular automata that lead to an ultimate periodic evolution, Class Three contains all cellular automata for which it is decidable whether or not it is able to transition from state α to state β using zero or more intermediate states, and Class Four contains all cellular automata.

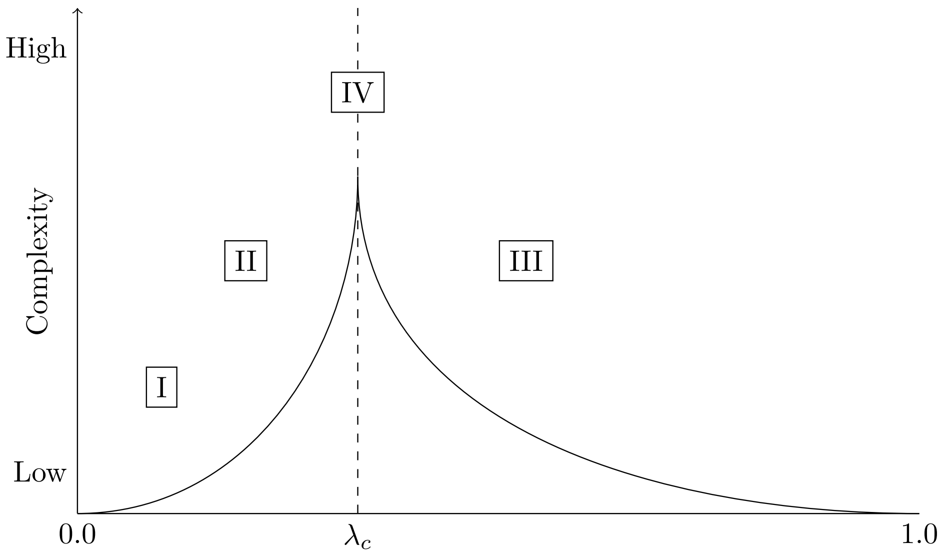


Figure 2.8: Location of Wolfram Classes in λ space, recreated from [25, Fig. 16].

λ-Parametrization and the Edge of Chaos

A different approach to cellular automata classification is through λ-parametrization [25]. The λ-parameter is a measure of what percentage of transitions in a cellular automaton's rule table go to an arbitrarily selected quiescent state. It is defined as

$$\lambda = \frac{K^N - n}{K^N} \tag{2.1}$$

given by (2.1), where K is is the number of different states a cell can have, N is the neighborhood size, and n is the number of state transitions that go the the quiescent state. For example, in Rule 30 of the Elementary Cellular Automata, taking state 0 as the quiescent state, 6 of 8 transitions go to the quiescent state. Since Elementary Cellular are binary cellular automata with a neighborhood size of 3, this means that

$$\lambda_{\text{Rule 30}} = \frac{2^3 - 6}{2^3} = 0.25 \tag{2.2}$$

of Rule 30 is as given by (2.2).

The idea is that cellular automata with similar λ-values tend to exhibit similar behaviors. This means that, as an example, Rule 129 of the elementary cellular automata with its $\lambda_{\text{Rule 129}} = 0.25$ should behave more similarly to Rule 30 than a Rule with a different λ, like Rule 85, which has a λ of $\lambda_{\text{Rule 85}} = 0.5$.

Langton examined a set of different 1-dimensional 4-state cellular automata with a neighborhood size of 5, and gave a qualitative classification on the behavior of the cellular automata in relation to their λ-values [25]. Langton observed that low λ-values tended to give cellular automata with a high amount of order, reaching a steady (potentially periodic) state quickly, and high λ-values tended to give cellular automata exhibiting chaotic behavior. Langton goes on to argue that the cellular automata most suitable for computation will be found at the boundary between these two extremes in behavior, at the "Edge-of-Chaos". In Fig. 16 of [25], recreated here as Figure 2.8, Langton illustrates the relationship between the computational complexity in cellular automata with regards to the λ-parameter and the possible locations in λ space of cellular automata of different Wolfram Classes. The λ-parametrization's significance in identifying computationally interesting cellular automata at the "Edge-of-Chaos" was later disputed by Mitchell et al., who suggested that the original findings are not properly reproducible [35].

2.4.5 Cellular Automata as a Physical Abstraction

The great advantage of computation in-materio is that it offers the possibility of performing computation "directly" in the material, as opposed to in some abstracted computational model implemented in a material. The latter will necessarily discard a large part of the computational power of the substrate as a consequence of the abstraction. It seems, then, that exploiting direct in-materio computation is optimal in terms of computation power. However, direct computation in-materio can be quite difficult, especially if universality and scalability is desired. There is an apparent trade-off between efficient usage of the computational complexity in the substrate and ease of programmability for practically useful results which seems to be related to the intuition that computational potential is lost in the abstraction from substrate to theoretical model. The larger the disaffinity between the abstract model and the physical processes in the material becomes, the larger the inefficiency in translation will grow. Thus, finding an abstract computational model that closely matches the physical properties of a material might minimize the computational gap between the physical and the abstract. [48]

Cellular automata is a promising abstract computational model for this purpose. Just like they seem to be in physical materials, the computational processes in cellular automata are massively parallel in a distributed and localized fashion. This is a completely different paradigm than the centralized model found in conventional computing.

2.5 Binary Logic Gates

A logic gate is a device that implements a Boolean function. It provides a mapping from a number of Boolean inputs to a single Boolean output. [46, p. 760]

Physically implemented logic gates over silicon transistors is the main building block used in mainstream microchip computers [43]. Most logic gates in practical use today are binary logic gates – they have two Boolean inputs and a single Boolean output. Binary logic gates are usually commutative, i.e. changing the order of the operators does not change the result. Formally, this means that for some binary operator $\star$, $a \star b = b \star a$. There are 8 possible commutative binary logic gates: FALSE, AND, XOR, OR, NOR, NXOR, NAND and TRUE. The truth tables for each of the 8 possible binary logic gates can be seen in Table 2.1.

Table 2.1: Truth Tables for Each of the 8 Possible Binary Logic Gates

I_A	I_B	O_{FALSE}	O_{AND}	O_{XOR}	O_{OR}	O_{NOR}	O_{NXOR}	O_{NAND}	O_{TRUE}
0	0	0	0	0	0	1	1	1	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Binary logic gates are combinatorial devices, which means that output given by binary logic gates depend only on the current input, not on any internal state [46, p.761]. They do in other words have no form of stateful memory capabilities.

2.5.1 Complexity

Different binary logic gates can perform computations of different complexity. The FALSE and TRUE logic gates always output the same answer, 0 and 1 respectively, regardless of input. This makes them the least complex binary logic gates. AND, OR, NAND and NOR are linearly separable. XOR and NXOR are not linearly separable, which in a sense makes them the most complex binary logic gates.

Chapter 3

Experiments

For this thesis, three series of experiments have been performed in an attempt to measure and understand the feasibility and utility of evolution in-materio on a mixed single-walled carbon nanotube and polymer substrate using cellular automata as tool for reasoning about computational power.

The first series of experiments aim to evolve logic gates in the material, as a first stepping stone on the way to cellular automata. This is done in an attempt to verify that the material is capable of performing simple evolved computation, and to measure the computational stability of the material.

In the second experiment series dubbed the an attempt is made at evolving three specific Elementary Cellular Automata in the material. The motivation for this is to investigate the limits of computational complexity in the material when used as a computational device within the specific constraints put forth by the method of computation used in this thesis.

A third series of experiments try to explore the computational capacity of the material by doing a broader evolutionary sweep of all 256 Elementary Cellular Automata in the material, enabling existing knowledge about Elementary Cellular Automata complexity to be used in gauging the computational capacity of the material-under-study.

In the proceeding sections, the experimental setup of the different experiments is detailed – first in a general sense explaining the common approach used in the experiments, and then in the special case for each experiment. The latter sections also include the results of each of the experiments.

3.1 Experimental Setup

On a high level, each of the experiments is implemented in a similar fashion: an electrode-wired material sample glass slide containing a sample of the material-under-study is connected to a scientific instrument called Mecobo [27]. This instrument is able to apply arbitrary voltage patterns onto the electrodes connected to the material, and observe voltage levels on other electrodes connected to the same material. The Mecobo is connected to an ordinary desktop computer running custom software to command and control the Mecobo instrument. The experiments model the material as a stateful computational device that can accept voltage series over electrodes as input over time and deliver computation results as voltage levels read at specific points. The challenge of realizing computation in the material using this model then boils down to finding a meaningful way to encode and decode computational input and output to and from time-voltage signals in order to transform the problem instances to be computed from the logical domain to the physical domain and back. A heuristically guided genetic algorithm search is employed to evolve the configuration required for a specific computational function in each of the experiments. Hence, realizing computation in-materio can be seen as a two-step process: first, the evolution of a stable configuration that enables the desired computation in the material, which is a process that might take some time (hours), and second, the execution of a single instance of the problem with some input, which is relatively fast (milliseconds).

3.1.1 Mecobo

The experiments were performed using the Mecobo v3.5 Evolution-in-Materio hardware and software platform [27]. On boot, the Mecobo FPGA is flashed with the firmware version whose compiled bitfile md5 checksum is `78561f68ff096c394213bd703abb-15a3`. The Mecobo motherboard was connected by USB to a Ubuntu 12.10 [6] desktop computer running the `host` [2] Mecobo interface / Thrift [1] server. The Genetic Algorithm solver functions as a Thrift client, and connects to the Thrift server in order to interact with the material that is connected to the Mecobo.

3.1.2 Genetic Algorithm Overview

The Genetic Algorithm solver used for the experiments is a custom generational Genetic Algorithm solver implemented in Python [44]. The algorithm maintains two populations of individuals: a child population and an adult population. For each generation, individuals are selected using some adult selection scheme for promotion to become members of a new adult population. Adult individuals are then selected using some parent selection scheme to be combined and create offspring using some crossover scheme, which are taken to be the new child population. Finally, child individuals are mutated according to some mutation scheme. Fitness is lazily calculated for individuals when a value is required (e.g. for selection), and is cached for an individual for as long as it is alive. That is, an individual is not fitness-evaluated more than once. An overview of the algorithm can be seen in Figure 3.1.

The solver is configurable in a variety of parameters, and thus each experiment described in this report describes the configuration used.

Adult Selection Schemes

The solver can be configured to use one of three different adult selection schemes: full generational replacement, over-production, and generational mixing. With full generational replacement, every adult currently in the adult population is removed from the population, and every child in the child population is promoted to adult status and added to the adult population.

With over-production, every adult currently in the adult population is removed from the population, and some children from the child population are promoted to adult status and added to the adult population.

With generational mixing, some adults currently in the adult population are removed from the population, and some children from the child population are promoted to adult status.

Parent Selection Schemes

The solver can be configured to use one of four different parent selection schemes.

Fitness-proportionate selection selects individuals by random chance, weighted by the individual's fitness. With this scheme, an individual with a high fitness is more likely to get selected than an individual with a low fitness. The probability

$$P(\text{selection}) = \frac{f(i)}{\sum_{n=1}^{|I|} f(I_n)} \tag{3.1}$$

for an individual i to be selected given a fitness function f from a population I is given by (3.1), where $|I|$ is the size of the population and I_n is the n^{th} individual in I.

Sigma-scaling selection is similar to fitness-proportionate selection, in that it selects individuals by random chance, weighted by the individual's fitness, but it scales an individual i's fitness $f(i)$ by a value s_i, signifying its relation to statistical properties about the population's fitnesses. The sigma-scaled fitness function

$$f'(i) = 1 + \frac{f(i) - f(\bar{I})}{2\sigma} \tag{3.2}$$

given by (3.2), where $f(\bar{I})$ is the average fitness for the population and σ is the standard deviation of the fitnesses in the population. The probability of selection for an individual i is then given by (3.1), substituting f with f'.

Tournament selection looks at k randomly chosen individuals from the population, and with probability $1 - \epsilon$ selects the fittest individual of these k individuals, or else selects a random individual from these k individuals.

Rank selection sorts all the individuals of the population by fitness, and selects an individual at random weighted by it's position in the ordered sequence of individuals, known as an individual's *rank*. The fittest individual i in a population of size n has a rank given by $i_r = n$. The least fit individual j in a population of size n has a rank given by $j_r = 1$. The probability

$$P(\text{selection}) = \frac{2i_r}{n(n-1)} \tag{3.3}$$

for an individual i to be selected from a population of size n is given by (3.3).

Crossover Schemes

The solver can be configured to use one of two different crossover schemes.

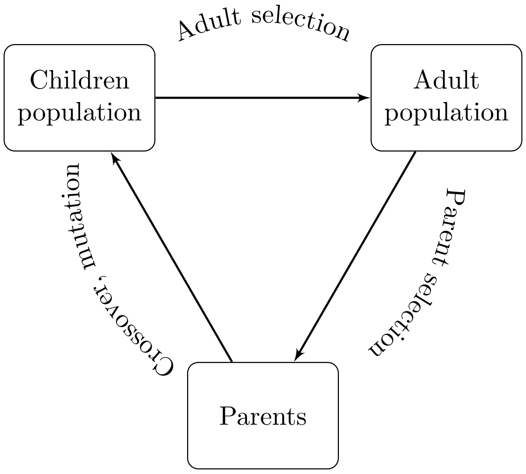


Figure 3.1: A flow diagram representing the path taken by individuals through the Genetic Algorithm solver.

Split crossover combines two individuals by copying with probability p the first n symbols of the first parent, and the remaining symbols of the second parent, else every symbol from the first parent. p is selected by configuration on a per-experiment basis, whilst n is a random number between 0 and the length of an individual's symbol vector generated for each crossover operation.

Genome component crossover combines two individuals by copying with probability p each symbol from either the first or second parent, chosen randomly with equal probability for each parent for each symbol, else every symbol from the first parent. Again, p is selected by configuration on a per-experiment basis.

Mutation Schemes

The solver can be configured to use one of two different mutation schemes.

Per-genome mutation mutates an individual with probability p selected by configuration on a per-experiment basis.

Per-genome component mutation mutates each single component of an individual (i.e. each gene) individually with probability p selected by configuration on a per-experiment basis.

The mutation operation for both schemes is configurable in p, and is specified per-experiment.

3.1.3 Material Overview

The computational substrate material used in the experiments is a random mesh of single-walled carbon nanotubes mixed with poly(butyl methacrylate) (PBMA) and dissolved in anisole (methoxybenzene) embedded in a glass plate with 16 gold electrodes into the material arranged in a 4x4 grid with 50 µm contacts and 100 µm pitch. The material, slide #1

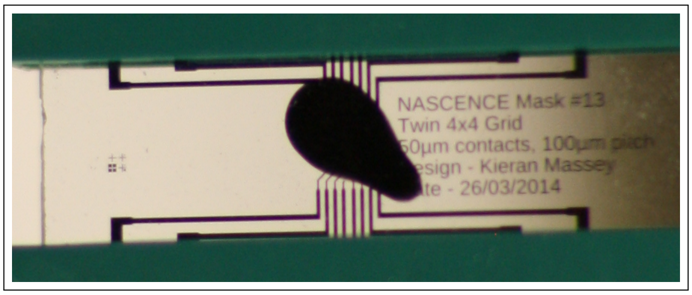


Figure 3.2: A photograph of the SWCNT material on its glass slide.

of batch #15, numbered B15S01, was prepared by Kieran Massey at Durham University by the following method: “20 µl of material are dispensed onto the electrode area; This is dried at 85°C for 30 min to leave a *thick film*; The hotplate is turned off and the substrates are allowed to cool slowly over a period of roughly 2 h to room temperature.” The carbon nanotube concentration in the material is 0.75%. The electrode configuration is “Mask #13 - Twin 4x4 grid”. All electrodes show connection resistances on the order of 20 kΩ, but it is reasonable to assume that the nanotube coverage over the electrodes is noticeably uneven, given the nanotube concentration level. [31]

Figure 3.2 shows a photograph of the material on its glass slide.

The material is a product of the NASCENCE project [40].

3.2 Evolving Logic Gates in-Materio

This experiment attempts to evolve Boolean binary logic gates in the material to see if the material is capable of performing simple evolved computation, and to measure the computational stability of the material. Three Boolean logic gates were evolved: the AND gate, the OR gate, and the XOR gate. The first two gates are implementations of linearly separable Boolean functions, whilst the third is not. The input-to-output-mapping function for each of the three evolved gates can be seen in Table 3.1.

Interestingly, a binary Boolean logic gate input-to-output-mapping function can also be seen as the transition function of a Sub-Elementary Cellular Automata. As such, evolving the AND, OR and XOR logic gates is equivalent to evolving the state transition functions of the S8, S14 and S6 Sub-Elementary Cellular Automata, respectively.

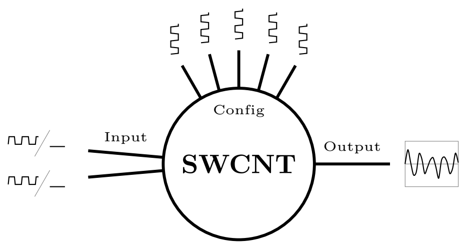


Figure 3.3: Input, output and configuration mapping of the material interface electrodes for logic gate computation.

3.2.1 Input Coding

Seven electrodes into the SWCNT/polymer composite material were used as input electrodes. Two of these electrodes represent the two inputs a and b to the binary Boolean function. Each of these two inputs a and b can be either a logical 0 or a logical 1. A logical 0 on an input electrode is signified by applying a selected constant static voltage, the exact value of which is decided on a per-solution basis. A logical 1 on an input electrode is signified by applying a digital square wave with a frequency of 10000Hz and a 3.3 V amplitude, from 0 V to 3.3 V. The other five input electrodes function as material configuration inputs, with a digital pulse wave with a specific frequency and duty cycle chosen on a per-solution bases from a predetermined range being applied upon each of them.

3.2.2 Output Coding

The computational output expected from a Boolean logic gate is a single Boolean value: 0 or 1. A single electrode into the material was used as an output electrode. The voltage output was sampled at 500000Hz for 80 ms from the 10th to the 90th ms of computation. The samples were averaged arithmetically and compared to a predetermined threshold value. If the average was higher than the threshold value plus some small empirically determined padding to reduce measurement noise, the Boolean output was interpreted as a logical 1. Conversely, if the average was lower or equal to the padded threshold value, the Boolean output was interpreted as a logical 0. The threshold value was obtained experimentally by running a calibration sweep of the material, which consisted of performing 200 logic gate computations with randomly generated inputs conforming to the coding specified in Section 3.2.1. The averaged output was calculated for each computation, and the median of these averages were taken to be the threshold value.

Which specific physical electrodes were used for which of the input and output signals was decided by a logical-to-physical mapping of electrodes coded in the evolvable genotype. An overview of the SWCNT/polymer composite material with its logical input and output electrodes can be seen in Figure 3.3.

Table 3.1: Logic Gates Input-to-Output-Mapping Function

I_1	I_0	O_{AND}	O_{OR}	O_{XOR}
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

3.2.3 Genetic Algorithm Configuration

Genotype Representation

The evolvable genotype was represented as a symbol vector of length 20 with 256 possible values for each symbol. The first 5 symbols of the genotype represent material configuration pulse wave frequencies. The next 5 symbols of the genotype represent material configuration pulse wave duty cycles. The next 8 symbols represent the pin mapping. Then follows a single symbol representing the static amplitude of logical low coding signal. The last symbol represents an output interpretation threshold offset. The genotype is illustrated in Figure 3.4.

Taking a symbol's value to be a numerical integer value between 1-256, letting G_n denote the n^{th} symbol of a genotype G, the pulse wave frequency

$$f_n = (G_n + 1) \times 100\text{Hz} \tag{3.4}$$

of a configuration electrode C$_n$ is given by (3.4). The pulse wave duty cycle

$$d_n = \frac{\lfloor G_{n+5} \times \frac{100}{255} \rfloor}{100} \tag{3.5}$$

of a configuration electrode C$_n$ is given by (3.5). The static voltage level

$$s = (G_{18} - 127) \times 0.039\,\text{V} \tag{3.6}$$

of a logical low input signal is given by (3.6). The center threshold offset

$$c = (G_{19} - 127) \times 0.156\,\text{V} \tag{3.7}$$

is given by (3.7).

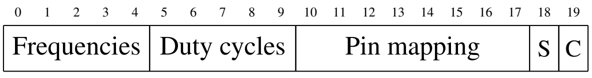


Figure 3.4: The genotype mapping for the logic gate experiment. S is the static amplitude of a logical low input signal and C is the center threshold offset.

Genetic Algorithm Parameters

The Genetic Algorithm solver was run with parameters as described in Table 3.2.

Table 3.2: Genetic Algorithm Solver Parameters for Logic Gate Evolution

Parameter	**Configuration**
Children pool size	40 individuals
Adult pool size	40 individuals
Adult selection scheme	Generational mixing
Parent selection scheme	Tournament, $\epsilon : 0.05, k : 8$
Crossover scheme	Per-genome component, p: 0.5
Mutation scheme	Per-genome component, p: 0.01

Fitness Function

The fitness of a genotype is given as function of how well it can compute the binary computation. For all four possible input combinations to the binary logic gate (0, 0), (0, 1), (1, 0) and (1, 1), the input and configuration electrodes were activated simultaneously for 100 ms. Samples were recorded for 80 ms, from the 10th to the 90th ms of the total activated time. For each set of inputs, the interpreted output was compared to the expected output from an ideal binary logical gate. If the output matched, 2 points were awarded to the total fitness. If the output was undecided (i.e., the measured output average was within the noise-reducing padding area around the threshold), 1 point was awarded to the total fitness. If the output otherwise did not match, 0 points were awarded to the total fitness. Finally, the fitness score was normalized to fit a 0-1 range by dividing the total score by 8.

The fitness function does not account for stability, e.g. by performing the same fitness evaluation multiple times and averaging the result.

3.2.4 Results

Computationally stable AND, OR and XOR logical gates were successfully evolved. To the best of the author's knowledge, this is the first time that stable XOR logic gates (and other gates) are evolved in SWCNT materials [23, 27]. Many different configurations yielding different logic gates were found. One configuration for each of the three types of logic gate is here selected for further examination. These three configurations are shown in Figure 3.5.

Although the search space in this experiment is quite large, there seems to be a lot of satisfactory solutions amongst the candidates. The search space contains 256^{20} possible candidates, yet in a typical Genetic Algorithm run with a population size of only 40 individuals, the solver is able to frequently find satisfactory solutions in the first or second generations, when the individuals are still mostly random.

Stability

Each of the three evolved solutions in Figure 3.5 were tested for correctness by performing repeated computations in-materio over all four possible inputs. Each solution was tested for 2000 repeated calculations over each of the four possible inputs, for a total of 8000 tests.

The evolved OR gate computed the correct value 8000 out of 8000 times, resulting in an estimated error rate of $0.0000\% \pm 0.0000$ *pp* at a 95% level of confidence. The evolved AND gate computed the correct value 8000 out of 8000 times, resulting in an estimated error rate of $0.0000\% \pm 0.0000$ *pp* at a 95% level of confidence. The evolved XOR gate computed the correct value 7955 out of 8000 times, resulting in an estimated error rate of $0.5625\% \pm 0.1639$ *pp* at a 95% level of confidence.

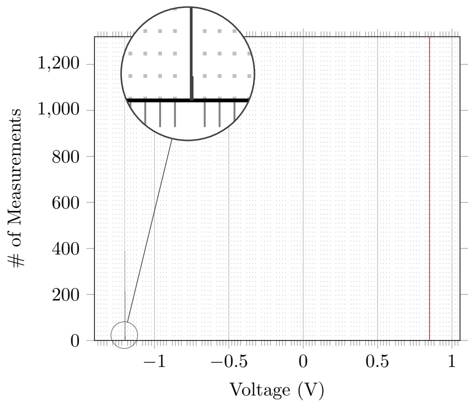
Histograms of the averaged physical output values measured in the stability tests are shown in Figure 3.7, Figure 3.6 and Figure 3.8.

Example Computations

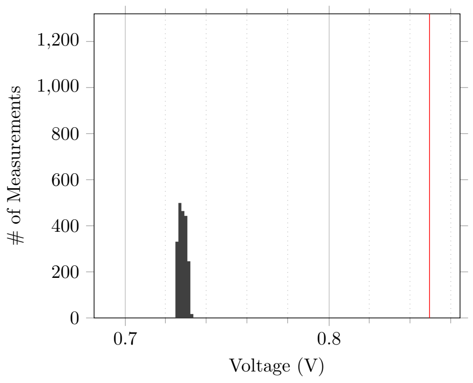
Here, a set of recorded AND, OR and XOR computations are presented. The material configuration for each of the three computations is the configuration coded in each of the three genotypes in Figure 3.5. Figure 3.10 shows the state of the (logically mapped) input and output electrodes over the course of the execution of the computation of the logical task $1 \wedge 0$. Figure 3.12 shows the state of the (logically mapped) input and output electrodes over the course of the execution of the computation of the logical task $1 \vee 0$. Figure 3.14 shows the state of the (logically mapped) input and output electrodes over the course of the execution of the computation of the logical task $1 \oplus 0$. Figure 3.9 shows samples recorded from four AND calculations performed in-materio. Figure 3.11 shows samples recorded from four OR calculations performed in-materio. Figure 3.13 shows samples recorded from four XOR calculations performed in-materio. The entire recording for each calculation is in reality 40000 samples long, but for practicality reasons only the first 500 samples are plotted here. The horizontal black lines represent the measured total average of the entire recording for each calculation. The horizontal magenta line together with a small gray padding band represents the threshold value above or under which a recorded average is taken to be a logical 1 or 0, respectively.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$\wedge$ {	233	47	146	232	47	148	236	179	195	76	83	214	37	108	127	203	125	147	29	127
$\vee$ {	45	127	211	252	28	249	246	191	201	185	161	120	116	11	207	83	43	86	168	127
$\oplus$ {	103	118	150	147	154	88	220	10	157	148	198	155	119	119	69	114	175	225	254	233

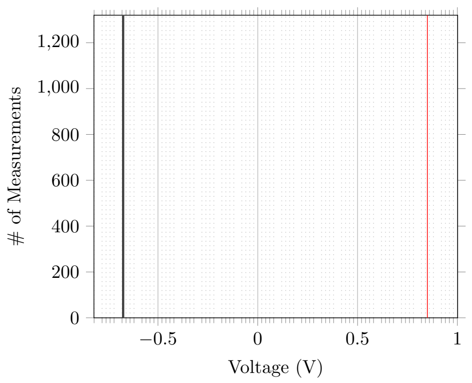
Figure 3.5: Genotypes representing successfully evolved $\wedge$, $\vee$, and $\oplus$ logic gates.



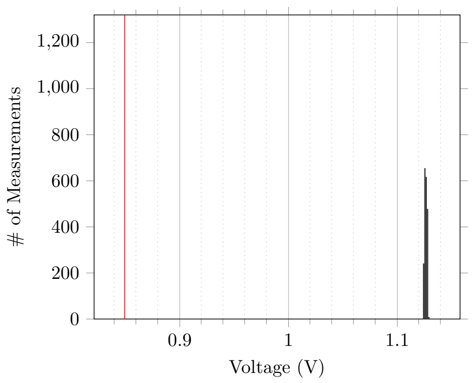
(a) $and(00_2)$



(b) $and(01_2)$

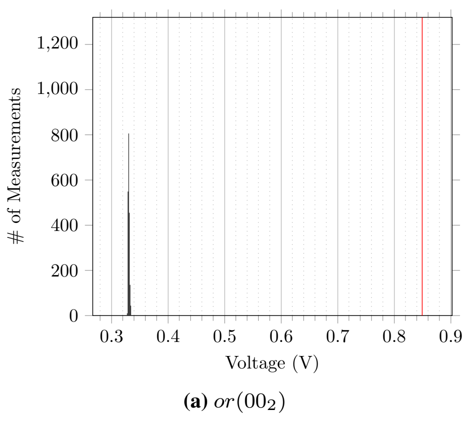


(c) $and(10_2)$

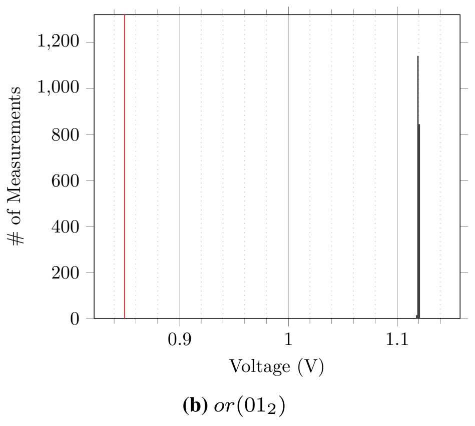


(d) $and(11_2)$

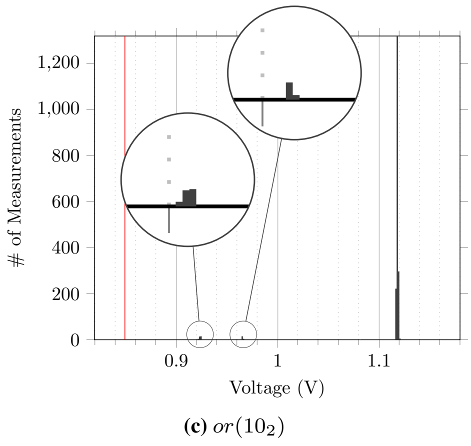
Figure 3.6: Histograms representing the measured average outputs over 8000 calculations using the evolved AND configuration, grouped by input. The output threshold used for mapping the measured output into the binary logical domain is plotted as a red vertical line in each histogram. Measurements that result in a wrong computational answer are in magenta. Interesting details are magnified. Each histogram bin is roughly $0.0012\,\mathrm{V}$ wide.



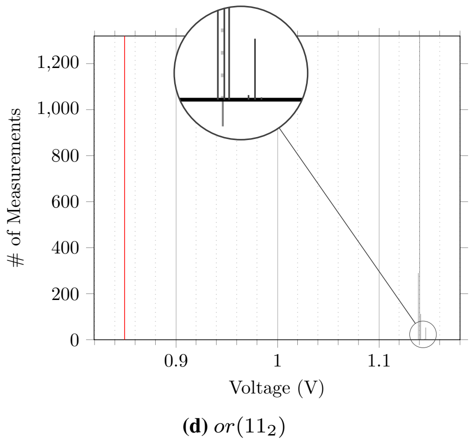
(a) $or(00_2)$



(b) $or(01_2)$

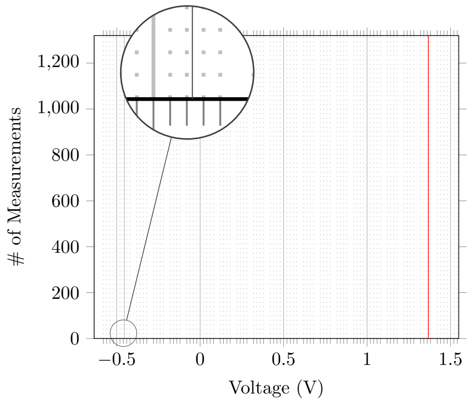


(c) $or(10_2)$

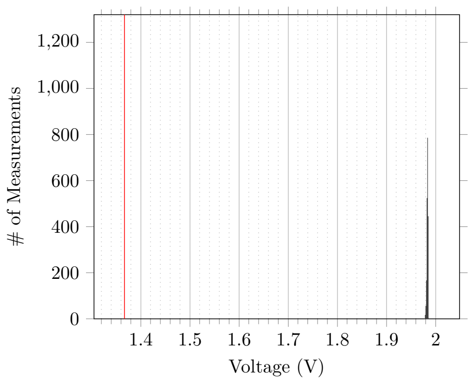


(d) $or(11_2)$

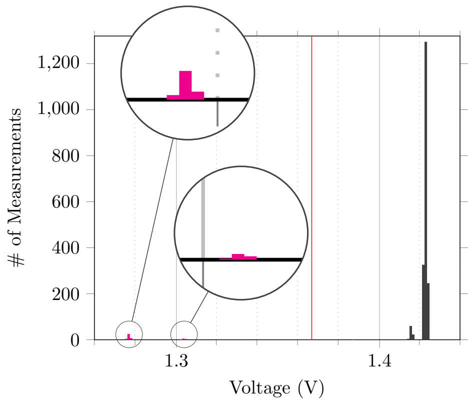
Figure 3.7: Histograms representing the measured average outputs over 8000 calculations using the evolved OR configuration, grouped by input. The output threshold used for mapping the measured output into the binary logical domain is plotted as a red vertical line in each histogram. Measurements that result in a wrong computational answer are in magenta. Interesting details are magnified. Each histogram bin is roughly $0.0012\,\mathrm{V}$ wide.



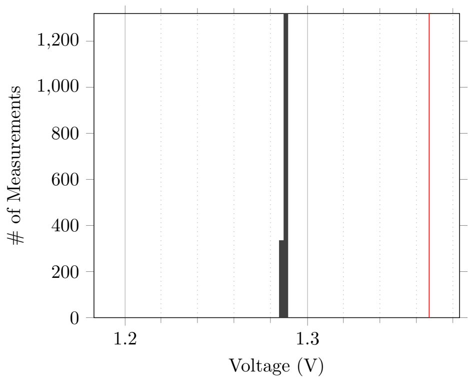
(a) $xor(00_2)$



(b) $xor(01_2)$



(c) $xor(10_2)$



(d) $xor(11_2)$

Figure 3.8: Histograms representing the measured average outputs over 8000 calculations using the evolved XOR configuration, grouped by input. The output threshold used for mapping the measured output into the binary logical domain is plotted as a red vertical line in each histogram. Measurements that result in a wrong computational answer are in magenta. Interesting details are magnified. Each histogram bin is roughly $0.0012\,\mathrm{V}$ wide.

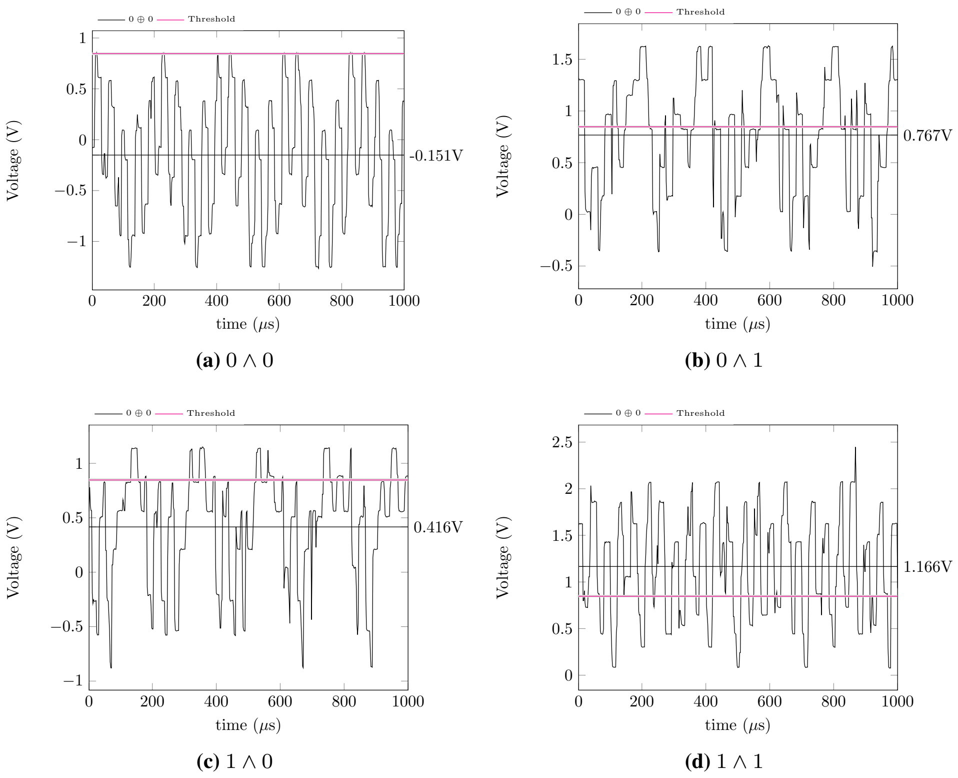
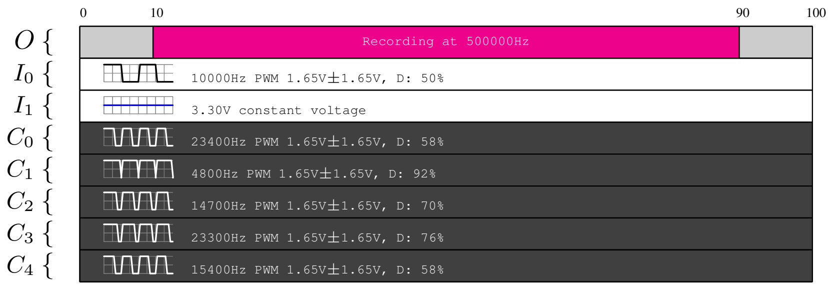
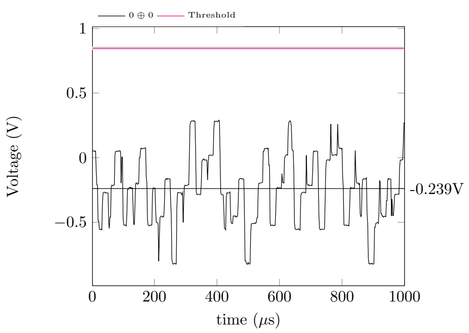


Figure 3.9: The first 500 raw material output samples recorded during four AND computations. The horizontal black lines represent average values for a sample series, and the horizontal red lines represent the output interpretation threshold value, which is 0.8496V.

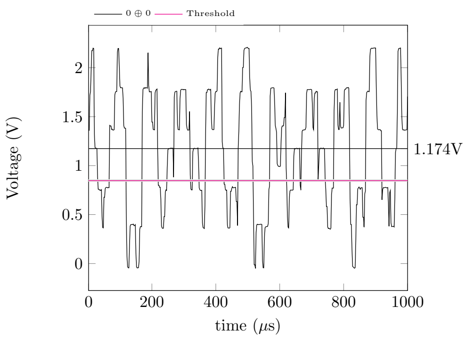


0
10
90
100
O
I_0
I_1
C_0
C_1
C_2
C_3
C_4
Recording at 500000Hz
10000Hz PWM 1.65V±1.65V, D: 50%
3.30V constant voltage
23400Hz PWM 1.65V±1.65V, D: 58%
4800Hz PWM 1.65V±1.65V, D: 92%
14700Hz PWM 1.65V±1.65V, D: 70%
23300Hz PWM 1.65V±1.65V, D: 76%
15400Hz PWM 1.65V±1.65V, D: 58%

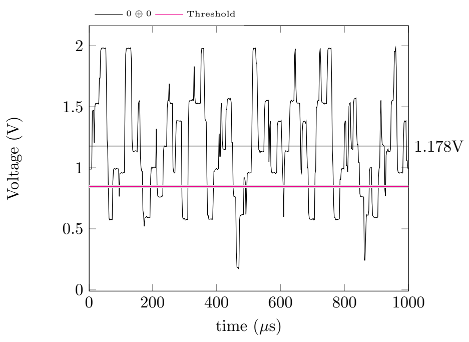
Figure 3.10: Timing overview of the activity on each logical electrode over time during a single computation of $1 \wedge 0$ on the evolved AND gate. Time progresses along the x-axis, labeled in milliseconds at points of interest. O is the output pin, I_n are the input pins, and C_n are the configuration pins. The small waveform illustrations reflect the duty cycle, but not the frequency. `D` is the duty cycle.



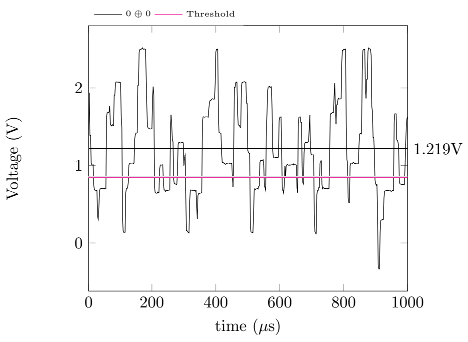
(a) $0 \vee 0$



(b) $0 \vee 1$



(c) $1 \vee 0$



(d) $1 \vee 1$

Figure 3.11: The first 500 raw material output samples recorded during four OR computations. The horizontal black lines represent average values for a sample series, and the horizontal red lines represent the output interpretation threshold value, which is 0.8496V.

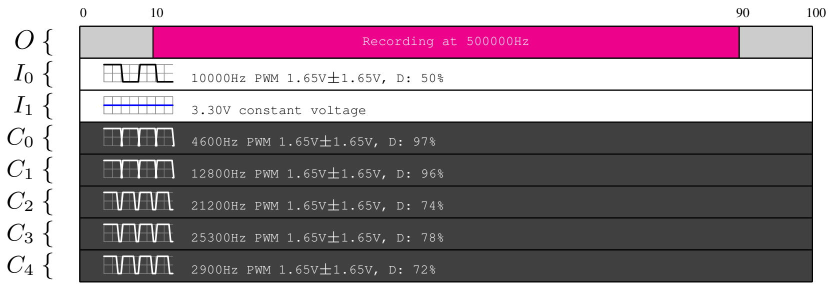
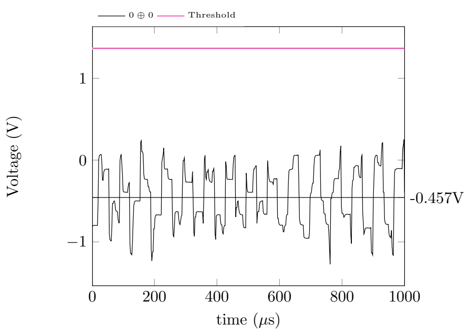
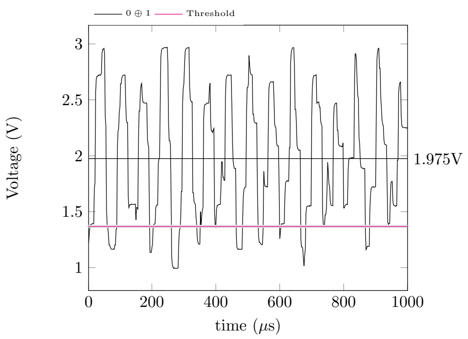

0
10
90
100
O {
I0 {
I1 {
C0 {
C1 {
C2 {
C3 {
C4 {
Recording at 500000Hz
10000Hz PWM 1.65V±1.65V, D: 50%
3.30V constant voltage
4600Hz PWM 1.65V±1.65V, D: 97%
12800Hz PWM 1.65V±1.65V, D: 96%
21200Hz PWM 1.65V±1.65V, D: 74%
25300Hz PWM 1.65V±1.65V, D: 78%
2900Hz PWM 1.65V±1.65V, D: 72%

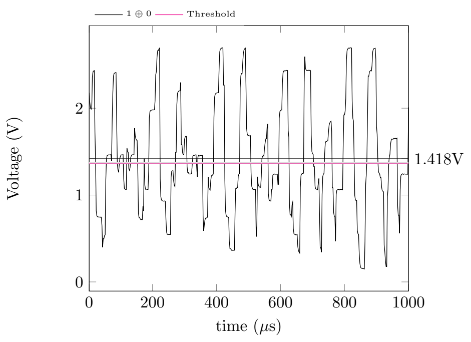
Figure 3.12: Timing overview of the activity on each logical electrode over time during a single computation of $1 \vee 0$ on the evolved OR gate. Time progresses along the x-axis, labeled in milliseconds at points of interest. O is the output pin, I_n are the input pins, and C_n are the configuration pins. The small waveform illustrations reflect the duty cycle, but not the frequency. `D` is the duty cycle.



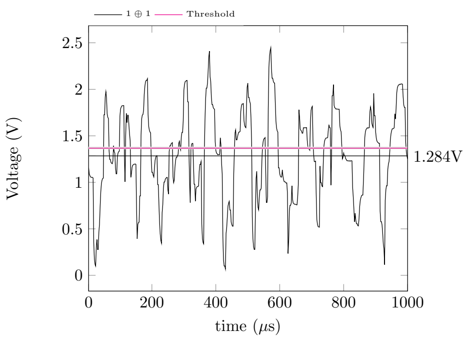
(a) 0 ⊕ 0



(b) 0 ⊕ 1

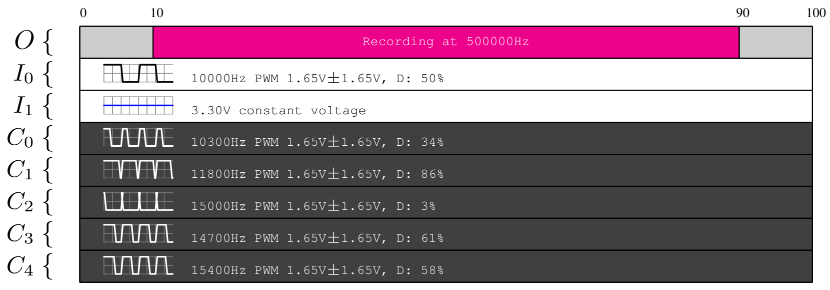


(c) 1 ⊕ 0



(d) 1 ⊕ 1

Figure 3.13: The first 500 raw material output samples recorded during four XOR computations. The horizontal black lines represent average values for a sample series, and the horizontal red lines represent the output interpretation threshold value, which is 1.367V.



0
10
90
100
O
Recording at 500000Hz
I_0
10000Hz PWM 1.65V±1.65V, D: 50%
I_1
3.30V constant voltage
C_0
10300Hz PWM 1.65V±1.65V, D: 34%
C_1
11800Hz PWM 1.65V±1.65V, D: 86%
C_2
15000Hz PWM 1.65V±1.65V, D: 3%
C_3
14700Hz PWM 1.65V±1.65V, D: 61%
C_4
15400Hz PWM 1.65V±1.65V, D: 58%

Figure 3.14: Timing overview of the activity on each logical electrode over time during a single computation of 1 ⊕ 0 on the evolved XOR gate. Time progresses along the x-axis, labeled in milliseconds at points of interest. O is the output pin, I_n are the input pins, and C_n are the configuration pins. The small waveform illustrations reflect the duty cycle, but not the frequency. `D` is the duty cycle.

Table 3.3: Input-to-Output-Mapping Functions for the Selected Elementary Cellular Automata

I_2	I_1	I_0	$O_{\text{Rule 54}}$	$O_{\text{Rule 151}}$	$O_{\text{Rule 110}}$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	0	1	0

3.3 Evolving Elementary Cellular Automata in-Materio

This experiment attempts to evolve select Elementary Cellular Automata transition functions in-materio, to test the hypothesis that computing with cellular automata in a single-walled carbon nanotube and polymer composite material is an interesting avenue for further research. Of particular interest are the Elementary Cellular Automata in Class III and Class IV of Wolfram's Elementary Cellular Automata classification, the former because they can be capable of performing complex computations [29], and the latter because they are famously conjectured by Wolfram to be capable of universal computation [55]. Two stable Elementary Cellular Automata have been evolved: Rule 151, a chaotic Class III cellular automata, and Rule 54, a complex Class IV cellular automata conjectured but not yet proven to be computationally universal [55, p. 697][30]. An evolution of Rule 110, a complex Class IV Elementary Cellular Automata for which a proof of universality of computation was formulated by Cook in [11], was also attempted, but no suitable solution was found.

Like in the logical gate experiment, the computational function can be thought of as a mapping from binary inputs to a binary output. The expected input-to-output mappings for Rule 54, Rule 151 and Rule 110, are given in Table 3.3, where I_n is input n, and $O_{\text{Rule } n}$ is the output for Rule n. The values are given in the logical domain.

3.3.1 Input Coding

Just as in the logic gate experiment, seven electrodes into the SWCNT material were used as input electrodes. This time, three of these electrodes represent the state of the three cells in the neighborhood set for an elementary cellular automaton rule transition. Each of these three inputs can either be a logical 0 or a logical 1, coded as a static voltage or a digital pulse applied to the input electrodes, respectively. The parameters for the static voltages and digital pulses are the same as in the logic gate experiment. The other four input electrodes function as a static material configuration, each electrode being applied upon a digital pulse signal chosen from a specific range of frequency and duty cycle combinations,

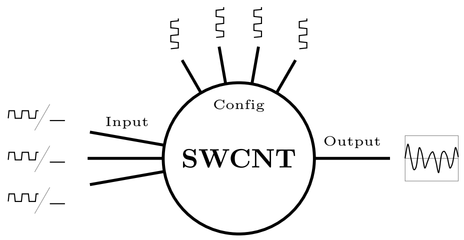


Figure 3.15: Input, output and configuration mapping of the material interface electrodes for Elementary Cellular Automata computation.

like in the logic gate experiment.

3.3.2 Output Coding

The computational output expected from any binary cellular automaton transition function is a single Boolean value: 0 or 1. This is the same output range as can be expected from a binary logical gate. Therefore, the output interpretation scheme for this experiment is the exact same as the one in the logic gate experiment, recalibrated for the input coding specified in Section 3.3.1 over 200 computations.

An overview of the logical electrode mapping can be seen in Figure 3.15.

3.3.3 Genetic Algorithm Configuration

Genotype

The evolvable genotype was represented as an 18 symbol long symbol vector with 256 possible values for each symbol. The first 4 symbols of the genotype represent pulse wave frequencies. The next 4 symbols of the genotype represent pulse wave duty cycles. The next 8 symbols represent the pin mapping. The penultimate symbol represents the static voltage level of a logical low input signal. The final symbol represents a center offset. This genotype coding is similar to the one used in the logic gate experiment, except that it has been adjusted for the different number of input and configuration electrodes.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Frequencies				Duty cycles				Pin mapping								S	C

Figure 3.16: The genotype mapping for this experiment. S is the static amplitude of a logical low input signal, and C is the center threshold offset.

Similar to the logic gate experiment, taking a symbol's value to be a numerical integer value between 1-256, letting G_n denote the n^{th} symbol of a genotype G, the pulse wave frequency

$$f_n = (G_n + 1) \times 100\text{Hz} \tag{3.8}$$

Table 3.4: Genetic Algorithm Solver Parameters for Elementary Cellular Automata Evolution

Parameter	Configuration
Children pool size	40 individuals
Adult pool size	40 individuals
Adult selection scheme	Generational mixing
Parent selection scheme	Tournament, $\epsilon : 0.05, k : 8$
Crossover scheme	Per-genome component, p: 0.5
Mutation scheme	Per-genome component, p: 0.1

of a configuration electrode C$_n$ is given by (3.8). The pulse wave duty cycle

$$d_n = \frac{\lfloor G_{n+4} \times \frac{100}{255} \rfloor}{100} \tag{3.9}$$

of a configuration electrode C$_n$ is given by (3.9). The static voltage level

$$s = (G_{16} - 127) \times 0.039\,\mathrm{V} \tag{3.10}$$

of a logical low input signal is given by (3.10). The center threshold offset

$$c = (G_{17} - 127) \times 0.156\,\mathrm{V} \tag{3.11}$$

is given by (3.11).

Genetic Algorithm Parameters

The Genetic Algorithm solver was run with parameters as described in Table 3.4. This setup is similar to the setup in the logic gate experiment, but with a higher mutation rate.

3.3.4 Fitness Function

The fitness of a genotype is given as a function of how well it can compute the target cellular automaton's state transition function. For all eight possible input combinations to the transition function the input and configuration electrodes were activated simultaneously for 100 ms. Samples were recorded for 80 ms, from the 10th to the 90th ms of the total activated time. The squared scaled confidence value

$$c_n = \frac{(o_n - T)^2}{5^2} \tag{3.12}$$

for input combination n is then calculated for each input combination as defined in (3.12), where T is the output threshold value, and o_n is the averaged output value for input combination n. For each set of inputs, the interpreted output is compared to the expected output from the input-to-output mapping function. If the output matches, c_n points are awarded to the total fitness. If the output is undecided, 0 points are awarded to the total fitness. If the output otherwise does not match, $-10c_n$ points are awarded to the total fitness. Then, if all outputs were correct, an additional 8 points are awarded to the total fitness. Finally, the fitness is transposed by further 8 points, and then scaled by a factor of 0.0625. Ultimately, this results in a fitness score which is greater than or equal to 1 if and only if all outputs are correct.

The fitness function does not account for stability, e.g. by performing the same fitness evaluation multiple times and averaging the result.

3.3.5 Results

Computationally stable Rule 54 and Rule 151 transition functions have been found. No suitable solution was found for Rule 110. The evolved solution configurations for Rule 54 and Rule 151 are shown in Figure 3.17.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Rule 54 {	160	217	183	55	99	200	160	12	115	180	166	205	30	89	175	119	243	138

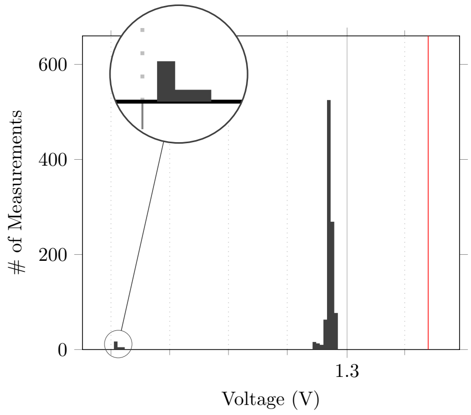
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Rule 151 {	149	220	16	174	123	119	75	157	28	51	163	154	138	115	125	36	219	2

Figure 3.17: Genotypes representing successfully evolved Rule 54 and Rule 151 state transition functions.

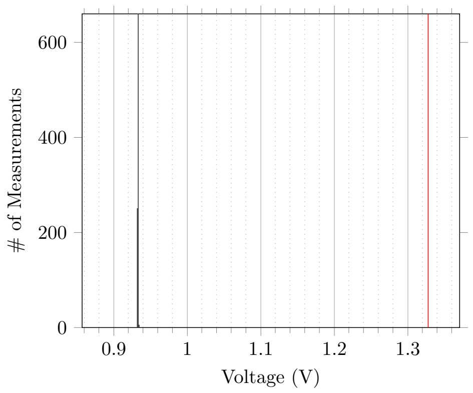
The Rule 151 solution was found in the 2nd generation of the Genetic Algorithm solver. The Rule 54 solution was found in the 5th generation of the Genetic Algorithm solver.

Stability

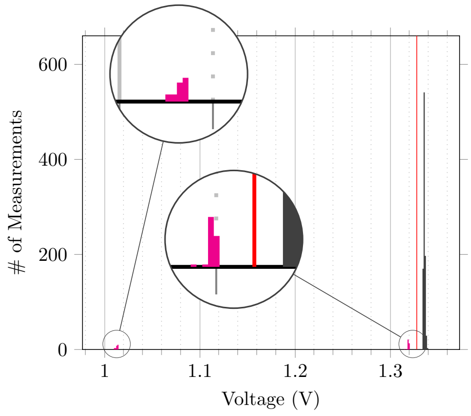
Both of the two evolved solutions in Figure 3.17 were tested for correctness by performing repeated computations in-materio over all eight possible inputs. Each solution was tested for 1000 repeated calculations of each of the eight possible inputs, for a total of 8000 tests. The evolved Rule 54 computed the correct value 7917 out of 8000 times, resulting in an estimated failure rate of $1.0375\% \pm 0.2221$ *pp* at a 95% level of confidence. The evolved Rule 151 computed the correct value 8000 out of 8000 times, resulting in an estimated failure rate of $0.0000\% \pm 0.0000$ *pp* at a 95% level of confidence. Figure 3.16 and Figure 3.15 show the levels of each of the measured average outputs for each of the 8000 calculations grouped by input for the evolved Rule 54 and the evolved Rule 151, respectively.



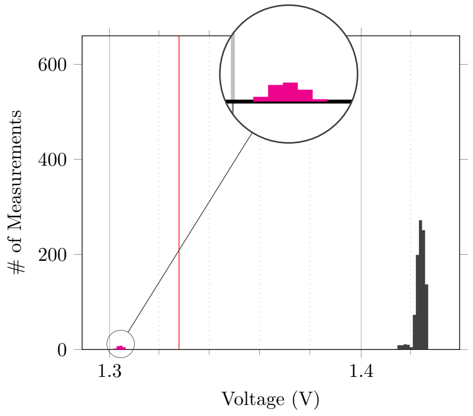
(a) $r54(000_2)$



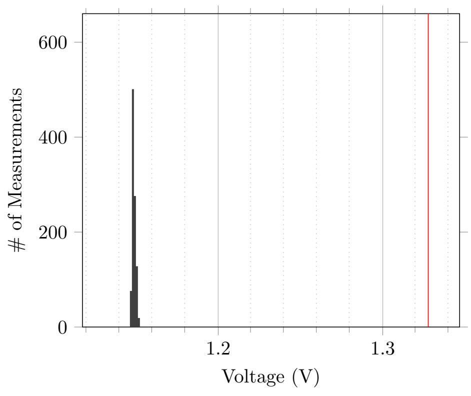
(b) $r54(001_2)$



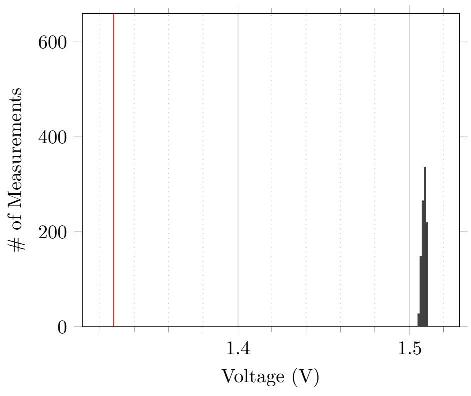
(c) $r54(010_2)$



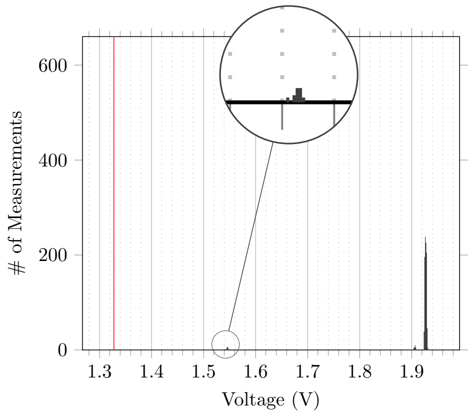
(d) $r54(011_2)$



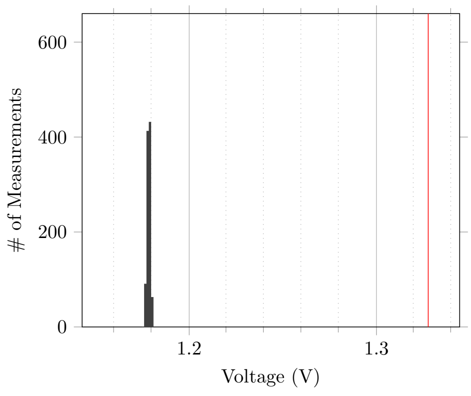
(e) $r54(100_2)$



(f) $r54(101_2)$

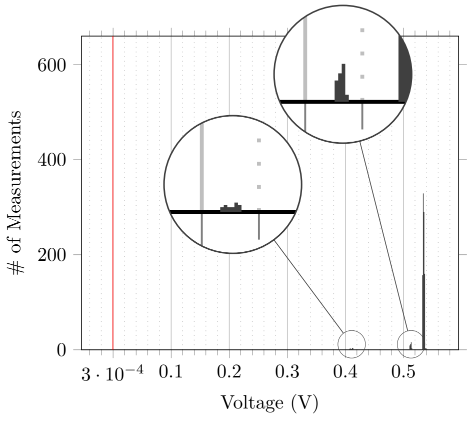


(g) $r54(110_2)$

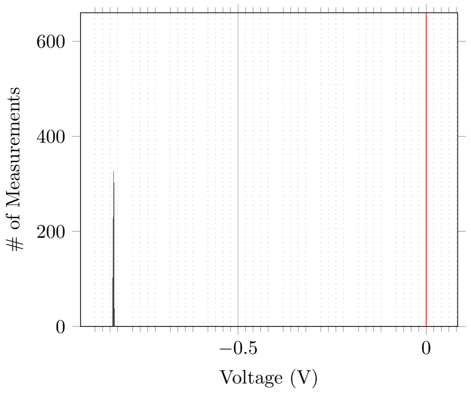


(h) $r54(111_2)$

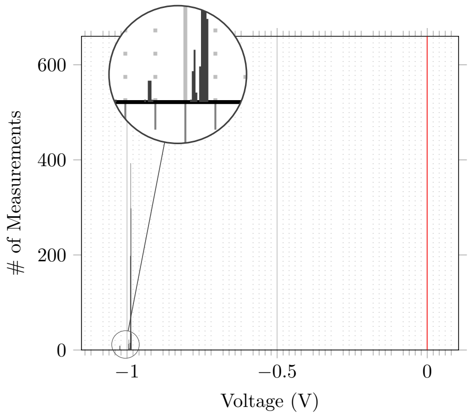
Figure 3.16: Histograms representing the measured average outputs over 8000 calculations using the evolved Rule 54 configuration, grouped by input. The output threshold used for mapping the measured output into the binary logical domain is plotted as a red vertical line in each histogram. Measurements that result in a wrong computational answer are in magenta. Interesting details are magnified.



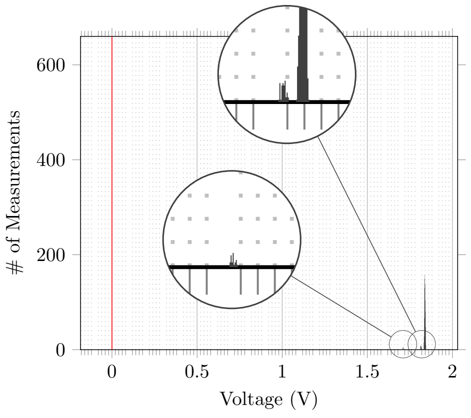
(a) $r151(000_2)$



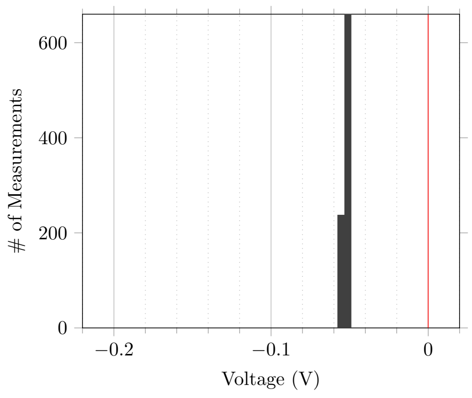
(b) $r151(001_2)$



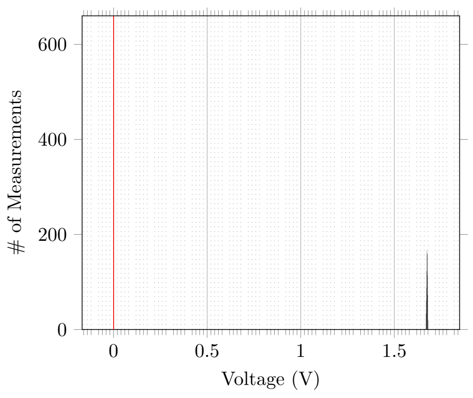
(c) $r151(010_2)$



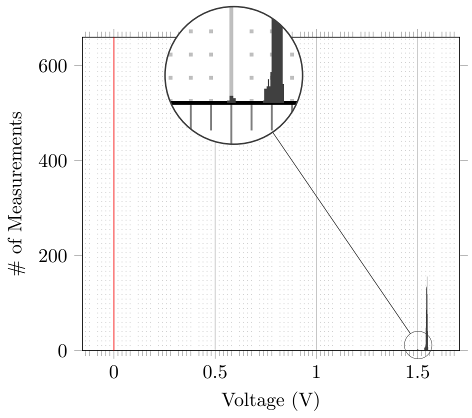
(d) $r151(011_2)$



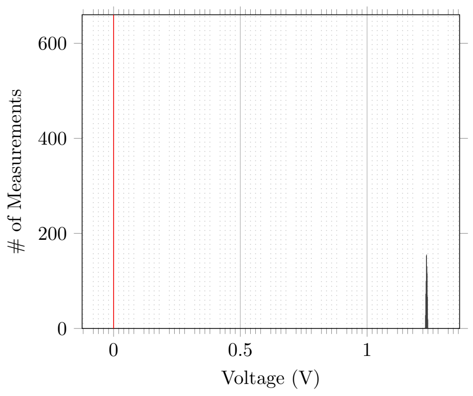
(e) $r151(100_2)$



(f) $r151(101_2)$



(g) $r151(110_2)$

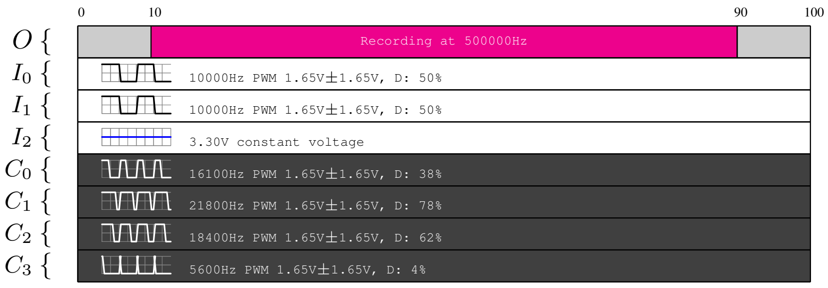


(h) $r151(111_2)$

Figure 3.15: Histograms representing the measured average outputs over 8000 calculations using the evolved Rule 151 configuration, grouped by input. The output threshold used for mapping the measured output into the binary logical domain is plotted as a red vertical line in each histogram. Interesting details are magnified.

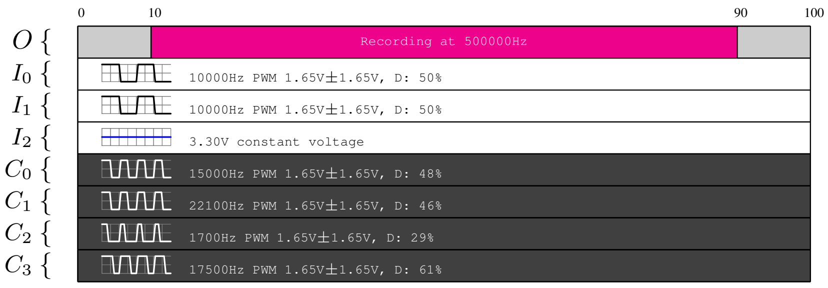
Example Computations

Here, a set of recorded Rule 54 and Rule 151 computations are presented. The material configurations are the configurations coded in the Rule 54 and Rule 151 genotypes in Figure 3.17. Figure 3.16 and Figure 3.17 shows the states of the (logically mapped) input and output electrodes over the course of the execution of the computation of the Rule 54 and Rule 151 transition functions for the neighborhood state 011_2. Figure 3.18 shows samples recorded from eight Rule 54 calculations performed in-materio. Figure 3.19 shows samples recorded from eight Rule 151 calculations performed in-materio. Again, like with the logic gates, the entire recording for each calculation is in reality 40000 samples long, and again for practicality reasons only the first 500 samples are plotted here. The horizontal black lines represent the measured total average of the entire recording for each calculation. The magenta threshold together with a small gray padding band represents the threshold value above or under which a recorded average is taken to be a logical 1 or 0, respectively.



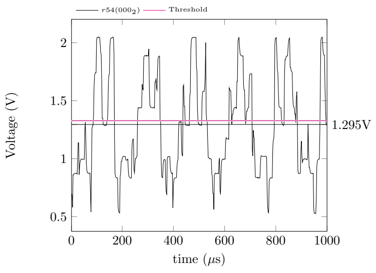
0
10
90
100
O {
Recording at 500000Hz
I_0 {
10000Hz PWM 1.65V±1.65V, D: 50%
I_1 {
10000Hz PWM 1.65V±1.65V, D: 50%
I_2 {
3.30V constant voltage
C_0 {
16100Hz PWM 1.65V±1.65V, D: 38%
C_1 {
21800Hz PWM 1.65V±1.65V, D: 78%
C_2 {
18400Hz PWM 1.65V±1.65V, D: 62%
C_3 {
5600Hz PWM 1.65V±1.65V, D: 4%

Figure 3.16: A timing overview of the activity on each logical electrode over time during a single computation of r54(011_2) on the evolved Rule 54 device. Time progresses along the x-axis, labeled in milliseconds at points of interest. O is the output pin, I_n are the input pins, and C_n are the configuration pins. The small waveform illustrations reflect the duty cycle, but not the frequency. `D` is the duty cycle.

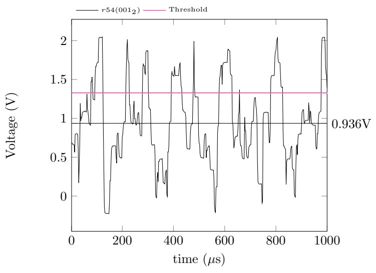


0
10
90
100
O {
Recording at 500000Hz
I_0 {
10000Hz PWM 1.65V±1.65V, D: 50%
I_1 {
10000Hz PWM 1.65V±1.65V, D: 50%
I_2 {
3.30V constant voltage
C_0 {
15000Hz PWM 1.65V±1.65V, D: 48%
C_1 {
22100Hz PWM 1.65V±1.65V, D: 46%
C_2 {
1700Hz PWM 1.65V±1.65V, D: 29%
C_3 {
17500Hz PWM 1.65V±1.65V, D: 61%

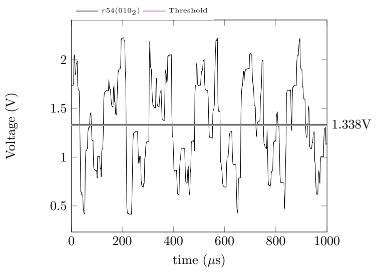
Figure 3.17: A timing overview of the activity on each logical electrode over time during a single computation of r151(011_2) on the evolved Rule 151 device. Consult the Figure 3.16 caption for a legend.



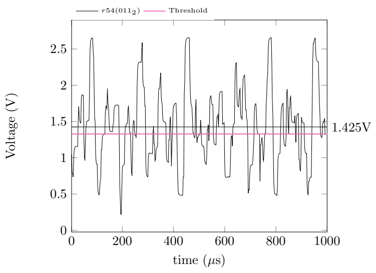
(a) $r54(000_2)$



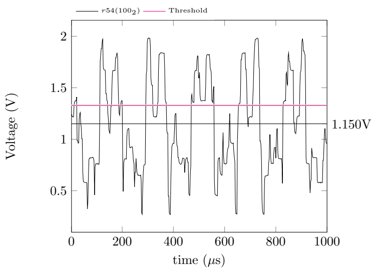
(b) $r54(001_2)$



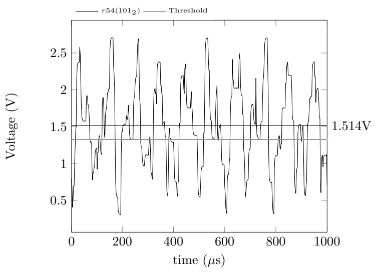
(c) $r54(010_2)$



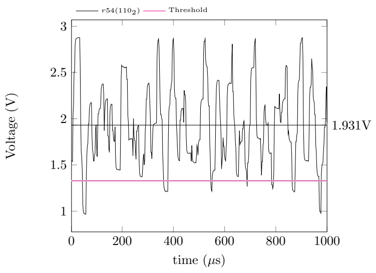
(d) $r54(011_2)$



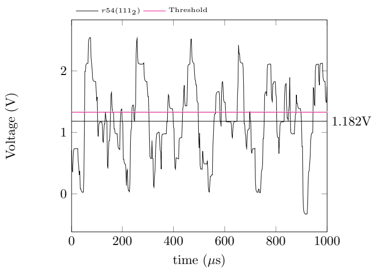
(e) $r54(100_2)$



(f) $r54(101_2)$

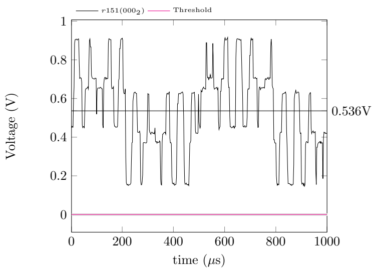


(g) $r54(110_2)$

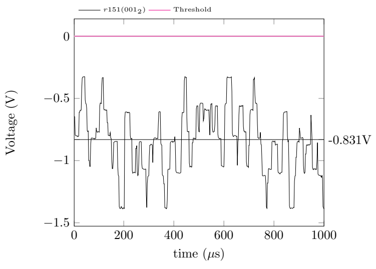


(h) $r54(111_2)$

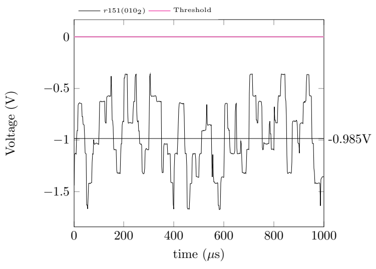
Figure 3.18: The first 500 raw material output samples recorded during eight Rule 54 computations. The black horizontal lines represent average values for a sample series, and the red horizontal line is the output threshold, which is 1.328V.



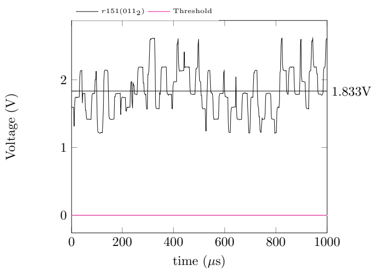
(a) $r151(000_2)$



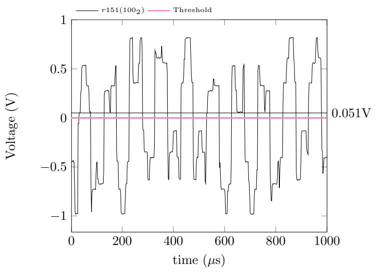
(b) $r151(001_2)$



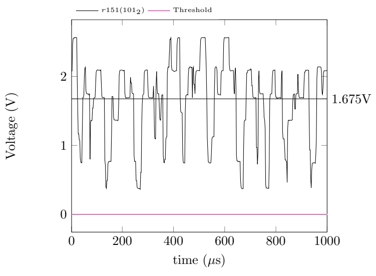
(c) $r151(010_2)$



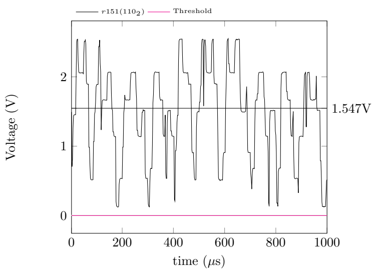
(d) $r151(011_2)$



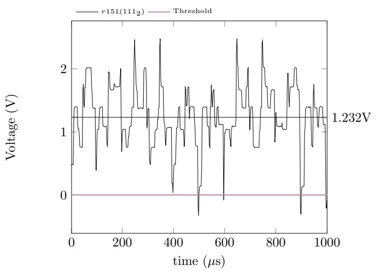
(e) $r151(100_2)$



(f) $r151(101_2)$



(g) $r151(110_2)$



(h) $r151(111_2)$

Figure 3.19: The first 500 raw material output samples recorded during eight Rule 151 computations. The black horizontal lines represent average values for a sample series, and the red horizontal line is the output threshold, which is 0.000V.

3.4 Evolving All Elementary Cellular Automata in-Materio

This experiment attempts to evolve all the 256 different Elementary Cellular Automata transition functions in-materio, one by one. This is done in an attempt to measure the complexity ceiling of the material by using evolvability of the different rules as a proxy indicator.

The experimental setup is similar to that of the individual logic gate and elementary cellular automata transition function experiments detailed in previous sections. Expected input-to-output mappings were constructed for each of the 256 rules and used as a target for the evolutionary algorithm.

The electrode configuration used in the evolution of each of the 256 elementary cellular automata is identical to the configuration used in the previous single elementary cellular automata experiments.

The calibration value obtained manually in the previous experiments was kept at the same fixed value as in the previous elementary cellular automata experiments for each of the 256 rules. This is because the calibration value does not vary much as long the electrode configuration and input-output coding remains the same.

An overview of the logical electrode mapping can be seen in Figure 3.15.

3.4.1 Input Coding

The input coding was identical to the one detailed in Section 3.3.1.

3.4.2 Output Coding

The output coding was identical to the one detailed in Section 3.3.2.

3.4.3 Genetic Algorithm Configuration

Genotype

The evolvable genotype format was the same as in the previous Elementary Cellular Automata experiment described in Section 3.3.

Genetic Algorithm Parameters

The parameters for the genetic algorithm are somewhat altered from the previous single Elementary Cellular Automata experiments. Mostly, the parameters have been tuned with the goal of reducing the amount of time needed to run each experiment, since there is a practical limit to how much time experiments can be allowed to run during the course of a Master's thesis. Specifically, the population sizes have been decreased and the maximum generation cutoff has been greatly reduced.

The Genetic Algorithm solver was run with parameters as described in Table 3.5. Aside from the changes mentioned above, this setup is identical to the setup in the previous experiment.

Table 3.5: Genetic Algorithm Solver Parameters for All Elementary Cellular Automata Evolution

Parameter	**Configuration**
Children pool size	20 individuals
Adult pool size	20 individuals
Adult selection scheme	Generational mixing
Parent selection scheme	Tournament, ϵ : $0.05, k$: 8
Crossover scheme	Per-genome component, p: 0.5
Mutation scheme	Per-genome component, p: 0.1

3.4.4 Fitness Function

The fitness function was the same as in Section 3.3.4, but operating on samples recorded for 8ms, from the 1st to the 9th ms of a total electrode activation time of 10ms. This change in the fitness function was made primarily to decrease the time taken to perform a single fitness evaluation, a necessity to be able to attempt to evolve all 256 Elementary Cellular Automata within the time frame of this thesis.

3.4.5 Results

Of the 256 rule evolution attempts, 42 rules were successfully found in the material. The average number of generations before a successful rule was evolved in the cases that found a solution was ~23.57. Table 3.6 shows the results of the evolutionary runs for each of the 256 rules.

Table 3.6: The results of each of the 256 evolutionary runs. Grey rows signify that the rule in that row was successfully evolved. *Binary* shows a binary representation of the rule transition table. *Gen.* is the number of generations simulated for each run. *Evo.* shows a small fitness graph for each run illustrating the best fitness for each generation along the x-axis.

Rule	**Binary**	λ	λ'	**Class**	**Gen.**	**Evo.**	**Best fitness**
Rule$_0$	□□□□□□□□	0.875	0.125	Class I	1		1.2024
Rule$_1$	□□□□□□□■	1	0	Class II	1		1.0203
Rule$_2$	□□□□□□■□	0.875	0.125	Class II	7		1.0800
Rule$_3$	□□□□□□■■	1	0	Class II	24		1.0874
Rule$_4$	□□□□□■□□	0.75	0.25	Class II	1		1.0047
Rule$_5$	□□□□□■□■	0.875	0.125	Class II	100		0.6235
Rule$_6$	□□□□□■■□	0.875	0.125	Class II	100		0.6140
Rule$_7$	□□□□□■■■	1	0	Class II	16		1.0278
Rule$_8$	□□□□■□□□	0.625	0.375	Class I	100		0.5740
Rule$_9$	□□□□■□□■	0.75	0.25	Class II	100		0.5821

Rule	Binary	λ	λ'	Class	Gen.	Evo.	Best fitness
Rule$_{10}$	□□□□■□■□	0.75	0.25	Class II	100		0.5649
Rule$_{11}$	□□□□■□■■	0.875	0.125	Class II	100		0.5331
Rule$_{12}$	□□□□■■□□	0.75	0.25	Class II	100		0.5648
Rule$_{13}$	□□□□■■□■	0.875	0.125	Class II	100		0.5381
Rule$_{14}$	□□□□■■■□	0.875	0.125	Class II	100		0.5640
Rule$_{15}$	□□□□■■■■	1	0	Class II	7		1.0247
Rule$_{16}$	□□□■□□□□	0.5	0.5	Class II	11		1.0728
Rule$_{17}$	□□□■□□□■	0.625	0.375	Class II	14		1.0759
Rule$_{18}$	□□□■□□■□	0.625	0.375	Class III	100		0.5932
Rule$_{19}$	□□□■□□■■	0.75	0.25	Class II	2		1.0066
Rule$_{20}$	□□□■□■□□	0.625	0.375	Class II	15		1.0043
Rule$_{21}$	□□□■□■□■	0.75	0.25	Class II	1		1.0063
Rule$_{22}$	□□□■□■■□	0.75	0.25	Class III	100		0.5924
Rule$_{23}$	□□□■□■■■	0.875	0.125	Class II	1		1.0162
Rule$_{24}$	□□□■■□□□	0.625	0.375	Class II	100		0.5343
Rule$_{25}$	□□□■■□□■	0.75	0.25	Class II	100		0.5512
Rule$_{26}$	□□□■■□■□	0.75	0.25	Class II	100		0.5317
Rule$_{27}$	□□□■■□■■	0.875	0.125	Class II	100		0.5380
Rule$_{28}$	□□□■■■□□	0.75	0.25	Class II	100		0.5329
Rule$_{29}$	□□□■■■□■	0.875	0.125	Class II	100		0.5311
Rule$_{30}$	□□□■■■■□	0.875	0.125	Class III	100		0.5399
Rule$_{31}$	□□□■■■■■	1	0	Class II	4		1.0031
Rule$_{32}$	□□■□□□□□	0.375	0.375	Class I	100		0.5996
Rule$_{33}$	□□■□□□□■	0.5	0.5	Class II	100		0.5624
Rule$_{34}$	□□■□□□■□	0.5	0.5	Class II	100		0.5351
Rule$_{35}$	□□■□□□■■	0.625	0.375	Class II	100		0.5559
Rule$_{36}$	□□■□□■□□	0.5	0.5	Class II	100		0.5340
Rule$_{37}$	□□■□□■□■	0.625	0.375	Class II	100		0.5493
Rule$_{38}$	□□■□□■■□	0.625	0.375	Class II	100		0.5334
Rule$_{39}$	□□■□□■■■	0.75	0.25	Class II	100		0.5420
Rule$_{40}$	□□■□■□□□	0.5	0.5	Class I	100		0.4955
Rule$_{41}$	□□■□■□□■	0.625	0.375	Class II	100		0.4959
Rule$_{42}$	□□■□■□■□	0.625	0.375	Class II	100		0.4975
Rule$_{43}$	□□■□■□■■	0.75	0.25	Class II	100		0.4984
Rule$_{44}$	□□■□■■□□	0.625	0.375	Class II	100		0.4972
Rule$_{45}$	□□■□■■□■	0.75	0.25	Class III	100		0.5025
Rule$_{46}$	□□■□■■■□	0.75	0.25	Class II	100		0.4987
Rule$_{47}$	□□■□■■■■	0.875	0.125	Class II	27		1.0017
Rule$_{48}$	□□■■□□□□	0.5	0.5	Class II	100		0.5847
Rule$_{49}$	□□■■□□□■	0.625	0.375	Class II	77		1.0529
Rule$_{50}$	□□■■□□■□	0.625	0.375	Class II	100		0.5703
Rule$_{51}$	□□■■□□■■	0.75	0.25	Class II	18		1.0244
Rule$_{52}$	□□■■□■□□	0.625	0.375	Class II	100		0.5306
Rule$_{53}$	□□■■□■□■	0.75	0.25	Class II	100		0.5524

Rule	**Binary**	λ	λ'	**Class**	**Gen.**	**Evo.**	**Best fitness**
Rule$_{54}$	□□■■□■■□	0.75	0.25	Class IV	100		0.5474
Rule$_{55}$	□□■■□■■■	0.875	0.125	Class II	4		1.0037
Rule$_{56}$	□□■■■□□□	0.625	0.375	Class II	100		0.4975
Rule$_{57}$	□□■■■□□■	0.75	0.25	Class II	100		0.5005
Rule$_{58}$	□□■■■□■□	0.75	0.25	Class II	100		0.4969
Rule$_{59}$	□□■■■□■■	0.875	0.125	Class II	100		0.5068
Rule$_{60}$	□□■■■■□□	0.75	0.25	Class III	100		0.4997
Rule$_{61}$	□□■■■■□■	0.875	0.125	Class II	100		0.5017
Rule$_{62}$	□□■■■■■□	0.875	0.125	Class II	100		0.5011
Rule$_{63}$	□□■■■■■■	1	0	Class II	12		1.0084
Rule$_{64}$	□■□□□□□□	0.25	0.25	Class I	88		1.0620
Rule$_{65}$	□■□□□□□■	0.375	0.375	Class II	100		0.5575
Rule$_{66}$	□■□□□□■□	0.375	0.375	Class II	100		0.5296
Rule$_{67}$	□■□□□□■■	0.5	0.5	Class II	100		0.5294
Rule$_{68}$	□■□□□■□□	0.375	0.375	Class II	88		1.0597
Rule$_{69}$	□■□□□■□■	0.5	0.5	Class II	100		0.5339
Rule$_{70}$	□■□□□■■□	0.5	0.5	Class II	100		0.5302
Rule$_{71}$	□■□□□■■■	0.625	0.375	Class II	100		0.5500
Rule$_{72}$	□■□□■□□□	0.375	0.375	Class II	100		0.4988
Rule$_{73}$	□■□□■□□■	0.5	0.5	Class II	100		0.4981
Rule$_{74}$	□■□□■□■□	0.5	0.5	Class II	100		0.4964
Rule$_{75}$	□■□□■□■■	0.625	0.375	Class III	100		0.5008
Rule$_{76}$	□■□□■■□□	0.5	0.5	Class II	100		0.4966
Rule$_{77}$	□■□□■■□■	0.625	0.375	Class II	56		1.0044
Rule$_{78}$	□■□□■■■□	0.625	0.375	Class II	100		0.4983
Rule$_{79}$	□■□□■■■■	0.75	0.25	Class II	100		0.5060
Rule$_{80}$	□■□■□□□□	0.375	0.375	Class II	100		0.5398
Rule$_{81}$	□■□■□□□■	0.5	0.5	Class II	100		0.5338
Rule$_{82}$	□■□■□□■□	0.5	0.5	Class II	100		0.5526
Rule$_{83}$	□■□■□□■■	0.625	0.375	Class II	100		0.5493
Rule$_{84}$	□■□■□■□□	0.5	0.5	Class II	100		0.5343
Rule$_{85}$	□■□■□■□■	0.625	0.375	Class II	10		1.0176
Rule$_{86}$	□■□■□■■□	0.625	0.375	Class III	100		0.5509
Rule$_{87}$	□■□■□■■■	0.75	0.25	Class II	2		1.0045
Rule$_{88}$	□■□■■□□□	0.5	0.5	Class II	100		0.4959
Rule$_{89}$	□■□■■□□■	0.625	0.375	Class III	100		0.4984
Rule$_{90}$	□■□■■□■□	0.625	0.375	Class III	100		0.4970
Rule$_{91}$	□■□■■□■■	0.75	0.25	Class II	100		0.5016
Rule$_{92}$	□■□■■■□□	0.625	0.375	Class II	100		0.4979
Rule$_{93}$	□■□■■■□■	0.75	0.25	Class II	17		1.0030
Rule$_{94}$	□■□■■■■□	0.75	0.25	Class II	100		0.5011
Rule$_{95}$	□■□■■■■■	0.875	0.125	Class II	100		0.5183
Rule$_{96}$	□■■□□□□□	0.375	0.375	Class I	100		0.5327
Rule$_{97}$	□■■□□□□■	0.5	0.5	Class II	100		0.5296

Rule	Binary	λ	λ'	Class	Gen.	Evo.	Best fitness
Rule$_{98}$	□■■□□□■□	0.5	0.5	Class II	100		0.5476
Rule$_{99}$	□■■□□□■■	0.625	0.375	Class II	100		0.5465
Rule$_{100}$	□■■□□■□□	0.5	0.5	Class II	100		0.5330
Rule$_{101}$	□■■□□■□■	0.625	0.375	Class III	100		0.5491
Rule$_{102}$	□■■□□■■□	0.625	0.375	Class III	100		0.5475
Rule$_{103}$	□■■□□■■■	0.75	0.25	Class II	100		0.5200
Rule$_{104}$	□■■□■□□□	0.5	0.5	Class II	100		0.4973
Rule$_{105}$	□■■□■□□■	0.625	0.375	Class II	100		0.4968
Rule$_{106}$	□■■□■□■□	0.625	0.375	Class IV	100		0.4947
Rule$_{107}$	□■■□■□■■	0.75	0.25	Class II	100		0.5012
Rule$_{108}$	□■■□■■□□	0.625	0.375	Class II	100		0.4997
Rule$_{109}$	□■■□■■□■	0.75	0.25	Class II	100		0.4988
Rule$_{110}$	□■■□■■■□	0.75	0.25	Class IV	100		0.4979
Rule$_{111}$	□■■□■■■■	0.875	0.125	Class II	100		0.5116
Rule$_{112}$	□■■■□□□□	0.5	0.5	Class II	65		1.0010
Rule$_{113}$	□■■■□□□■	0.625	0.375	Class II	100		0.5288
Rule$_{114}$	□■■■□□■□	0.625	0.375	Class II	100		0.5473
Rule$_{115}$	□■■■□□■■	0.75	0.25	Class II	100		0.5465
Rule$_{116}$	□■■■□■□□	0.625	0.375	Class II	49		1.0270
Rule$_{117}$	□■■■□■□■	0.75	0.25	Class II	100		0.5514
Rule$_{118}$	□■■■□■■□	0.75	0.25	Class II	100		0.5343
Rule$_{119}$	□■■■□■■■	0.875	0.125	Class II	1		1.0084
Rule$_{120}$	□■■■■□□□	0.625	0.375	Class IV	100		0.4966
Rule$_{121}$	□■■■■□□■	0.75	0.25	Class II	100		0.5022
Rule$_{122}$	□■■■■□■□	0.75	0.25	Class III	100		0.4968
Rule$_{123}$	□■■■■□■■	0.875	0.125	Class II	100		0.5124
Rule$_{124}$	□■■■■■□□	0.75	0.25	Class IV	100		0.5001
Rule$_{125}$	□■■■■■□■	0.875	0.125	Class III	100		0.5099
Rule$_{126}$	□■■■■■■□	0.875	0.125	Class III	100		0.4997
Rule$_{127}$	□■■■■■■■	1	0	Class III	2		1.0128
Rule$_{128}$	■□□□□□□□	0.125	0.125	Class I	100		0.5406
Rule$_{129}$	■□□□□□□■	0.25	0.25	Class III	100		0.5508
Rule$_{130}$	■□□□□□■□	0.25	0.25	Class II	100		0.5540
Rule$_{131}$	■□□□□□■■	0.375	0.375	Class II	100		0.5347
Rule$_{132}$	■□□□□■□□	0.25	0.25	Class II	100		0.5566
Rule$_{133}$	■□□□□■□■	0.375	0.375	Class II	100		0.5243
Rule$_{134}$	■□□□□■■□	0.375	0.375	Class II	100		0.5496
Rule$_{135}$	■□□□□■■■	0.5	0.5	Class III	100		0.5462
Rule$_{136}$	■□□□■□□□	0.25	0.25	Class I	100		0.5205
Rule$_{137}$	■□□□■□□■	0.375	0.375	Class IV	100		0.5256
Rule$_{138}$	■□□□■□■□	0.375	0.375	Class II	100		0.5252
Rule$_{139}$	■□□□■□■■	0.5	0.5	Class II	100		0.5451
Rule$_{140}$	■□□□■■□□	0.375	0.375	Class II	100		0.5447
Rule$_{141}$	■□□□■■□■	0.5	0.5	Class II	100		0.5265

Rule	Binary	λ	λ'	Class	Gen.	Evo.	Best fitness
Rule$_{142}$	■□□□■■■□	0.5	0.5	Class II	100		0.5445
Rule$_{143}$	■□□□■■■■	0.625	0.375	Class II	67		1.0329
Rule$_{144}$	■□□■□□□□	0.25	0.25	Class II	100		0.5420
Rule$_{145}$	■□□■□□□■	0.375	0.375	Class II	100		0.5246
Rule$_{146}$	■□□■□□■□	0.375	0.375	Class III	100		0.5580
Rule$_{147}$	■□□■□□■■	0.5	0.5	Class IV	100		0.5315
Rule$_{148}$	■□□■□■□□	0.375	0.375	Class II	100		0.5555
Rule$_{149}$	■□□■□■□■	0.5	0.5	Class III	100		0.5475
Rule$_{150}$	■□□■□■■□	0.5	0.5	Class III	100		0.5457
Rule$_{151}$	■□□■□■■■	0.625	0.375	Class III	21		1.0389
Rule$_{152}$	■□□■■□□□	0.375	0.375	Class II	100		0.5298
Rule$_{153}$	■□□■■□□■	0.5	0.5	Class III	100		0.5502
Rule$_{154}$	■□□■■□■□	0.5	0.5	Class II	100		0.5474
Rule$_{155}$	■□□■■□■■	0.625	0.375	Class II	100		0.5259
Rule$_{156}$	■□□■■■□□	0.5	0.5	Class II	49		1.0040
Rule$_{157}$	■□□■■■□■	0.625	0.375	Class II	27		1.0033
Rule$_{158}$	■□□■■■■□	0.625	0.375	Class II	100		0.5523
Rule$_{159}$	■□□■■■■■	0.75	0.25	Class II	5		1.0198
Rule$_{160}$	■□■□□□□□	0.25	0.25	Class I	100		0.4979
Rule$_{161}$	■□■□□□□■	0.375	0.375	Class III	100		0.4980
Rule$_{162}$	■□■□□□■□	0.375	0.375	Class II	100		0.4954
Rule$_{163}$	■□■□□□■■	0.5	0.5	Class II	100		0.4961
Rule$_{164}$	■□■□□■□□	0.375	0.375	Class II	100		0.4976
Rule$_{165}$	■□■□□■□■	0.5	0.5	Class III	100		0.5003
Rule$_{166}$	■□■□□■■□	0.5	0.5	Class II	100		0.4979
Rule$_{167}$	■□■□□■■■	0.625	0.375	Class II	100		0.5014
Rule$_{168}$	■□■□■□□□	0.375	0.375	Class I	100		0.4984
Rule$_{169}$	■□■□■□□■	0.5	0.5	Class IV	100		0.4964
Rule$_{170}$	■□■□■□■□	0.5	0.5	Class II	100		0.4948
Rule$_{171}$	■□■□■□■■	0.625	0.375	Class II	100		0.5014
Rule$_{172}$	■□■□■■□□	0.5	0.5	Class II	100		0.4984
Rule$_{173}$	■□■□■■□■	0.625	0.375	Class II	100		0.4967
Rule$_{174}$	■□■□■■■□	0.625	0.375	Class II	100		0.4975
Rule$_{175}$	■□■□■■■■	0.75	0.25	Class II	100		0.5126
Rule$_{176}$	■□■■□□□□	0.375	0.375	Class II	100		0.4981
Rule$_{177}$	■□■■□□□■	0.5	0.5	Class II	100		0.5000
Rule$_{178}$	■□■■□□■□	0.5	0.5	Class II	100		0.4996
Rule$_{179}$	■□■■□□■■	0.625	0.375	Class II	100		0.5038
Rule$_{180}$	■□■■□■□□	0.5	0.5	Class II	100		0.4991
Rule$_{181}$	■□■■□■□■	0.625	0.375	Class II	100		0.4994
Rule$_{182}$	■□■■□■■□	0.625	0.375	Class III	100		0.4986
Rule$_{183}$	■□■■□■■■	0.75	0.25	Class III	100		0.5185
Rule$_{184}$	■□■■■□□□	0.5	0.5	Class II	100		0.4980
Rule$_{185}$	■□■■■□□■	0.625	0.375	Class II	100		0.4995

Rule	Binary	λ	λ'	Class	Gen.	Evo.	Best fitness
Rule$_{186}$	■□■■■□■□	0.625	0.375	Class II	100		0.4991
Rule$_{187}$	■□■■■□■■	0.75	0.25	Class II	100		0.4998
Rule$_{188}$	■□■■■■□□	0.625	0.375	Class II	100		0.5021
Rule$_{189}$	■□■■■■□■	0.75	0.25	Class II	100		0.5017
Rule$_{190}$	■□■■■■■□	0.75	0.25	Class II	35		1.0042
Rule$_{191}$	■□■■■■■■	0.875	0.125	Class II	45		1.0214
Rule$_{192}$	■■□□□□□□	0.25	0.25	Class I	100		0.4982
Rule$_{193}$	■■□□□□□■	0.375	0.375	Class IV	100		0.4982
Rule$_{194}$	■■□□□□■□	0.375	0.375	Class I	100		0.4957
Rule$_{195}$	■■□□□□■■	0.5	0.5	Class III	100		0.4991
Rule$_{196}$	■■□□□■□□	0.375	0.375	Class II	100		0.4984
Rule$_{197}$	■■□□□■□■	0.5	0.5	Class II	100		0.5038
Rule$_{198}$	■■□□□■■□	0.5	0.5	Class II	100		0.4986
Rule$_{199}$	■■□□□■■■	0.625	0.375	Class II	100		0.5046
Rule$_{200}$	■■□□■□□□	0.375	0.375	Class II	100		0.4962
Rule$_{201}$	■■□□■□□■	0.5	0.5	Class II	100		0.4971
Rule$_{202}$	■■□□■□■□	0.5	0.5	Class II	100		0.4975
Rule$_{203}$	■■□□■□■■	0.625	0.375	Class II	100		0.4970
Rule$_{204}$	■■□□■■□□	0.5	0.5	Class II	100		0.4969
Rule$_{205}$	■■□□■■□■	0.625	0.375	Class II	100		0.5051
Rule$_{206}$	■■□□■■■□	0.625	0.375	Class II	51		1.0035
Rule$_{207}$	■■□□■■■■	0.75	0.25	Class II	21		1.0123
Rule$_{208}$	■■□■□□□□	0.375	0.375	Class II	100		0.4978
Rule$_{209}$	■■□■□□□■	0.5	0.5	Class II	100		0.5000
Rule$_{210}$	■■□■□□■□	0.5	0.5	Class II	100		0.4966
Rule$_{211}$	■■□■□□■■	0.625	0.375	Class II	100		0.5000
Rule$_{212}$	■■□■□■□□	0.5	0.5	Class II	100		0.4994
Rule$_{213}$	■■□■□■□■	0.625	0.375	Class II	100		0.5068
Rule$_{214}$	■■□■□■■□	0.625	0.375	Class II	100		0.5004
Rule$_{215}$	■■□■□■■■	0.75	0.25	Class II	100		0.5187
Rule$_{216}$	■■□■■□□□	0.5	0.5	Class II	100		0.4975
Rule$_{217}$	■■□■■□□■	0.625	0.375	Class II	100		0.4986
Rule$_{218}$	■■□■■□■□	0.625	0.375	Class II	100		0.4936
Rule$_{219}$	■■□■■□■■	0.75	0.25	Class II	100		0.5009
Rule$_{220}$	■■□■■■□□	0.625	0.375	Class II	100		0.4970
Rule$_{221}$	■■□■■■□■	0.75	0.25	Class II	100		0.5100
Rule$_{222}$	■■□■■■■□	0.75	0.25	Class II	100		0.5002
Rule$_{223}$	■■□■■■■■	0.875	0.125	Class II	51		1.0128
Rule$_{224}$	■■■□□□□□	0.375	0.375	Class I	100		0.4979
Rule$_{225}$	■■■□□□□■	0.5	0.5	Class IV	100		0.4973
Rule$_{226}$	■■■□□□■□	0.5	0.5	Class II	100		0.4969
Rule$_{227}$	■■■□□□■■	0.625	0.375	Class II	100		0.5004
Rule$_{228}$	■■■□□■□□	0.5	0.5	Class II	100		0.4983
Rule$_{229}$	■■■□□■□■	0.625	0.375	Class II	100		0.4990

Rule	**Binary**	λ	λ'	**Class**	**Gen.**	**Evo.**	**Best fitness**
Rule$_{230}$	■■■□□■■□	0.625	0.375	Class II	100		0.4961
Rule$_{231}$	■■■□□■■■	0.75	0.25	Class II	100		0.4981
Rule$_{232}$	■■■□■□□□	0.5	0.5	Class II	100		0.4966
Rule$_{233}$	■■■□■□□■	0.625	0.375	Class II	100		0.4959
Rule$_{234}$	■■■□■□■□	0.625	0.375	Class I	100		0.4932
Rule$_{235}$	■■■□■□■■	0.75	0.25	Class I	100		0.4997
Rule$_{236}$	■■■□■■□□	0.625	0.375	Class II	100		0.4995
Rule$_{237}$	■■■□■■□■	0.75	0.25	Class II	100		0.5007
Rule$_{238}$	■■■□■■■□	0.75	0.25	Class I	100		0.4988
Rule$_{239}$	■■■□■■■■	0.875	0.125	Class I	33		1.0123
Rule$_{240}$	■■■■□□□□	0.5	0.5	Class II	100		0.4990
Rule$_{241}$	■■■■□□□■	0.625	0.375	Class II	100		0.4971
Rule$_{242}$	■■■■□□■□	0.625	0.375	Class II	100		0.4965
Rule$_{243}$	■■■■□□■■	0.75	0.25	Class II	100		0.5082
Rule$_{244}$	■■■■□■□□	0.625	0.375	Class II	100		0.4977
Rule$_{245}$	■■■■□■□■	0.75	0.25	Class II	100		0.5087
Rule$_{246}$	■■■■□■■□	0.75	0.25	Class II	100		0.4992
Rule$_{247}$	■■■■□■■■	0.875	0.125	Class II	5		1.0080
Rule$_{248}$	■■■■■□□□	0.625	0.375	Class I	100		0.4956
Rule$_{249}$	■■■■■□□■	0.75	0.25	Class I	100		0.5040
Rule$_{250}$	■■■■■□■□	0.75	0.25	Class I	100		0.4988
Rule$_{251}$	■■■■■□■■	0.875	0.125	Class I	100		0.5083
Rule$_{252}$	■■■■■■□□	0.75	0.25	Class I	100		0.4959
Rule$_{253}$	■■■■■■□■	0.875	0.125	Class I	100		0.5114
Rule$_{254}$	■■■■■■■□	0.875	0.125	Class I	100		0.4990
Rule$_{255}$	■■■■■■■■	1	0	Class I	1		1.0192

Chapter 4

Analysis

The experiments show that evolved computation in-materio is indeed possible in single-walled carbon nanotube and polymer composite meshes, as a multitude of different linear and non-linear binary logic gates and Elementary Cellular Automata transition tables have been successfully evolved in the material. Using these results as a basis, an attempt can be made to reason about the computational properties of the underlying material itself.

4.1 Complexity Ceiling of the Material

From a theoretical standpoint, computation is a purely abstract mathematical concept. In abstract, a computation can be described as a series of transitions between a number of states caused by some input.

Defining a concrete physical device as a piece of physical material that can be in any one of a finite number of states at any given time, and which can transition from one state to another as a reaction to external excitations, creating a physical implementation of a device that can perform computation in the abstract then becomes an exercise in defining a mapping from the physical states of the material to the states in the abstract computation such that the state transitions in the physical domain result in meaningful state transitions in the abstract domain. Multiple physical states can map to the same abstract state, but a physical state may not map to multiple abstract states. A direct consequence of this is that since there are only a finite number of states a physical device can be in, assuming discrete physics, there is a limit to which computations may be implemented in a given physical device. A physical device capable of being in N different states can implement a computation with at most N states. Any computation with more than N states cannot be implemented in that physical device, as they will by the pigeonhole principle[1] violate the limitation that a physical state may not map to multiple abstract states.

[1]The pigeonhole principle states that if n items are put into m containers, and $n > m$, then at least one container must contain more than one item.

Practical implementations of abstract computations in the physical domain today use a large number of physical states for each logical state.

As an example, consider the MOSFET transistor NAND gate, a physical computational device. It is a physical implementation of the logical NAND gate, an abstract computational device. A MOSFET transistor can be in an enormous amount of different physical states as the electrical charges in different parts of the device vary. Yet, the abstract computation device it implements only has two states, which is also the output of the device: logical 0 and logical 1.

The larger the disparity between the number of physical states used and the number of abstract states used in a mapping, the less efficient the computer implementation is in terms of utilizing the computational potential of the physical substrate. The relationship between the number of physical state and the number abstract states

$$\eta = \frac{|S_A|}{|S_P|} \tag{4.1}$$

can be defined as in (4.1), where S_A is the set of all possible states in the abstract computation device and S_P is the set of all possible states in the physical computation device. Then, a physical implementation of an abstract computation device with a higher η is more material-efficient than one with a lower η, with $\eta = 1$ being the absolute theoretical maximum for this measure. The usefulness of η in measuring implementation efficiency presupposes a discrete physical environment, as the converse implies that all materials can be in an infinite number of different states. Currently, it is not known whether or not the physical world is discrete.

Armed with the assumption that a physical material sample has a ceiling for the computational complexity of devices that can be implemented in them (i.e. has a maximum number of states it can be in and transition between), together with the intuition that it is easier to find a physical-to-abstract state mapping with a larger η than one with a smaller η, the results of the Elementary Cellular Automata evolution experiments explained in Chapter 3 can be used as a proxy measurement of the computational complexity ceiling of the material-under-study in this thesis.

4.2 Evolvability and the λ-Parameter

Recall that the lambda parameter is one of many different proposed schemes of classification of cellular automata. Cellular automata with a λ-parameter close to 0 tend toward a frozen, non-changing structure over time, while cellular automata with a λ-parameter close to 1 tend toward completely chaotic behavior; "complex" behavior lies in-between [25]. Assuming that different materials have different inherent potentials for computation complexity with regards to evolution in-materio, the λ-parameter of different evolved cellular automata in a material can be used as a proxy for measuring the complexity ceiling of that material. Since this metric is a proxy metric, it is limited in scope to the specific methods used for evolution and interpretation of computation.

Care must be taken when using λ-parameter, as it is originally only well-defined for a subset of all cellular automata. A cellular automaton only has a λ-parameter if the non-quiescent state transitions, i.e. state transitions that are not transitions to the quiescent state, are randomly and uniformly distributed over the remaining non-quiescent states [25]. For binary cellular automata there can only be one non-quiescent state, which means that all binary cellular automata strictly speaking have a well-defined λ since non-quiescent state transitions are randomly and uniformly distributed over the only non-quiescent state. However, the lack of choice in non-quiescent states does alter the qualitative behaviors of binary cellular automata at high λ-parameters when compared to the original findings in [25]. High λ-parameter binary cellular automata will tend to frozen structures rather than chaotic behavior. As such, it can be useful to define a binary variant of the λ-parameter, the λ'-parameter, which is like the λ-parameter except that the quiescent state is always the most transitioned-to state. This means that the λ'-parameter is effectively a mirroring of the λ-parameter around $\lambda = 0.5$ as the maximum value.

Looking at the results from the evolution experiments from Chapter 3, and at the All-ECA experiment in particular, some conclusions can be made with regards to the complexity of the material. It seems that Elementary Cellular Automata with extreme λ-parameters, i.e. closer to $\lambda = 0$ and $\lambda = 1$, or closer to $\lambda' = 0$, evolve more easily than Elementary Cellular Automata that have a λ-parameter somewhere in-between. The Elementary Cellular Automata that evolve the least easily in the experimental setup are the ones with the largest λ'-parameters. An overview of evolution difficulty measured as the average number of generations used for evolutionary runs grouped by λ'-parameter can be seen in Figure 4.2. A different overview of evolution difficulty measured as the average fitness of the best individual of the last generation of each evolutionary run grouped by λ'-parameter can be seen in Figure 4.1. Both measures tell the same story: Elementary Cellular Automata with extreme λ'-parameters evolve more easily than Elementary Cellular Automata with λ'-parameters close to 0.5. These findings are in-line with the intuition that more computationally complex cellular automata should take longer to evolve, if taken together with the idea that the most computationally complex cellular automata appear at the Edge-of-Chaos [25], i.e. at the phase transition between ordered and chaotic behavior. Looking at the location of the Edge of Chaos in Figure 2.8, these results support the notion that for Elementary Cellular Automata the Edge of Chaos lies around λ-parameter values of ~0.5.

4.3 Evolvability and Wolfram Classification

Recall that a different classification scheme for cellular automata is the Wolfram Classification Scheme. Additional insights to the questions around computational complexity in the material might be gleaned from looking at the evolvability of the different Elementary Cellular Automata grouped by Wolfram Classes.

In the Wolfram Classification, the classes are ordered by complexity, so if the hypothesis that less complex Elementary Cellular Automata evolve in-materio more easily than more complex Elementary Cellular Automata, then it is reasonable to expect that evolving a Class I automaton should on average require fewer generations than a Class II automation, a Class II automaton should on average require fewer generations than a Class III automaton, and finally a Class III automaton should require on average fewer generations to evolve than a Class IV automation.

Figure 4.3 shows the distributions of generations simulated in order to evolve an ac-

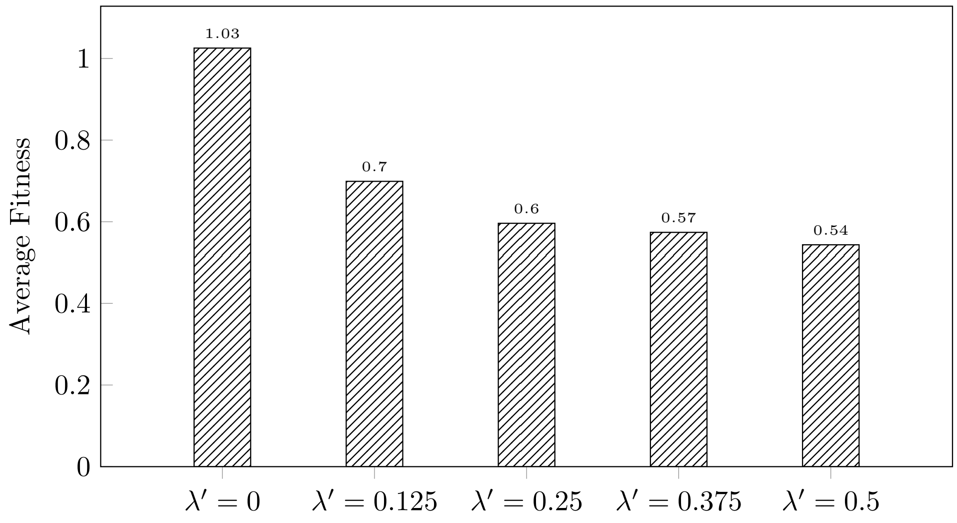


Figure 4.1: The average fitness of the best individual of the last generation of an evolutionary run grouped by λ'-parameter.

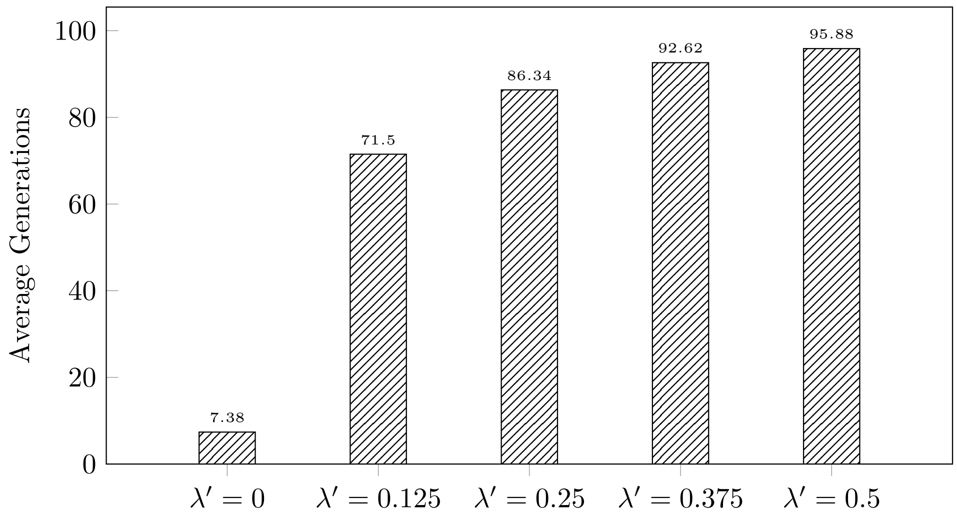


Figure 4.2: The average number of generations simulated per evolutionary run grouped by λ'-parameter.

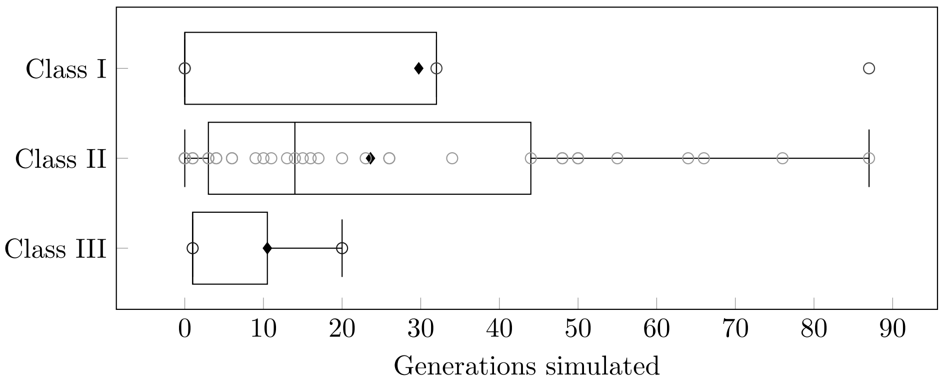


Figure 4.3: Distribution of evolution length as measured by generation count grouped by Wolfram Class. Circles show the number of generations required to evolve a rule. The boxes are Tukey-style boxplots [32], and show median and quartile values as vertical lines and average values as diamonds, with whiskers showing the smallest value larger than (the lower quartile - 1.5 IQR) and the largest value smaller than (the upper quartile + 1.5 IQR), IQR being the difference between the upper quartile and the lower quartile. Class IV is not present as there were no successful Class IV evolution runs in the All-ECA experiment.

ceptable Elementary Cellular Automata for each of the four classes. The distributions do not include non-successfully evolved Elementary Cellular Automata.

Figure 4.4 shows the class distribution of the 42 successfully evolved Elementary Cellular Automata compared to the class distribution of all 256 Elementary Cellular Automata.

Of the 256 Elementary Cellular Automata, 25 (~9.8%) are Class I, 192 (~75.8% are Class II), 27 (~10.5%) are Class III, and 12 (~4.7%) are Class IV. If the opposite of what the hypothesis predicts were true, i.e. that Elementary Cellular Automata are on average equally likely to be successfully evolved in-materio regardless of Wolfram Class, a similar distribution of classes should be present in the set of successfully evolved Elementary Cellular Automata in the All-ECA experiment. Of the successfully evolved Elementary Cellular Automata in the All-ECA experiment, however, ~9.5% are Class I, ~85.7% are Class II, ~4.8% are Class III and 0% are Class IV. This is a very different class distribution than what should be expected if the any Elementary Cellular Automata were equally likely to evolve successfully. Hence, the results from the experiment indicate that there might be a correlation between cellular automata complexity and evolvability in-materio.

4.4 Evolvability and Set Bits in an Elementary Cellular Automaton Rule

As a contrast to looking at correlations between cellular automata complexity and evolvability in-materio, other potential correlators should be looked at as well. Perhaps the evolvability of an Elementary Cellular Automata is not a function of its complexity, but rather simply the numbers of set bits in the Elementary Cellular Automaton Rule number.

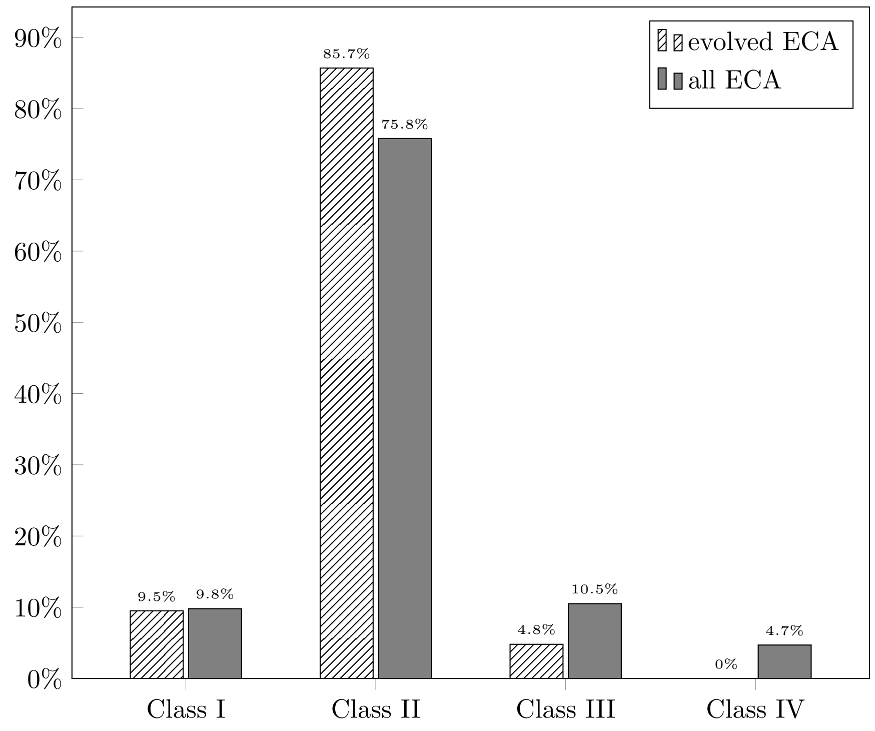


Figure 4.4: Elementary Cellular Automata Grouped by Wolfram Class.

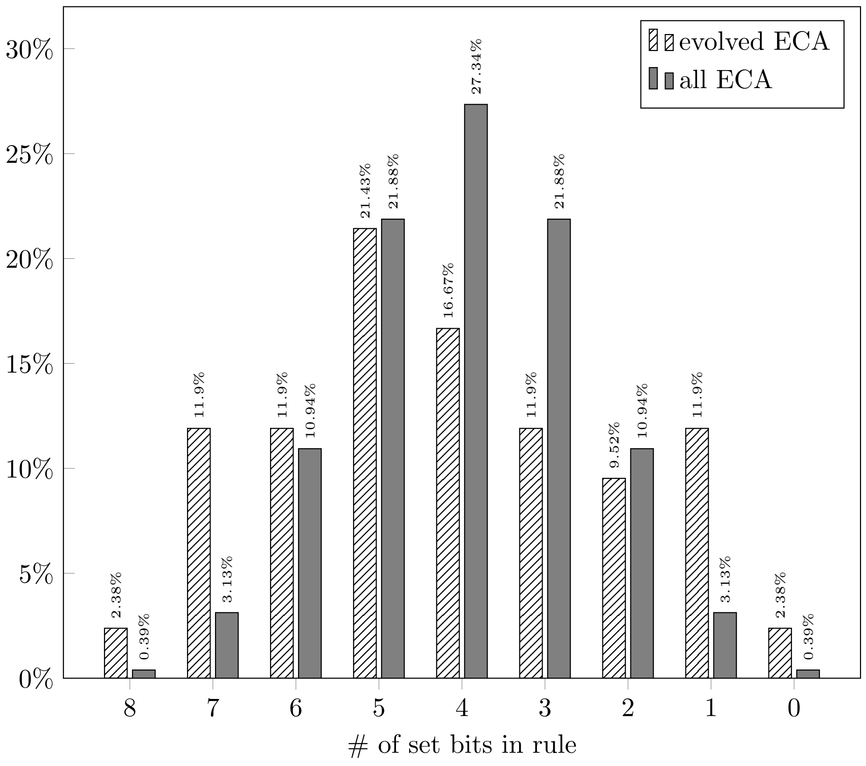


Figure 4.5: Elementary Cellular Automata grouped by set bits in rule.

It is easy to imagine a simple hypothetical material for which this is the case – any material which favors set bits rather than unset bits as output would qualify. An extreme example is a hypothetical material which always outputs set bits regardless of input. Looking at the distribution of the amount of set bits in the 256 Elementary Cellular Automata compared to distribution of the amount of set bits in the 42 successfully evolved Elementary Cellular Automata, solutions with 6-8 bits set are over-represented, and solutions with 2-5 bits set are under-represented. Solutions with 0 or 1 set bits are over-represented again. A comparison of the two distributions can be seen in Figure 4.5.

Counting the number of set bits in any binary cellular automaton, and certainly therefore in the Elementary Cellular Automata, is analogous to calculating the λ-parameter of a rule.

4.5 Sensitivity Analysis

Only a single evolutionary run has been executed for each of the 256 Elementary Cellular Automata in the experiment detailed in Section 3.4. When taken individually, a single run for each of the 256 different rules is not enough to be able to draw meaningful conclusions about a single Elementary Cellular Automaton in-materio, statistically speaking. As an example, consider the evolution of Elementary Cellular Automata Rule$_{54}$ in the Single Elementary Cellular Automaton experiment (Section 3.3) and in the All Elementary Cellular Automata experiment (Section 3.4). The All-ECA experiment was not able to evolve a suitable Rule$_{54}$, yet a suitable solution for that same rule was clearly found in the Single-ECA experiment. Care must therefore be taken not to attribute more importance to the results of this experiment than should be afforded when analyzed from a statistical point-of-view. Looking at the experiment as a whole, then, treating the different runs as repeat experiments over different classifications, is the better approach to extracting a meaningful interpretation of this experiment.

For the purpose of reducing the worst-case evolution time for a single evolutionary run, each run was capped at 100 generations. That is, if a solution was not found after 100 generations, the run would be considered unsuccessful. This generation cap has probably pruned away a couple would-be-successful evolutions had the generation cap been higher, e.g. capped at 1000 generations. Although more Elementary Cellular Automata were successfully evolved after few generations rather than many, as illustrated in Figure 4.3, it seems reasonable to assume that more generations per evolutionary run would ultimately yield more successful evolutions.

Again for the purpose of reducing the evolution time for a single evolutionary run, the fitness evaluation was changed to a $10\,\mathrm{ms}$-based computation rather than a $100\,\mathrm{ms}$-based computation. This change could also impact evolvability of a rule in the All-ECA experiment when compared to the Single-ECA experiment. Ultimately, when considering the maximum frequency ($50\,\mathrm{kHz}$) of oscillation on the input electrodes, the frequency of sampling on the output electrode ($500\,\mathrm{kHz}$), the relative order of magnitude between the two, a model of the material that assumes an electrical stabilization on the order of microseconds or less, and considering the fact that the $100\,\mathrm{ms}$ value was chosen rather arbitrarily in the Single-ECA experiment to begin with, it seems at least intuitively unlikely that a change of execution time from $100\,\mathrm{ms}$ to $10\,\mathrm{ms}$ should greatly impact evolvability of a fit solution. Still, intuition aside, the impact of this change remains an open question.

Further, still motivated by time constraints, the population size in the experiments was reduced from 40 to 20 individuals in each of the adult and child pools. A change in this direction generally increases the number of generations that must be simulated before an acceptable solution is found, and a too small population increases the risk of the evolutionary algorithm getting stuck at local maxima in the fitness space. Still, the decreased population size is still well within the limits of what has been shown to work for evolution in-materio in random single-walled carbon nanotube and polymer meshes. In a number of experiments, desired computation is successfully evolved in-materio using an evolutionary algorithm population size of 5 [10, 36, 37, 38], which is considerably less than the 20+20 population size used in the latter experiments in this thesis.

Even with all these time-saving changes to the largest experiment, performing a single fitness evaluation of a solution candidate in the material still takes on the order of 10 s to compute because of various unavoidable overheads. With the enormous number of fitness evaluations required by these experiments, performing the experiments has taken several months of around-the-clock in-materio computation. Increasing the number of repeat runs of each rule evolution and perhaps also increasing the generation cap for each evolutionary run would improve results in terms of statistical significance, but is unfortunately prohibitively time-consuming, and therefore out of scope for this thesis.

4.6 Material Sample

The same material sample was used for all the experiments. Random single-walled carbon nanotube (SWCNT) and polymer mesh devices, as the name suggests, are randomly constructed. Because of this, different material samples may exhibit vastly different computational behavior. There are many different variables such as nanotube concentration, electrode layout, production methods and more that may improve or decrease the material's aptitude for computation substration. One of the goals for this thesis is to investigate the viability of SWNCT as a material for Evolution-in-Materio, and while the material is shown to support stable complex computation, there are still many facets of the material left to investigate.

An abstract computational device evolved on one material sample cannot be used on a different material sample directly. This limits the commercial potential of SWCNT devices when used for evolution-in-materio, since while they can be efficiently mass-produced [14], each individual physical device needs to have a unique configuration evolved to be useful.

4.7 Stability of Results

The stability of the results is greater than that of previous work [23, 27], and the evolved solutions seem stable enough to be called "stable" solutions in the context of Evolution-in-Materio. Looking closer at the distribution of measurements in Figure 3.8, Figure 3.16 and Figure 3.15, a peculiarity becomes apparent. There is sometimes a small separate clustered group of measurements far away from the median which severely reduce the stability of the otherwise very tight clustering of measurements around the median. This seems strange, and may be caused by some complex intrinsic process within the material itself, but it may also be caused by some experimental error in process, equipment, software or similar. If the latter is the case, the true computational stability of the solutions may very well be much greater than what they are measured to be in the experiments in this thesis. Still, how stable is stable enough? Comparing to the error rate of consumer-grade conventional computers, which, while not published anywhere, seems to be on the order of one quintillion operations per error[2], the computational devices presented in this thesis are anything but stable.

4.8 Environmental Dependence

The experiments model the material as an ideal device that only reacts to electrical signals on the electrodes. In reality, the computational properties and process probably vary based on other external effects such as changes in temperature, light, and other environment variables. No special care was taken to maintain a stable environment – the experiments were run on a desk in a shared computer hardware laboratory in close proximity to noisy computers, a soldering station and multiple different types of lamps and light fixtures, as you might commonly expect to find in a computer hardware laboratory.

The relative stability of the results despite lack of a strictly controlled environment suggest that the material is reasonably invariant to the changing environmental effects of an indoor environment. This is also what one might expect when looking at the material from a material sciences perspective. The demonstrated environmental invariance in the computational substrate corroborates the attractiveness of single-walled carbon nanotube and polymer composite meshes as a computational substrate.

4.9 Speed of Computation

Currently, performing a computation in-materio takes on the order of $10\,\mathrm{ms}$ to complete. This is because the input/output encoding is specified somewhat arbitrarily to last for that length of time. $10\,\mathrm{ms}$ is quite slow compared to even consumer-grade conventional computers, which are easily capable of upwards of hundreds of millions of operations over the same time period. That being said, the results presented in this thesis are a proof of concept, and computation speeds may be improved upon in further work.

4.10 Where Does Computation Take Place?

Does the computation actually take place in-materio? When performing evolution-guided search for computation in a material, the entire input domain and output range of the computational function is known, and a signal encoding and decoding process is performed off-material. This can make it hard to pinpoint exactly where the computation takes place. Certainly it is possible to construct a fitness evaluator and input/output encoding that is so

[2]An estimated 2 billion operations each second every day for 20 years before the silicon microchip wears out.

complex that it can find computation in anything – even random noise. In such a case, the computation is in reality happening outside of the material. How can the origin of computation be measured? It can be helpful to replace the material with different hypothetical materials and imagine what would happen if the same computations were performed using the switched hypothetical materials, but still using the same input and output coding schemes. Considering the following three hypothetical alternative materials, some insight might be gained into the computational complexity of the SWCNT material: 1. a computationally "dead" material that always outputs the same static signal(s); 2. a material that produces "true random noise" on its output(s) regardless of the input; and 3. a material that linearly combines its input(s) and passes it on to its output(s). Does the computation that allegedly happens in the real material also happen when the material is replaced with one these hypothetical materials? For one, the real material certainly out-performs the "dead" material – all of the implemented functions show a range that requires the output voltage to be above or below some static non-changing threshold level depending on the input. Since the expected output depends on the input, and the static threshold crucially does *not* change based on the input, it demonstrably performs more computation than the "dead" material.

Now, in the case of the random material, it is possible that the random output happens to measure on the right side of the threshold level for different inputs by pure chance. However, it will probably not do so very often, statistically speaking. The evolved devices presented in this thesis are all reasonably stable in their output, or at least much more stable than what one can expect from a "true" random material. This hints at an understanding where at least some of the computation happens in the material itself.

In the case of the linearly combining material, linear computation is possible in-materio almost by definition, but computations that are not linearly separable should not be implementable. The evolved XOR gate, and several of the Elementary Cellular Automata in the SWCNT material, however, are not linearly separable functions. Thus, the SWCNT material seems to exhibit computational promise beyond linearly separable functions.

Chapter 5

Conclusion

This thesis has explored the idea of using Evolution-in-Materio to exploit a single-walled carbon nanotube and polymer composite random mesh material for abstracted computation using cellular automata. The goals were to investigate the capacity for computational complexity in the material-under-study, and to reason about the complexity ceiling for a computational substrate for Evolution-in-Materio in a general sense. The experiments presented in this thesis show that reasonably stable linear and non-linear binary logic gates, Sub-Elementary Cellular Automata, and Class I, II, III and IV Elementary Cellular Automata can be successfully evolved in-materio in a random single-walled carbon nanotube and polymer mesh material, and as such demonstrate theoretical viability of the material as a computational substrate. The degree of complexity of different abstract computational devices is correlated with the probability of a successful evolution of a physical implementation of those abstract devices. As such, the degree of success in evolution of Elementary Cellular Automata of different Wolfram Classes and with differing λ-parameters in-materio has been used as a proxy to measure the complexity ceiling for the material-under-study, and, given that the proxy metric is accurate, supports Langton's notions of complexity at the Edge-of-Chaos [25]. Further, a simple theoretical measure η for implementation efficiency of abstract computation devices in physical materials has been introduced.

5.1 Further Work

Evolution-in-materio is a time-consuming approach to designing computational devices. The analysis and conclusions made in this thesis could be strengthened significantly by increasing the number of evolutionary runs made to increase sample sizes, as discussed in Section 4.5. In order to do this efficiently, new implementation schemes could be devised that allow for shorter computation times in-materio for quicker fitness evaluation, which again allows for faster Evolution-in-Materio. This also synergizes well with lifting the efficiency of an evolved device out of the realm of proof-of-concepts and into the realm of situationally useful computational devices.

The current input/output encoding scheme for signals in and out of the material are not chainable without intermediate conversion. This limits the usefulness in creating larger composite computation devices through traditional componentized design. One possible avenue for further work is to search for an input-output compatible chainable representation that allows for feeding output from one device as input to the next.

Energy efficiency is one of the areas in which Evolution-in-Materio could show promise. No work has been done in this thesis to measure energy efficiency of the presented devices. One possible avenue for further work is to measure and compare the relative energy efficiency of different signal encoding schemes coupled with different abstract computational devices implemented in the material-under-study.

In Section 4.7, a peculiarity in stability measurements is discussed. One possible avenue for further work is to investigate this peculiarity to determine the cause, which could result in greater stability.

No special effort has been made to measure the environmental dependence of the material, as discussed in Section 4.8. One possible avenue for further work is to examine how environmental factors such as temperature, light and others affect computation in the material.

The work in this thesis focuses on the evolution of computational devices in-materio. However, this is just the first step in designing complete computing systems that use evolved in-materio designs. One possible avenue for further work is to build a more practical complete computing system leveraging Evolution-in-Materio, be it specialized or universal in terms of computational power. One example of such a possible system would be a hybrid conventional/materio system where a computationally powerful Elementary Cellular Automata, e.g. the computation-universal Rule 110 [11], is implemented in-materio and used to calculate state transitions for a cellular automaton simulation where the state of the simulation is kept track of on a traditional computer for practicality. An illustration of a possible setup for such a simulation can be seen in Figure 5.1[1].

Ultimately, Evolution-in-Materio in the long term promises new possibilities for physical computational devices with extreme properties arising from specialized exploitation of substrates beyond what is possible with traditional design approaches such as extremely high energy efficiency or extremely low latency in real-time systems. It is even possible to envision computing systems where material configurations are evolved “on-the-fly” to be used for a short period of period of time before it is discarded as the requirements of the environment changes, by way of analogy much like a just-in-time compiler from the world of programming language interpreters works.

[1]Desktop computer drawing adapted with permission from sweetclipart.com.

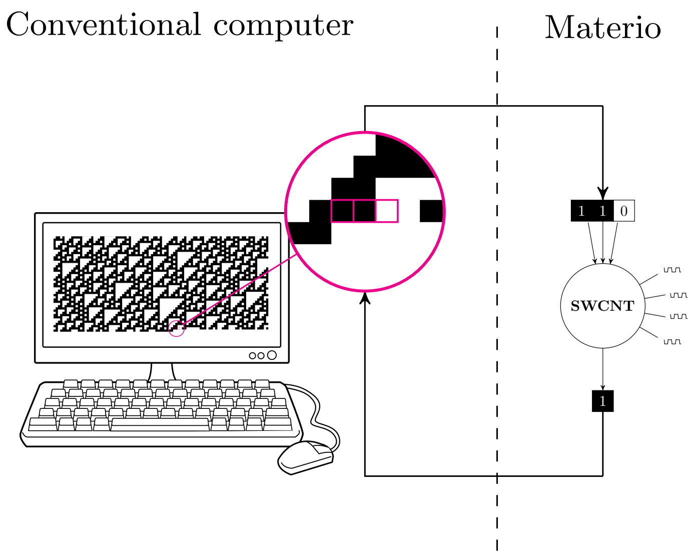


Figure 5.1: A setup for a practical simulation of Rule 110 in a hybrid conventional/materio device. The transitions are computed in-materio, and the state is stored on a conventional computer.

Bibliography

[1] Apache Software Foundation, 2007. Thrift.

[2] atypic, simonharding, 2015. mecobo. [Accessed 18-December-2015].
URL `https://web.archive.org/web/20151218034716/https://github.com/NASCENCE/mecobo/tree/b8c69f95a2252c35dc2104401b0eee0decaca428`

[3] Bandini, S., Mauri, G., Serra, R., 2001. Cellular automata: From a theoretical parallel computational model to its application to complex systems. Parallel Computing 27 (5), 539 – 553, cellular automata: From modeling to applications.

[4] Bessonov, A. A., Kirikova, M. N., Petukhov, D. I., Allen, M., Ryhänen, T., Bailey, M. J. A., Feb 2015. Layered memristive and memcapacitive switches for printable electronics. Nat Mater 14 (2), 199–204, letter.

[5] Bockrath, M., Mar 2006. Carbon nanotubes: The weakest link. Nat Phys 2 (3), 155–156.

[6] Canonical Ltd. and Ubuntu community, 2012. Ubuntu 12.10.

[7] Chanthbouala, A., Garcia, V., Cherifi, R. O., Bouzehouane, K., Fusil, S., Moya, X., Xavier, S., Yamada, H., Deranlot, C., Mathur, N. D., Bibes, M., Barthélémy, A., Grollier, J., Oct 2012. A ferroelectric memristor. Nat Mater 11 (10), 860–864.

[8] Chapman, P., 2004. Life Universal Computer. [Accessed 16-December-2015].
URL `https://web.archive.org/web/20150217021129/http://www.igblan.free-online.co.uk/igblan/ca/`

[9] Charlier, J.-C., Blase, X., Roche, S., May 2007. Electronic and transport properties of nanotubes. Rev. Mod. Phys. 79, 677–732.

[10] Clegg, K., Miller, J., Massey, K., Petty, M., 2014. Travelling Salesman Problem Solved 'in materio' by Evolved Carbon Nanotube Device. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (Eds.), Parallel Problem Solving from Nature –

PPSN XIII. Vol. 8672 of Lecture Notes in Computer Science. Springer International Publishing, pp. 692–701.

[11] Cook, M., 2004. Universality in Elementary Cellular Automata. Complex Systems 15 (1), 1–40.

[12] Culik, II, K., Yu, S., Apr. 1988. Undecidability of CA Classification Schemes. Complex Syst. 2 (2), 177–190.

[13] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., Burger, D., 2011. Dark Silicon and the End of Multicore Scaling. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. ISCA '11. ACM, New York, NY, USA, pp. 365–376.

[14] Gabriel, J., Bradley, K., Collins, P., 2006. Dispersed growth of nanotubes on a substrate. EP Patent App. EP20,030,808,389.

[15] Gale, E., Mayne, R., Adamatzky, A., de Lacy Costello, B., 2014. Drop-coated Titanium Dioxide Memristors. Materials Chemistry and Physics 143 (2), 524–529.

[16] Greene, D., 2009. Completed Universal Computer/Constructor. [Accessed 16-December-2015].
URL `https://web.archive.org/web/20150210043146/http://pentadecathlon.com/lifeNews/2009/08/post.html`

[17] Harding, S., Miller, J., June 2004. Evolution in materio: a tone discriminator in liquid crystal. In: Evolutionary Computation, 2004. CEC2004. Congress on. Vol. 2. pp. 1800–1807 Vol.2.

[18] Harding, S., Miller, J., June 2004. Evolution in materio: initial experiments with liquid crystal. In: Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on. pp. 298–305.

[19] Harding, S., Miller, J., June 2005. Evolution in materio: a real-time robot controller in liquid crystal. In: Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on. pp. 229–238.

[20] Harding, S., Miller, J. F., 2005. Evolution in materio: Evolving logic gates in liquid crystal. In: In Proceedings of the workshop on unconventional computing at ECAL 2005 VIIIth European. p. 12.

[21] Hornby, G. S., Lohn, J. D., Linden, D. S., 2011. Computer-Automated Evolution of an X-Band Antenna for NASA's Space Technology 5 Mission. Evolutionary Computation 19 (1), 1–23.

[22] Hromkovic, J., Oliva, W. M., 2002. Algorithmics for Hard Problems, 2nd Edition. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[23] Kotsialos, A., Massey, M., Qaiser, F., Zeze, D., Pearson, C., Petty, M., September 2014. Logic gate and circuit training on randomly dispersed carbon nanotubes. International journal of unconventional computing. 10 (5-6), 473–497.

[24] Krieger, J., Spitzer, S., 2004. Non-traditional, non-volatile memory based on switching and retention phenomena in polymeric thin films. In: Non-Volatile Memory Technology Symposium, 2004. pp. 121–124.

[25] Langton, C., June 1990. Computation at the Edge of Chaos Phase Transitions and Emergent Computation. Physica D 42, 12–37.

[26] Lu, X., Chen, Z., 2005. Curved Pi-Conjugation, Aromaticity, and the Related Chemistry of Small Fullerenes (¡C60) and Single-Walled Carbon Nanotubes. Chemical Reviews 105 (10), 3643–3696, pMID: 16218563.

[27] Lykkebø, O., Harding, S., Tufte, G., Miller, J., 2014. Mecobo: A Hardware and Software Platform for In Materio Evolution. In: Ibarra, O. H., Kari, L., Kopecki, S. (Eds.), Unconventional Computation and Natural Computation. Vol. 8553 of Lecture Notes in Computer Science. Springer International Publishing, pp. 267–279.

[28] Marcus, M., Akera, A., Spring 1996. Exploring the architecture of an early machine: the historical relevance of the ENIAC machine architecture. Annals of the History of Computing, IEEE 18 (1), 17–24.

[29] Martinez, G. J., Seck-Tuoh-Mora, J. C., Zenil, H., 2012. Wolfram's Classification and Computation in Cellular Automata Classes III and IV. ArXiv e-prints.

[30] Martínez, G. J., Adamatzky, A., McIntosh, H. V., 2006. Phenomenology of glider collisions in cellular automaton rule 54 and associated logical gates. Chaos, Solitons & Fractals 28 (1), 100 – 111.

[31] Massey, K., 2014. Material for NTNU 3. private communication.

[32] Michael Frigge, David C. Hoaglin, B. I., 1989. Some implementations of the boxplot. The American Statistician 43 (1), 50–54.

[33] Miller, J., Downing, K., 2002. Evolution in materio: looking beyond the silicon box. In: Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on. pp. 167–176.

[34] Mitchell, M., 1996. Computation in cellular automata: A selected review. Nonstandard Computation, 385–390.

[35] Mitchell, M., Hraber, P. T., Crutchfield, J. P., 1993. Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations. Complex Systems 7, 89–130.

[36] Mohid, M., Miller, J., Harding, S., Tufte, G., Lykkebo, O., Massey, M., Petty, M., Dec 2014. Evolution-in-materio: A frequency classifier using materials. In: Evolvable Systems (ICES), 2014 IEEE International Conference on. pp. 46–53.

[37] Mohid, M., Miller, J., Harding, S., Tufte, G., Lykkebo, O., Massey, M., Petty, M., Sept 2014. Evolution-in-materio: Solving function optimization problems using materials. In: Computational Intelligence (UKCI), 2014 14th UK Workshop on. pp. 1–8.

[38] Mohid, M., Miller, J., Harding, S., Tufte, G., Lykkebø, O., Massey, M., Petty, M., 2014. Evolution-In-Materio: Solving Machine Learning Classification Problems Using Materials. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (Eds.), Parallel Problem Solving from Nature – PPSN XIII. Vol. 8672 of Lecture Notes in Computer Science. Springer International Publishing, pp. 721–730.

[39] Naitoh, Y., Horikawa, M., Shimizu, T., 2007. New Nonvolatile Memory Effect Showing Reproducible Large Resistance Ratio Employing Nano-gap Gold Junction. In: Symposium I – Materials and Processes for Nonvolatile Memories II. Vol. 997 of MRS Proceedings.

[40] NAnoSCale Engineering for Novel Computation using Evolution, 2015. NASCENCE Project Website. [Accessed 16-December-2015].
URL `https://web.archive.org/web/20151025081433/http://nascence.no/`

[41] NAnoSCale Engineering for Novel Computation using Evolution, 2015. NASCENCE Project Website: Materials. [Accessed 16-December-2015].
URL `https://web.archive.org/web/20151103012714/http://nascence.no/index.php/materials`

[42] Pask, G., 1958. Physical analogues to the growth of a concept. Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory., 879–922.

[43] Patterson, D. A., Hennessy, J. L., 2008. Computer Organization and Design, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design), 4th Edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[44] Python Software Foundation and Guido van Rossum, 2010. Python 2.7.

[45] Rendell, P., 2006. This Is a Turing Machine Implemented in Conway's Game of Life. [Accessed 16-December-2015].
URL `https://web.archive.org/web/20150303022657/http://rendell-attic.org/gol/tm.htm`

[46] Rosen, K. H., 2007. Discrete Mathematics and Its Applications, 6th Edition. McGraw-Hill Higher Education.

[47] Snow, E. S., Novak, J. P., Campbell, P. M., Park, D., 2003. Random networks of carbon nanotubes as an electronic material. Applied Physics Letters 82 (13), 2145–2147.

[48] Stepney, S., 2008. The Neglected Pillar of Material Computation. Physica d-Nonlinear phenomena 237 (9), 1157–1164.

[49] Tang, Z. K., Zhang, L., Wang, N., Zhang, X. X., Wen, G. H., Li, G. D., Wang, J. N., Chan, C. T., Sheng, P., 2001. Superconductivity in 4 Angstrom Single-Walled Carbon Nanotubes. Science 292 (5526), 2462–2465.

[50] Thompson, A., 1997. An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics. In: Higuchi, T., Iwata, M., Weixin, L. (Eds.), Proc. 1st Int. Conf. on Evolvable Systems (ICES'96). Springer-Verlag, Berlin, pp. 390–405.

[51] Tiarks, D., Baur, S., Schneider, K., Dürr, S., Rempe, G., Jul 2014. Single-Photon Transistor Using a Förster Resonance. Phys. Rev. Lett. 113, 053602.

[52] Ulstrup, S., Johannsen, J. C., Cilento, F., Miwa, J. A., Crepaldi, A., Zacchigna, M., Cacho, C., Chapman, R., Springate, E., Mammadov, S., Fromm, F., Raidel, C., Seyller, T., Parmigiani, F., Grioni, M., King, P. D. C., Hofmann, P., Jun 2014. Ultrafast Dynamics of Massive Dirac Fermions in Bilayer Graphene. Phys. Rev. Lett. 112, 257401.

[53] Umeo, H., 2012. Firing Squad Synchronization Problem in Cellular Automata. In: Meyers, R. A. (Ed.), Computational Complexity. Springer New York, pp. 1094–1130.

[54] Wolfram, S., Jul 1983. Statistical mechanics of cellular automata. Rev. Mod. Phys. 55, 601–644.

[55] Wolfram, S., 2002. A New Kind of Science. Wolfram Media.

[56] Ye, P. D., Wilk, G. D., Yang, B., Kwo, J., Chu, S. N. G., Nakahara, S., Gossmann, H.-J. L., Mannaerts, J. P., Hong, M., Ng, K. K., Bude, J., 2003. GaAs metal–oxide–semiconductor field-effect transistor with nanometer-thin dielectric grown by atomic layer deposition. Applied Physics Letters 83 (1), 180–182.

[57] Zhirnov, V., Cavin, R., Hutchby, J., Bourianoff, G., Nov 2003. Limits to binary logic switch scaling - a gedanken model. Proceedings of the IEEE 91 (11), 1934–1939.