Design and Implementation

of a

Framework for the Interconnection of Cellular Automata

in

Software and Hardware

by

Brandon James DeHart

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Brandon James DeHart

Abstract

There has been a move recently in academia, industry, and the consumer space towards the use of unsupervised parallel computation and distributed networks (i.e., networks of computing elements working together to achieve a global outcome with only local knowledge). To fully understand the types of problems that these systems are applied to regularly, a representative member of this group of unsupervised parallel and distributed systems is needed to allow the development of generalizable results. Although not the only potential candidate, the field of cellular automata is an excellent choice for analyzing how these systems work as it is one of the simplest members of this group in terms of design specification. The current ability of the field of cellular automata to represent the realm of unsupervised parallel and distributed systems is limited to only a subset of the possible systems, which leads to the main goal of this work of finding a method of allowing cellular automata to represent a much larger range of systems.

To achieve this goal, a conceptual framework has been developed that allows the definition of interconnected systems of cellular automata that can represent most, if not all, unsupervised parallel and distributed systems. The framework introduces the concept of allowing the boundary conditions of a cellular automaton to be defined by a separately specified system, which can be any system that is capable of producing the information needed, including another cellular automaton. Using this interconnection concept, two forms of computational simplification are enabled: the deconstruction of a large system into smaller, modular pieces; and the construction of a large system built from a heterogeneous set of smaller pieces. This framework is formally defined using an interconnection graph, where edges signify the flow of information from one node to the next and the nodes are the various systems involved.

A library has been designed which implements the interconnection graphs defined by the framework for a subset of the possible nodes, primarily to allow an exploration of the field of cellular automata as a potential representational member of unsupervised parallel and distributed systems. This library has been developed with a number of criteria in mind that will allow it to be instantiated on both hardware and software using an open and extendable architecture to enable interaction with external systems and future expansion to take into account novel research. This extendability is discussed in terms of combining the library with genetic algorithms to find an interconnected system that will satisfy a specific computational goal. There are also a number of

novel components of the library that further enhance the capabilities of potential research, including methods for automatically building interconnection graphs from sets of cellular automata and the ability to skip over static regions of a given cellular automaton in an intelligent way to reduce computation time. With a particular set of cellular automaton parameters, the use of this feature reduced the computation time by 75%.

As a demonstration of the usefulness of both the library and the framework that it implements, a hardware application has been developed which makes use of many of the novel aspects that have been introduced to produce an interactive art installation named 'Aurora'. This application has a number of design requirements that are directly achieved through the use of library components and framework definitions. These design requirements included a lack of centralized control or data storage, a need for visibly dynamic behaviour in the installation, and the desire for the visitors to the installation to be able to affect the visible movement of patterns across the surface of the piece. The success of the library in this application was heavily dependent on its instantiation on a mixture of hardware and software, as well as the ability to extend the library to suit particular needs and aspects of the specific application requirements.

The main goal of this thesis research, finding a method that allows cellular automata to represent a much larger range of unsupervised parallel and distributed systems, has been partially achieved in the creation of a novel framework which defines the concept of interconnection, and the design of an interconnection graph using this concept. This allows the field of cellular automata, in combination with the framework, to be an excellent representational member of an extended set of unsupervised parallel and distributed systems when compared to the field alone. A library has been developed that satisfies a broad set of design criteria that allow it to be used in any future research built on the use of cellular automata as this representational member. A hardware application was successfully created that makes use of a number of novel aspects of both the framework and the library to demonstrate their applicability in a real world situation.

Acknowledgements

I would like to share my gratitude with the following people for their support throughout the time both leading up to and the duration of my Masters of Applied Science Degree. Without their encouragement and positive reinforcement this thesis would not exist.

To begin with, I would like to thank my supervisor, Professor Rob Gorbet, and Professor Philip Beesley for the opportunities that I have been given throughout the course of my Masters program to travel all over the world developing and installing the kinds of artistic installations that initially inspired the main topic of this thesis.

I would further like to thank both Professor Gorbet and Eric Kubica for their guidance and assistance in the development and writing of this thesis. The contents, organization, and overall topic of this thesis would not exist without our (sometimes lengthy) discussions of the possible topics involved, and without their direct feedback on the thesis document itself.

Special thanks also go out to all of the friends that I made throughout my undergraduate and graduate years here, both classmates and professors, for their willingness to listen to ideas, discuss problems, and simply for ensuring that I take the time to enjoy life.

My gratitude goes out to my family at all levels for their general support and encouragement throughout my university years up to this point. I moved halfway across the continent to go to university but their support and love followed along with me.

Finally, I would like to share my deepest thanks for my wife, Sarah, for her support, motivation, encouragement, and (most importantly) unconditional love. Without her presence, inspiration and insight to keep me grounded and focused on the world around me, any success that I may find in life is utterly meaningless.

I dedicate this work to my wife, Sarah,
and to those we both have lost.

Table of Contents

List of Figures

1. Introduction

The field of cellular automata was designed to aid in the understanding of the simplest types of unsupervised parallel computation. An individual cellular automaton is primarily an interconnection of elemental computational units, referred to as *cells* in a *cell matrix*, whose states vary depending only on the states of cells in a localized neighbourhood. In contrast to supervised parallel systems, in which tasks are divided up among parallel computing elements and the results combined to achieve a global outcome, the computation in a cellular automaton is unsupervised and happens only locally based on these neighbourhoods. The purpose of this dissertation is to present a new formal framework for considering interconnections of heterogeneous collections of cellular automata, to expand the tools available for design and analysis of unsupervised parallel and distributed systems.

Cellular automata are characterized by a number of *parameters*, the most important of which are cell geometry and neighbourhood, number of cell states, and state transition rules. The most basic form of cellular automaton consists of a square grid of cells (geometry) where each cell is either on or off (two states). Which specific cells are on or off can change over time based on a particular set of rules. The rules governing each cell's state are based on which neighbouring cells were on or off at the previous point in time. It is important to understand that using only these simple design parameters, a cellular automaton as a whole can reach a specific pattern of on and off cells even though the states of individual cells are only influenced by the states of their own neighbours. Through this ability to achieve a global task with only local knowledge, the field of cellular automata can be considered the most basic example of unsupervised parallel computation. In other words, this form of parallel computing consists primarily of a group of individual computational elements that are each carrying out tasks which contribute to an overall goal, without any particular element having knowledge of what the goal actually is.

The field of cellular automata spans a wide range of systems, from the simple example described above to complex designs known as universal computers that are able to emulate any other known computing system [1]. The following examples illustrate the breadth of applications for cellular automata: physics and traffic simulations [2], pattern recognition [3], digital image manipulation [4], cryptography [5], and technology-based art installations [6]. Due to their relative simplicity of specification, cellular automata are also used to model other more complex

computing systems to produce generalizable results [7].

Cellular automata are inherently parallel systems, but they are generally simulated on serial computers, which can lead to a significantly longer computation time compared to parallel implementations [8]. This longer simulation time may be one reason why, in a review of the literature, few examples were found of using cellular automata to simulate any large-scale parallel systems. Fully parallel implementations do exist in hardware [9], but the number of physical connections between modules becomes unwieldy for a cellular automaton of any useful size. To fully understand this aspect of possible system designs, a metric for the level of parallelism[1] in a system has been developed in this dissertation that is represented on the *x* axis of the computing spectrum in Figure 1.1. This metric is based on a ratio of the number of processing units used by a system to the number of basic computing elements in that system, where the cells are these basic elements for a cellular automaton. In this way, a given system can range from being fully parallel, if every element has its own processor, to being fully serial, for a very large system with many elements all computed on one processor.

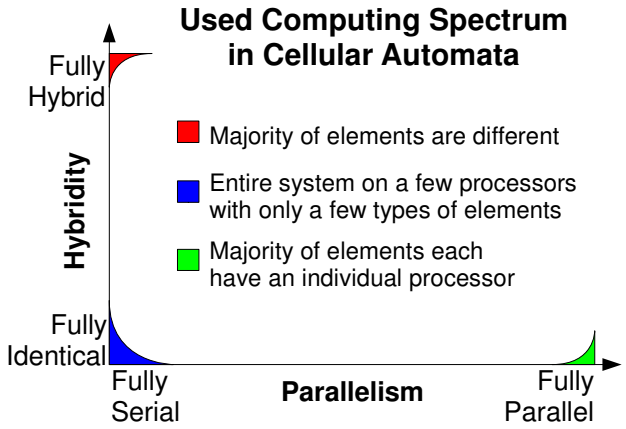

Figure 1.1. The current computing spectrum in cellular automata. Parallelism is the ratio of processing units to the total number of elements, while hybridity is the ratio of unique types of elements to the number of elements.

Although the classical definition of a cellular automaton requires all of its cells to have the same parameters, there are instances within the field of cellular automata where each cell is allowed to have different parameters. Such a system is known as a hybrid cellular automaton [10], and these systems are designed to allow different elements within the system to have different individual functionality, all used in harmony to achieve an overall goal. To allow an exploration of this aspect of the computing spectrum, a second metric has been developed for the level of homogeneity, or hybridity, in a system, which is used as the *y* axis of Figure 1.1. This metric is based on a ratio of the number of unique sets of element parameters being used to the number of elements in that system. Similar to the first metric, a system can range from being fully identical, when many elements are all using one set of parameters, to fully hybrid, where every single element has its own unique set.

1 The author recognizes that parallelism is an overloaded word and can have a number of different meanings depending on the topic, field, or subject. In the course of this dissertation, the definition above will be used.

As is evident from the occupied areas in Figure 1.1, there are three main regions of the computing spectrum that are currently in use within the field of cellular automata. The majority of work in cellular automata focuses on their implementation on large serial systems, with processors that often have two or four cores, leading to a large mass on the left extreme of the spectrum due to the ratio between processors (1 to 4) and cells (1000s). Since the classical definition of cellular automata dictates the creation of entire systems with the same parameters, there is a similar mass on the bottom extreme of the spectrum due to the ratio between parameters (1) and cells (1000s). Finally, in the field of cellular automata there is effectively no previous work that explores the middle range of either parallelism or hybridity, which leads to the three visibly separated regions shown.

As an example from the field of cellular automata in the middle of the parallelism spectrum, a set of processors could each compute the state of a particular subset of cells within a cellular automaton. This is between the extremes of having either a single processor computing an entire cellular automaton (at the left) or each individual cell having its own processor (at the right). An example of a medium hybridity system would be if each particular subset of cells used a unique set of system parameters. This is between the extremes of a typical cellular automaton (at the bottom), where all cells have the same parameters, to fully hybrid systems (at the top), where every cell's parameters are different. Although a few of these systems have been created in hardware as implementation shortcuts, the full range of parallelism and hybridity that makes up the computing spectrum has not been explored from a theoretical or design point of view.

This thesis research is directly motivated by the desire to create a framework that will allow the development of cellular automata which exist in these open spaces away from the occupied regions of the currently used computing spectrum[2]. This will create an environment where an increase in computational efficiency can be achieved while limiting the incidental increase in physical complexity. This motivation for exploring the middle range of Figure 1.1 is related to the trade-offs between serial vs. parallel, and homogeneous vs. hybrid, implementations. Fully

2 While this section describes the theoretical motivation, there is also a parallel, practical motivation, which spawned the theoretical investigation. In the summer and fall of 2010, I created the hardware and firmware for an artistic installation, Aurora, based on embodied cellular automata, comprised of 2592 cells. Since there was a desire for the behaviour of the different parts of the installation to be unique, or at least different from other adjacent behaviours, the need for a hybrid mix of cellular automata parameters became clear. This in turn led to the desire to have a systematic way to analyze a series of connected CAs having different parameters. The lack of an existing framework in the literature resulted in the development of the conceptual framework that is the focus of this thesis research, and a large portion of the developments in the library as well.

serial systems have relatively high computational overhead compared with fully parallel systems. However, fully parallel systems are much more complex due to the number of physical connections required to communicate cell states in each neighbourhood.

The exploration of the full range of parallelism and hybridity allows for a trade-off between computation time and complexity, as well as allows individual areas of a system to acquire specialized functions thanks to the potential for mid-range hybridity. By enabling a specification of the fraction of processors per cell, and the variability of parameters across the cellular automaton, this exploration also potentially allows the use of existing search and optimization tools to locate solutions in the field where there is currently no ability to investigate, due to the lack of such a framework. This outlines a need to develop a library that implements this framework and enables the creation of a wide range of novel types of cellular automata, on both serial and parallel hardware, for use in this search.

In the case of this dissertation, this need is satisfied through the development of a library based in part on a novel conceptual framework that enables this expansion along both axes of the computing spectrum. This is done by defining the components of a framework for interconnecting a set of individual cellular automata together into a graph, and implementing this framework as part of a larger cellular automata library. By varying the size of the individual cellular automata in the graph, and allowing each of them to have their own set of system parameters, the entire solution space of Figure 1.1 can be examined. An exploration is also made as part of this dissertation of one particular point in the solution space through the application of this library in the design of a specific parallel, hybrid hardware system.

1.1. Motivating Factors

The field of cellular automata was introduced in the 1940s by Jon von Neumann, considered to be the father of the modern serial computer architecture [11]. Even then there was a need foreseen for both parallel and serial systems in computational theory. In part due to the advent of the transistor making computation of any kind exponentially cheaper, the simpler serial and limited parallel[3] architectures exploded into their current computing monopoly while the harder to build, and seemingly unnecessary at the time, fully parallel architectures languished behind with little support. The need for fully parallel systems has only become apparent in recent years as computing needs have reached, and in some cases exceeded, the limits of the simpler systems.

3 In this context, limited parallelism is defined as any parallel system with fewer than ten processing cores.

To accelerate the transition to the kinds of parallel and distributed systems that are envisioned in academia and industry as the future of computing, a representative system must be chosen that can systematically model the entire range of potential systems to allow the discovery of generalizable results. In this dissertation, the field of cellular automata is investigated as one such representative system, since it can transparently and clearly represent a large subset of these systems using very simple parameters. Since completing global tasks using only local knowledge is a key component of unsupervised parallel computing, the suitability of cellular automata in representing general parallel systems is a result of their core functionality: any particular global task, ranging from needing local to global scales of information processing, is carried out by many simple units with only local knowledge and connections. In addition, the simplicity of the specification of all cellular automata, even those with complex behaviour, makes the field attractive as a representative group of their fellow parallel and distributed systems.

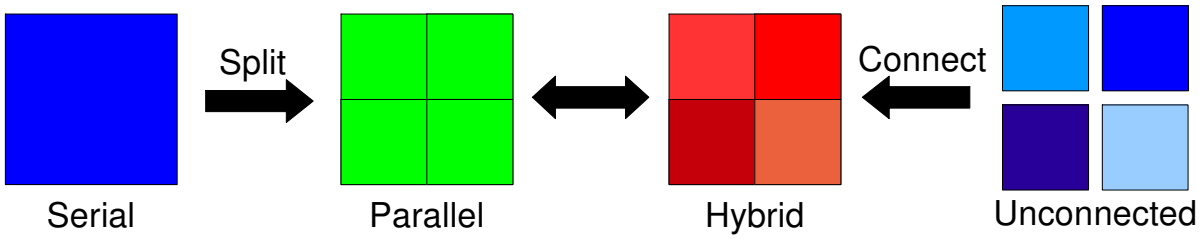

Figure 1.2. Various simple forms of system conversions that must be enabled by the framework.

Given the current state of the art in the field of cellular automata, and its currently used range of the parallel/hybrid spectrum as depicted in Figure 1.1, a major void exists in the full set of representable systems. This void can be filled through the creation of a conceptual framework that allows the definition of cellular automata that exist across the full range of possible levels of both parallelism and hybridity. As shown in Figure 1.2 using similar colours to those used in the computing spectrum graph from Figure 1.1, the framework would need to be able to *split* a large serial system into a set of parallel modules, each running on a unique processor, as well as being able to take a number of smaller unconnected serial systems and *connect* them into a larger hybrid system. It would also need to have the freedom to convert between these two new types of systems. On achieving these forms of system conversions, the framework would then enable cellular automata to act as representative systems for the majority of, if not all, parallel and distributed systems that currently exist throughout the parallel/hybrid spectrum.

Developing such a conceptual framework, and a library that implements and enhances its abilities, is the overall motivation of this research. Any library that implements this framework

will also need to be able to handle the simulation and implementation of the full range of possible cellular automata on both software and hardware, as well as being easily usable in current and future research. To shape and target the design of the library, these key motivational factors have been condensed down into specific criteria for success in the next section. Due to the vast amount of potential research in the use of cellular automata as representative members of most unsupervised parallel and distributed systems, these criteria are primarily focused on the key aspects needed to drive the development of a high-level library that initially uses a limited set of system parameters but is designed to be extendable in the future.

1.2. Design Criteria

A set of criteria has been developed to guide the successful design of the library. These criteria have been developed to ensure the library allows the implementation and development of software- and hardware-based cellular automata for use as representative members in the investigation of parallel and distributed systems. For this library to be successful at a high level it must: implement the new conceptual framework, be usable on any suitable combination of hardware and software, use an open architecture to allow interfacing with other systems, replicate the functionality of the majority of currently available functionality, and allow simple extensions to create new functionality. These five core aspects are used to define the set of criteria that the library must satisfy.

The primary criterion for the success of this library is the implementation of the new conceptual framework. To achieve this, the library must have the ability to separate the necessary computation in a cellular automaton into parallel modular pieces to spread the usable spectrum of computation across the full range of parallelism. This allows for the multitudes of computations needed for a massive cellular automaton to be split into smaller computational modules, as simply demonstrated in the left system transition of Figure 1.2. The library must also allow each individual module to specify its own set of parameters, so that the overall system can exist anywhere along the full range of hybridity, as demonstrated in the right transition of Figure 1.2. Depending on the scale and speed that is necessary for the task at hand, the modules could be implemented on the different cores of a multiple-core processor, a subset of servers in a server farm, or even a distributed set of microcontrollers. The use of computational modules would also allow a measurable increase in the speed and size of possible cellular automata with every

modular addition of resources, similar to the methods currently used to increase the throughput of server farms by adding new banks of servers as needed.

The secondary criterion for success is to ensure that any improvements that are developed as a part of this library are directly applicable in both software and hardware implementations so that the computational modules used to satisfy the primary criterion can be directly instantiated. To accomplish this, the library will need to be built from a set of fairly basic computational modules that can be directly instantiated in software or hardware. Steps will also need to be taken to reduce the typical complexity of hardware implementations of cellular automata for the same reason. There are a number of key physical elements that will need to be optimized before hardware implementations are as viable as their software counterparts, including the number of connections used, communication systems, power demands, and storage capacity. However, the key non-physical element that contributes the most towards ensuring that hardware implementations are as viable as those in software is an increase in overall computational efficiency. This boost in efficiency would also act to ensure that power demands were at a minimum and could potentially improve other physical elements as well, such as by reducing the number of necessary connections or frequency of communication.

Along with the primary and secondary criteria, there are a few ancillary criteria that have been identified which will increase the usefulness of the library in future research. The first criterion is that the library architecture is open, to allow for simple access to data from any of the various systems that are generally used alongside cellular automata, such as the genetic algorithm approach to evolutionary optimization. To achieve this criterion, an interface must be developed as part of the library that will allow external systems to query useful data from the library, as well as to manipulate and analyze both local and global properties of the cellular automata that are implemented. This will create the capability to externally measure and interact with the cellular automata using the library in both high- and low-level ways, such as allowing a genetic algorithm to determine the density or global state of a particular cellular automaton or enabling a graphical interface with the ability to specify initial cell state conditions manually.

The second ancillary criterion for the success of the library is the inclusion of the current standards of functionality in the field of cellular automata. The field as a whole contains a large number of different possible system parameters, of which only the most important have been introduced here, which can be used to fully describe a large body of previous work. To model all

of this previous work, a library must provide the ability to use any of the possible parameters. However, the majority of previous work uses only a limited subset of the potential parameters that are available, and it is this limited subset that must be implemented to satisfy the criterion. While this initial work will focus only on a subset of the full range of parameters, the next criterion ensures that the library is able to allow expandability to facilitate future development.

The last ancillary criterion is creating the capability of extending the core library. This allows the introduction and development of new functionality above what is currently available in this and other cellular automata libraries. There is a wide potential for new forms of functionality, and with each form there exists the ability to define and solve new tasks that cannot currently be specified. As stated in the previous paragraph, this capability will also allow all of the possible system parameters that currently exist, and many that may exist in the future, to be incorporated into the library. The integration of these criteria in the design of the library can be regarded as a completely successful design.

The next section provides a high-level overview of the methods used to develop a library that is designed to satisfy these criteria. The achievement of fine control over the levels of hybridity and parallelism of cellular automata in this library relies directly on a particular implementation of the interconnection framework that is the core novel component of this research. The achievement of the remainder of these criteria, such as increasing efficiency and allowing extendability, relies on the development of specific novel aspects of the library that are also introduced in the next section.

1.3. Methodology

To achieve the criteria that were set out in the previous section, a number of changes are necessary compared to common methods of building cellular automata libraries. These changes are realized primarily through the design of a framework that allows an expansion throughout the computing spectrum in the field of cellular automata, and through the creation of a library that implements both this framework and a number of additional novel aspects. A hardware application of this library is also developed, where a subset of the library functionality remains in software while the remainder is replaced with functionally-equivalent hardware components. This application is designed to serve as a demonstration of both the framework and the library being applied directly within a physically realized system in a way that will test the successful

integration of all of the design criteria.

The biggest fundamental change from more common implementations, such as [12], and the key idea behind the new conceptual framework, is a mechanism which allows two different types of computational simplification: breaking down a large system into parallel pieces and uniting a number of different parallel systems into a single larger system. In this way, the new framework allows for cellular automata, and therefore the set of parallel and distributed systems they can simulate, to occupy a much broader area of the computing spectrum shown in Figure 1.1, as these two types of simplification can be restated as parallelism and hybridity, respectively. The success in implementing this framework in the library, and therefore achieving the primary criterion, is built on two design aspects: the separation of computation into modules and the joining of a mix of cellular automata into a more complex system. The framework at its core creates the ability to interconnect a number of individual systems in a variety of ways to produce new and interesting computational systems throughout the parallelism and hybridity spectrum.

In addition to implementing this framework and easily achieving the second ancillary criterion of duplicating the limited set of system parameters that are available in other libraries, this design includes a number of interesting and unique aspects of its own. The main new aspect of this library is the creation of a method for quickly skipping over any areas of a cellular automaton that are not changing over time. This method has the potential to vastly increase the efficiency of the system overall, as the majority of the computational time for any cellular automaton is spent on the calculation of which cells are changing state at any given time. Since there are many different types of cellular automata, with many varied forms of local and global changes in activity, at any particular time this method could either be helping or hindering the efficiency of the overall system. These varying effects on the overall efficiency, as well as the effects of the implementation of the framework, are analyzed for a number of specific cellular automata. This new aspect, as well as others that will be discussed as part of the library description, serve to achieve the final ancillary criterion of creating new functionality.

An additional feature of the library is the inclusion of an interface which allows external systems to analyze and interact with any cellular automaton that it has created. This includes being able to both measure the current state of the system and manipulate its local and global properties. This interface spans a broad range of possible functions which includes the ability to both change the state of a single cell in real-time using a graphical interface, and classify a set of parameters

based on the system's reaction to a specific set of initial conditions. Extensions to the interface, as well as to the library itself, are also possible to allow future research to easily build upon its core components. This interface to the library directly achieves the first ancillary criterion of having an open architecture.

Finally, this library is extended and converted into a mixed software and hardware implementation that is used in an embodied system to serve as both a demonstration of the achievement of the secondary criterion and the use of the library. The library, and by extension the framework, is directly applied to allow a set of different cellular automata to each be computed locally on their own hardware, while acting as a single massive cellular automaton, and the region-skipping method is applied to reduce the time and power needed to compute the global state of each individual cellular automaton. As a part of this hardware implementation, an evolutionary computing method known as genetic algorithms is combined with the library to allow the system as a whole to discover a particular hybrid set of parameters that will solve a specific task. This ability to work with a joined set of different cellular automata increases the solution space for a given task, while potentially allowing more efficient solutions to existing problems or even solutions where none currently exist.

1.4. Contributions

The main goal of this thesis research is to enable an exploration of the majority of the available computing spectrum by using the field of cellular automata as a representative member of a subset of parallel and distributed systems. As a direct solution to this goal, a conceptual framework is developed that allows the field of cellular automata to emerge from the edges of the computing spectrum and spread across the full range of both parallelism and hybridity. To implement this framework, and create the capability for a practical exploration of this spectrum, a library has been designed that is judged based on a number of design criteria. These criteria ensure that the library can be instantiated and used in a wide range of potential investigations throughout the hardware and software realms and the computing spectrum. To demonstrate the use of the library, using a mixture of software and hardware, and to underline the capabilities of the new conceptual framework, an embodied hardware application has been developed that makes use of all of the library components needed to successfully achieve the chosen criteria.

1.5. Outline

This section is intended to provide the reader with a chapter by chapter summary of the core contents of this dissertation. This has been done to both guide reading and to provide a high level skeleton which will be fleshed out in the chapters to follow.

A solid foundation is provided in Chapter 2 of the core concepts in the field of cellular automata, as well as discussions of a number of related topics. These topics include two specific types of cellular automata, a method for computational analysis, hardware applications, and a method of searching the field for the solution to a particular problem. The purpose of these discussions is to ensure that the reader has a suitable understanding of the underlying topics before moving into the details of the framework itself. This is done not only to simplify comprehension, but also to allow the reader to appreciate the differences between the state of the art and the developments that arise in the course of this thesis research.

Once this foundation has been laid, Chapter 3 defines the framework that is the primary contribution of this thesis research. This framework has been developed to define all of the conceptual elements needed to allow the interconnection of a set of cellular automata. The chapter contains a description of the various elements that are introduced to the field of cellular automata, as well as definitions of how they function with respect to the field in general. The core functional element of this framework is an interconnection graph in which the main nodes are cellular automata, other nodes are interconnectable systems, and the edges between all of these nodes define how each pair is connected.

A newly created library for cellular automata research is described in Chapter 4 that implements this framework, as well as a few other unique developments. The chapter begins with a brief discussion of the core implementation aspects of the library, including scope limiting, followed by three dedicated sections. The first section describes the novel aspects of the library, which primarily consists of two things: interconnection and being able to skip static regions while updating. The next section discusses the changes in efficiency for each of these primary aspects to quantitatively demonstrate the overhead and gains associated with each, with a focus on the dependence of static region skipping on global dynamics. Finally, the last section outlines a few specific applications including interactive systems and the use of the library in searching for solutions to specific tasks.

The description of an interactive art installation, which was built using the concepts introduced and developed in both the framework and library, is contained in Chapter 5. The installation in question, named “Aurora,” is used primarily in this dissertation as a demonstration of the application of the novel aspects of the framework and the success of the library design. It also serves to introduce various hardware issues that arise in the implementation of cellular automata as embodied systems, and discusses how a few can be resolved by specific aspects of the library.

Finally, the main conclusions of this thesis research, and potential future research topics, are discussed in Chapter 6. The library is determined to satisfy all of the criteria that were set out in Section 1.2, and the specifics of how each criterion was satisfied is discussed in detail. Based on this success, the framework's potential uses in the field of computational theory are explored as future work. In particular, there is a discussion of its use in the analysis of the computational robustness of hardware implementations of cellular automata in the face of various forms of software and hardware failure. The application of the library to a few potential tasks is described, with an emphasis on tasks other than those typically seen in the field of cellular automata.

2. Background

To understand the contributions of this thesis research, an introduction to the field of cellular automata is necessary, along with a description of how the field can be combined with both embodied hardware and evolutionary computing. Most of the recent work that is discussed in this section, in both cellular automata and their combinations with other systems, is done in one dimension (1D) with very little work in two or more dimensions. In fact, one of the main factors that could be affecting the absence of multidimensional research in the field of cellular automata is the lack of a library similar to what is outlined by the criteria from Section 1.2. Without a physically implementable library that would allow fast updates of very large parallel systems, the computational needs and required infrastructure of multidimensional cellular automata may have led to their avoidance on the part of the majority of researchers in the field. This would explain the presence of extensive cellular automata work in 1D with only extremely minimal work in 2D and beyond, even though the field of cellular automata first began in 2D.

There is an extensive realm of systems that fit into the field of cellular automata. There are many different types of cellular automata, and within each there exists a range of possible global dynamics. This range of dynamics includes systems that have a static or periodic structure, are statistically random, or even act chaotically. There are even known combinations of system parameters that create a cellular automaton that is capable of universal computation [1]. Thanks to their simplicity in definition, cellular automata are an excellent tool for the analysis of distributed systems and can be used to emulate and simulate both serial and parallel computing systems across a broad spectrum of research fields. A core taxonomy that will be used throughout this dissertation is developed in the next section, as well as brief explorations of both the history and more recent work in the field of cellular automata.

After the core section on cellular automata, two related fields are introduced in the following two sections that are both used directly as design components in the hardware application outlined in Chapter 5 of this dissertation. The first of these related fields is the implementation of cellular automata in hardware in the form of embodied systems. The general concepts of embodied systems are introduced and compared to more popular embedded systems, and details of embodied forms of cellular automata and some recent work in their development is discussed. The second related field is a subset of the larger field of evolutionary computing that are known

as genetic algorithms. Again, their general concepts are described before the introduction of their combination with cellular automata and some recent work are discussed.

2.1. Cellular Automata

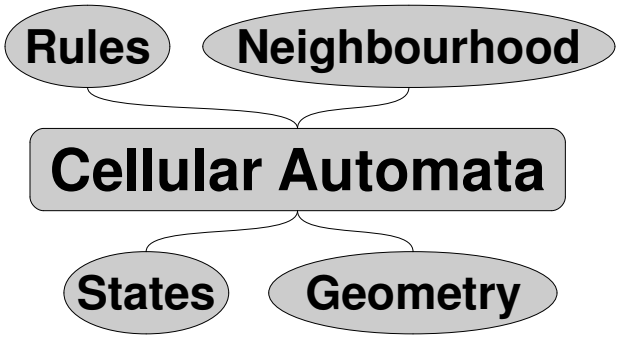

Figure 2.1. Concept map of the core set of parameters that define any cellular automaton.

As mentioned above, the field of cellular automata consists of a wide spectrum of possible systems, but the majority of this spectrum can be specified using only a few simple parameters. These parameters, shown in Figure 2.1, are generally concerned with two core aspects of a particular cellular automaton: how individual cells within the system are specified, and why and when the state of these cells changes over time. The cells in a cellular automaton are defined based on their geometry, their state, and which cells are considered to be neighbours. The progression of cell states is based on a rule which dictates what the new state of a cell will be given the current state of that cell and the states of its neighbours.

More formally, a cellular automaton is a mathematical model that can be viewed as a set of cells distributed spatially across a lattice, each of which acts as a finite state machine that uses only local knowledge of neighbouring cell states to progress through a limited set of states over time. The spatial distribution in 2D typically uses a full lattice of shapes that completely covers an infinite plane. To create a cell matrix that contains all of the cells in the system, each individual shape in the lattice becomes a distinct cell which has a particular state. Every cell also has access to the state of its neighbouring cells' states, where the local neighbourhood can be defined in many different ways but is generally based on how far away cells are from each other.

In 2D, in the case where an infinite plane is not used, there are also boundary conditions which define how the cells on the edges of a cellular automaton relate to each other in terms of which cells are neighbours. This most often takes the form of connecting the top edge of a cellular automaton to the bottom, and the right to the left, to create an environment that is toroidal, or doughnut shaped, to simulate an infinite plane. However, other boundary conditions seen in the literature include defining the edges as static cells, having the cell states chosen randomly at run-time, or acting as reflectors which take on either the state of the direct edge cells or their

neighbouring cell one step away from the edge, as if in a mirror [13].

The rule that is used to cause a change from one state to the next is generally known as the transition rule or function. The combination of the size of the neighbourhood and the number of states each cell can have dictates the number of possible transition rules for a particular cellular automaton. These locally informed rules are typically deterministic, synchronous, and act concurrently to calculate the next state of every cell in the system at once, based only on each cell's local knowledge of its neighbourhood at the previous time step. For the purpose of this work, probabilistic, asynchronous, and any other forms of rules will not be considered.

The shape that is used as the basis for the cell matrix can be chosen from a list of any of the various geometric objects that can be tiled to completely cover a plane, commonly known as tessellations. There are many different mixed sets of shapes that can achieve this coverage, as is evident from the various methods of laying stone tiles, but the majority of these tilings create a number of different relationships between the various cells. These relationships have a direct impact on how neighbourhoods are defined, as can be seen in the example of Penrose tiles in Figure 2.2, which have been used to implement a cellular automaton in the past [14] but require the definition of a number of different neighbourhoods to work.

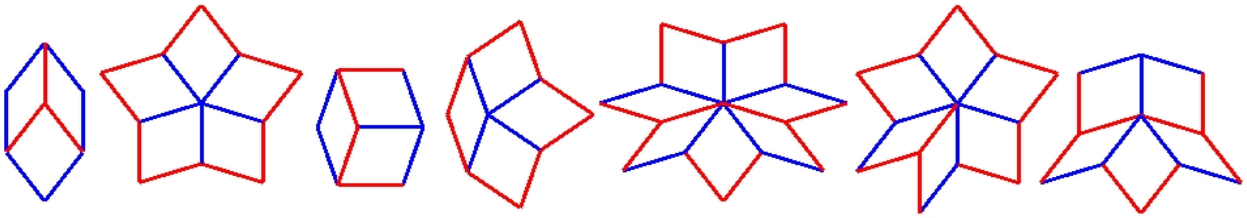

Figure 2.2. Shows the seven different allowable vertex relationships in a Penrose tiling using rhombs. These can be combined to build a variety of neighbourhoods with a large range of different numbers of neighbours.

To avoid the need for complex neighbourhood calculations, a common relationship between shapes is desired. To this end, only homogeneous shapes that always have the same number of neighbours will be considered in this dissertation. Even with this restriction, there are still a number of potential tessellating shapes available: any rectangular or triangular shape, most hexagonal shapes, and one pentagonal shape. The hexagonal tiling restriction requires only that two of the shape's sides are parallel and congruent, while there is only one pentagonal tiling without mixed neighbourhoods. It is important to realize that the shape of the individual cells within a cellular automaton have little effect on the overall shape of the cell matrix, other than to contribute their particular relationships to the cell-level details of the edges of the system.

In addition to the need to choose a shape for the cells, the range of possible cell states must be chosen. This state can range from a discrete two-state system, generally referred to as a binary cellular automaton, through to as many states as there are integers, to purely continuous states that have only a maximum and minimum value and can take on any real value in between. The choice of the type of state (discrete or continuous) and the range (typically from 0 to a positive value) has a direct effect on how the transition rule is defined. With a continuous state, the transition rule is a function that takes the cell states as arguments and calculates the new state. For the remainder of this dissertation, it is assumed that the state is always discrete as the current abilities of computing systems has little distinction between the two at a high enough number of discrete states. However, it is important to note that the range of applicability of this research remains valid even without the restriction of using only discrete states.

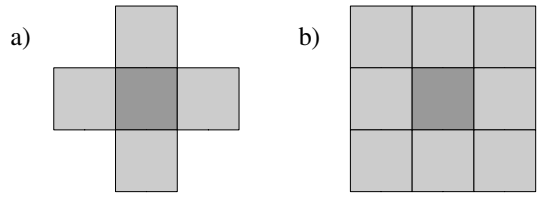

Figure 2.3. The von Neumann (a) and Moore (b) neighbourhoods of an inner cell (dark grey) in a square tessellation with a one cell distance.

The selection of the neighbourhood is a key parameter of the specification of a cellular automaton, as it dictates, along with the number of possible cell states, how many transition rules are available. There are many different neighbourhoods that can be chosen, and which one to use depends greatly on the desired behaviour of the cellular automaton in question. Any change to the symmetry or size of the neighbourhood can have vastly different results. The two most popular types of neighbourhood, shown in Figure 2.3, are both named after historical figures in the field of cellular automata: von Neumann and Moore. The von Neumann neighbourhood (Figure 2.3a) is defined as including all cells that are within a given number of orthogonally connected cells, while the Moore neighbourhood (Figure 2.3b) is defined as including all cells that are within a given number of radially connected cells. In this case, orthogonal distance is measured by how many shared cell edges must be crossed to get to a cell, while radial distance is based on how many shared cell edges or corners must be crossed. For a visual method of distinguishing between these two types, think of the von Neumann neighbourhood as diamond shaped while the Moore is round.

When a particular cellular automaton is not implemented on an infinite plane but as a finite sized cell matrix, as is usually done, there is a requirement to deal with how neighbourhoods are specified on the boundaries of the system. These boundary conditions can take many different

forms, each of which can contribute very different changes in the behaviour to the overall system. The simplest form of boundary conditions are static, where all missing neighbours are considered to be in a predefined state (typically 0) that does not change over time. Although this is very simple to implement, it is generally destructive to any information processing that is being done. A simple extension of this method generates a random valid state each time a missing neighbour's state is needed. This method introduces random noise to the edges of the system, which can be either beneficial or damaging, depending on the task at hand.

The most common type of boundary conditions that are used in the research world are spatially periodic boundaries. In most periodic boundary conditions, the top edge cells of a square-shaped cell matrix are connected to the bottom edge cells as if the entire matrix were wrapped around a tube, and the left and right edge cells of this cylinder wrap in a similar way. Given this tube that has been wrapped end to end, periodic boundary conditions are also commonly referred to as a toroidal environment, as the cell matrix acts the same as the surface of a torus, or doughnut (Figure 2.4). Using these boundary conditions, an infinite plane can be simulated if the cell matrix is large: any patterns that appear to move or grow across the system will continue to move or grow indefinitely until they interact with themselves. An example of a pattern that appears to move across the cell matrix, commonly known as a 'glider', is discussed in Subsection 2.1.1.

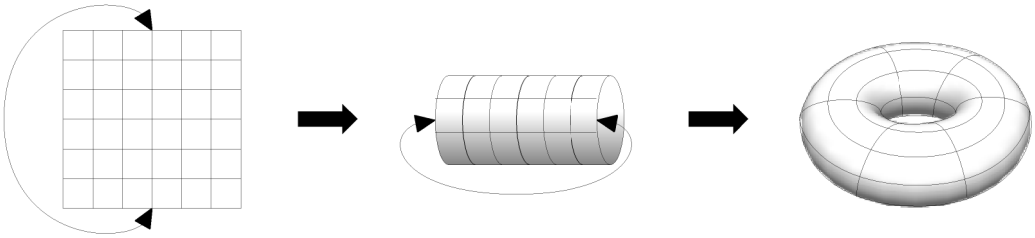

Figure 2.4. Example of how a square cell matrix can be given periodic boundaries in both directions to create a toroidal environment. Although the cells on the torus appear distorted, their neighbourhoods remain the same.

There are many possible types of transition rules, with the most popular types known generally as fully specified, rotationally or axially symmetric, and various totalistic rules. A selection of how these rule types label a given set of cell states is shown in Figure 2.5. Each of the different rule types defines the next state of a cell based on different patterns in the cell's neighbourhood. When using fully specified rules (FS), every possible pattern of neighbourhood states is used, while with symmetric rules, only the sets of symmetrically equivalent neighbourhood patterns are used. The equivalence between neighbourhood patterns in symmetric rules changes based on the type of symmetric rules used: in axial rules (AS), patterns are equivalent if they are mirrored

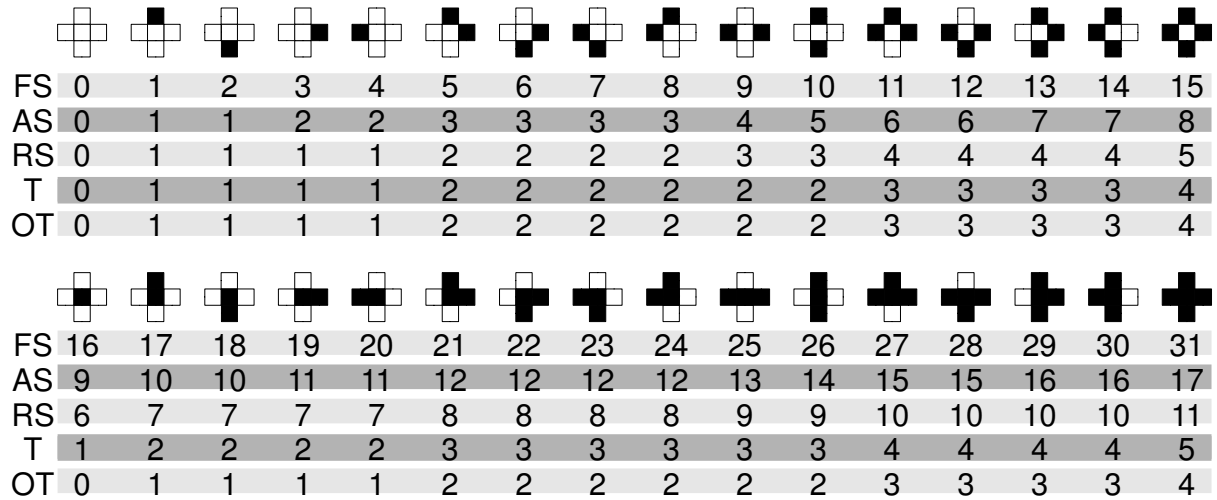

Figure 2.5. Example of how the various types of rules label each set of states using a size 1 von Neumann neighbourhood in a square tessellation. The numbers represent the specific unique rule index that would be used.

copies along a given axis, while in rotational rules (RS) patterns are equivalent if they are rotated copies. The most easily defined types of rules are totalistic rules (T), which are based only on the number of cells in a neighbourhood in each particular state, without needing to know their specific pattern. There are a number of simple extensions that are also commonly used in the realm of totalistic rules to expand their usefulness, shown in Figure 2.6.

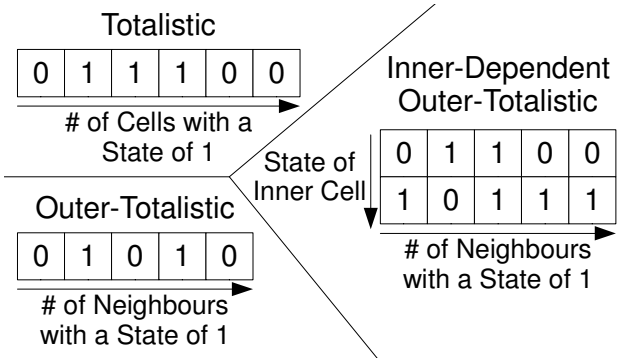

Figure 2.6. Examples of how the most common totalistic rule types are indexed and specified.

The omission of the state of the inner cell from a totalistic rule is known as an outer-totalistic rule (OT), and means that the state of any particular cell cannot affect its own next state. This extension to general totalistic rules is typically done in tandem with the further extension of inner-dependence, which creates the ability to define a distinct set of outer-totalistic rules for every possible inner cell state. This means that the state of a cell has a direct influence on its own next state, in a more powerful way than in general totalistic rules where it shares equal influence with its neighbours. Compared with general outer-totalistic rules, the addition of inner-dependence increases the total number of potential rules by a power of n, the number of possible inner cell states, along with increasing the size of each rule by a factor of n.

When using any discrete totalistic rules, the simplest method of defining a transition function is to create a look-up table for the next state, indexed in a unique way using the number of cells in

each state within a neighbourhood. In a two-state system, this index is usually the number of cells in the 'on' or 1 state, while the contents of the table are simply the next state for the cell (1's and 0's). As shown in Figure 2.7, if a cell in a four neighbour two-state outer-totalistic system has three neighbours in the 'on' state (1, black), and the value at $i = 3$ in the look-up table is a 1, then the cell's next state is a 1. Note that due to the outer-totalistic extension, the previous state of the central cell is not used in any way.

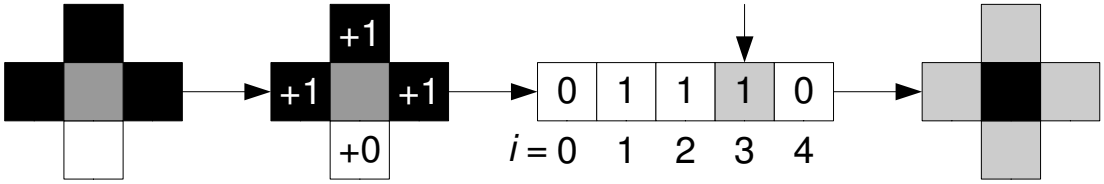

Figure 2.7. Example of how an outer-totalistic rule is applied using a size 1 von Neumann neighbourhood in a square tessellation. Grey cells indicate a cell state that is either irrelevant to the rule, as is the case with the inner cell before the transition, or unknown, as is the case with the outer cells that may have changed based on unshown cells.

With the added complication of inner-dependence, the number of look-up tables is multiplied by the number of possible inner cell states, n. In most implementations this takes the form of a 2D matrix of cell states, where the column index into a given look-up table is provided as before while the row index to select a particular table is simply the value of the current state of the inner cell. Since the number of tables in the matrix is always the same as n, each column can be replaced by compressing that column's contents into a single value. This is achieved by multiplying each look-up tables' contents by n to the power of the row index, then adding together the new contents of each column. Figure 2.8 demonstrates this compression for a particular four neighbour two-state inner-dependent outer-totalistic rule.

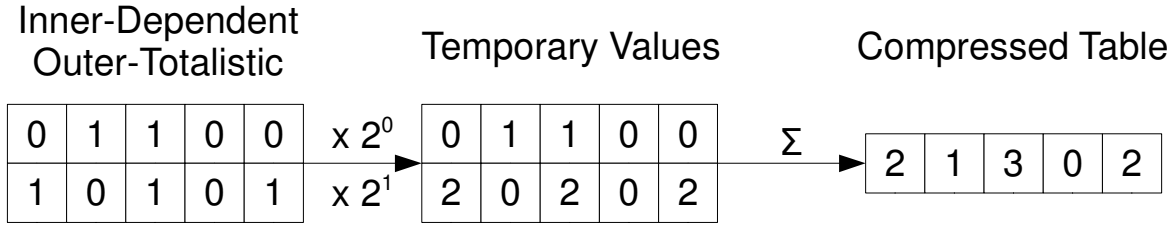

Figure 2.8. Example of how an inner-dependent outer-totalistic rule is compressed into a single table of values.

The final parameter to be specified is also the only parameter that is more often a part of the problem definition or randomly chosen than a part of the actual system specification itself. This parameter consists of the initial state that each individual cell in a given cellular automaton starts in before the system begins to progress through time, known as initial conditions. In many

problems, these initial conditions are actually the input to the problem and the remainder of the system parameters are used to attempt to manipulate these initial states to find some sort of output. In other problems, there are a set of very specific initial conditions for large regions of the cell matrix that act in tandem with the system parameters to accomplish a task, while a small region of varying cells is used as the input.

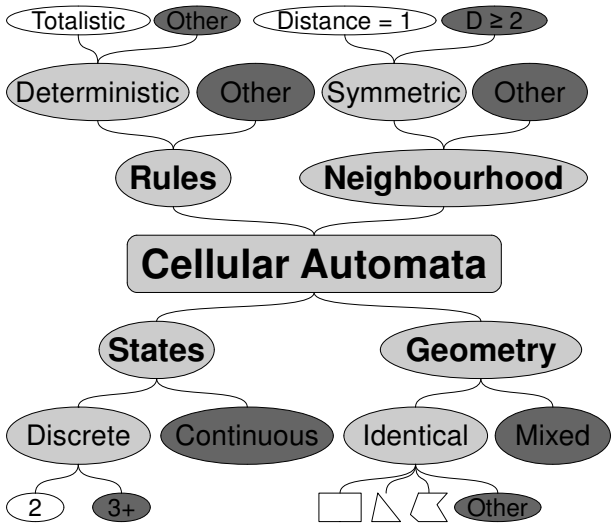

Figure 2.9. Concept map of the limited set of system parameters that are available within the library. White nodes are usable parameters, while dark grey nodes are not. Light grey nodes indicate partial implementation.

The field of cellular automata is far too large for any kind of broadly inclusive research to be done in the scope of a graduate thesis, or even in any single document. Therefore, in the implementation of the library in Chapter 4, only a subset of the possible parameters will be used as shown in Figure 2.9. However, since any unconnected cellular automaton is represented by only a few basic graph nodes, the framework developed in the next chapter is able to accommodate most, if not all, of the current field of cellular automata. The subset of parameters used in the library (in white) is: 2D shapes that are homogeneous and can tessellate infinitely, two discrete states, Moore and von Neumann neighbourhoods with a size of one cell, and all forms of totalistic rules including outer-totalistic and inner-dependent variations. The specific reasons behind each of these choices of parameter limitations are discussed in detail as a part of describing the implementation of the library in Chapter 4. However, for the purpose of this thesis, this subset of parameters will be used as the standard in the field of cellular automata.

Due to the vast numbers of possible combinations of parameters, there are only really two options for using cellular automata as computational task-solving systems. The first consists of exhaustively searching for, or carefully hand-picking, a set of parameters to achieve a necessarily simple goal. Due to the inherent difficulty of finding useful parameters in the vast space of cellular automata, this manual design or search generally includes the need to define very specific initial conditions as suggested in the previous paragraph. This is the most common method used in research, going as far back as when John von Neumann created the very first cellular automaton in the 1940s [11]. A summarized listing of the major milestones in the history

of the field of cellular automata is discussed in the first subsection.

In addition to the common parameters previously discussed, there are a wide range of other parameters and aspects of cellular automata that have been introduced recently that include: dissipative systems, where external inputs can change behaviour [15]; hybrid systems, where each cell can have different parameters [10]; and hierarchical systems, where each cell's state is based on an entirely separate cellular automaton [16]. A few specific examples of these recent developments in the field of cellular automata are discussed further in Subsection 2.1.2.

Further extending the idea of modifications to the traditionally software-based field of cellular automata, a number of hardware applications merit investigation. To this end, Section 2.2 begins with a brief introduction to the field of embodied hardware, including how it relates to more common embedded systems. It then moves on to a discussion of how cellular automata are generally implemented as an embodied system, and a selection of recent work is discussed that deals with systems used to accomplish a variety of tasks that range from computationally complex control systems to architectural designs made strictly for their visual appeal.

Other than the exhaustive searching and carefully hand-picking option, the second option for designing cellular automata to achieve specific tasks, which is steadily growing in popularity, consists of searching the large potential parameter-space using automated search methods. The most promising method used in this application is a form of evolutionary computing known as genetic algorithms. Since genetic algorithms are specifically designed to be able to search large, non-linear spaces that are defined by a number of discrete or continuous numerical parameters, they are an excellent fit for this type of search. A brief introduction to genetic algorithms is given in Section 2.3, along with details and recent work on their use with cellular automata.

2.1.1. Historical Development

As mentioned previous in this section, the field of cellular automata began with John von Neumann in the 1940s with an investigation into the computational aspects of self replication [11]. In his efforts towards creating a self-replicating machine, he was inspired by Ulam [17] to use a regular structure, similar to the biological cells that Ulam himself was working with, as the environment for replication. Using a 2D environment of square cells, each of which could take on one of 29 different states, and a fully specified transition rule based on a cell's own state and the state of its four edge-connected neighbours (the von Neumann neighbourhood of size 1), von Neumann created the first cellular automaton. With this environment, he was successful in

designing a self-replicating machine which was, in fact, a universal constructor [1]: it was capable of building any structure that was possible within the environment given a suitable localized set of initial conditions for the cellular automaton. Unfortunately, von Neumann passed away before he was able to publish his work so it was not brought to the public's attention until 1966 when Arthur W. Burks edited and completed it on his behalf. Leading up to and following its public release, a number of major developments came about in the field, most of which were dependent on von Neumann's work: Edward Moore [18], who proposed the first finite state machine in 1956; Edgar Codd [19] and Christopher Langton [20], who built on the research on self-replicating machines in the late 1960s and early 1970s, and Edwin Roger Banks [21], who wrote a doctoral thesis on information processing and transmission in cellular automata in 1971.

The next major step in the development of the field of cellular automata came with the introduction in the 1960s of John Conway's "Game of Life", based on his lattice experience with John Leech and Ulam's cells [22]. The Game of Life is not truly a game in the classical sense, but the plethora of interesting patterns and behaviours that come about with only subtle changes in its initial conditions make it very entertaining to play with nonetheless. It consists of a 2D square tessellation of cells using a size-1 Moore neighbourhood with two states per cell, typically labelled as *alive* or *dead*. The inner-dependent, outer-totalistic transition rules are simple: if a dead cell has exactly three live neighbours it becomes alive, otherwise it stays dead; if a live cell has exactly two or three neighbours it stays alive, otherwise it becomes dead. In the compressed inner-dependent outer-totalistic rule system that was described earlier, this is [0 0 2 3 0 0 0 0 0].

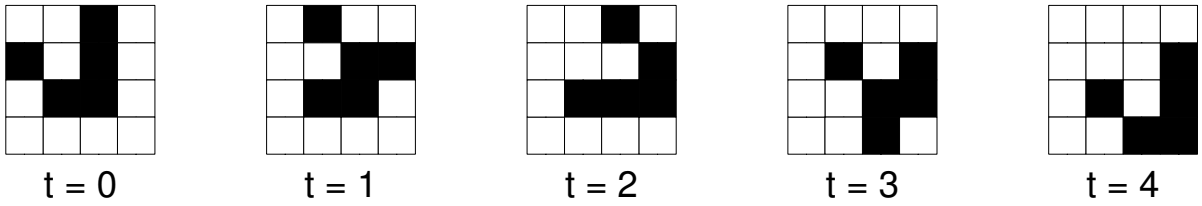

Figure 2.10. A moving pattern of cell states known as a 'glider' in John Conway's Game of Life cellular automaton. The pattern of 'live' cells at t = 0 is recreated at t = 4, after having moved one cell to the right and one cell down.

These simple rules led to so many interesting patterns, such as the 'glider' in Figure 2.10, and algorithmic possibilities, thanks to the ability to create logic gates with these 'gliders', that they were presented in Martin Gardner's Mathematical Puzzles column for Scientific American [23]. This column's popularity in the academic world led to a number of newsletters and a plethora of research papers, the most important of which are contained within the recently released book

“Game of Life Cellular Automata” by Andrew Adamatzky [14]. A few of the key discoveries that were made based on the Game of Life, the majority of which are also discussed in this new book, include: its ability to support universal computation [24]; a super-set of “Life-Like” rules, which have many similar properties to the Game of Life rules [25]; and even the use of common cell state patterns in the Game of Life to create both music ([26]..[29]) and art ([6], [30]).

The last major milestone in the history of the field of cellular automata comes courtesy of Stephen Wolfram and his series of papers on the topic in the early 1980s [7]. These papers were mainly concerned with a class of cellular automata that Wolfram labelled as “Elementary Cellular Automata”: 1D cellular automata with two states per cell and only the two closest cells for neighbours. Based on this limited set of neighbours and states, there are only 256 (2^2^3) different fully specified rules, as shown in [31]. Wolfram's main contribution to the field is a system of classification for cellular automata that categorizes them into four distinct groups, and has been shown to remain a valid classification mechanism in multidimensional cellular automata as well as the simple 1D systems for which it was developed [32]. Briefly, these classes are: Class 1, eventual progression of every cell in the system to a homogeneous state regardless of initial conditions; Class 2, reduction to static or temporally periodic patterns from any initial conditions; Class 3, visibly chaotic patterns from the majority of initial conditions; and Class 4, a mix of periodic and translating patterns that show complex global structure and are capable of information processing and transfer. Further work by Matthew Cook showed that 'Rule 110', the only Class 4 Elementary Cellular Automata, is capable of universal computation [33].

In the time since these major milestones, there have also been a number of minor developments that have grown the field of cellular automata to its current status as a major part of the computational space. Some of these developments have been briefly introduced earlier in this section, and the main ones are discussed further in the next subsection on more recent work in the field. The main development that is important to this work in terms of cellular automata is the introduction of hybrid systems: where individual cells within a system can each have different parameters. Following a discussion of hybrid systems, and other recent developments in the field of cellular automata, there are two sections dedicated to topics that are also important to this work: hardware implementations of cellular automata, in Section 2.2, and the use of genetic algorithms to search for parameter sets, in Section 2.3.

2.1.2. Recent Work

In this subsection, a number of examples are used to demonstrate a subset of the potential changes that can be made to a more typical cellular automaton to expand its abilities and utility. Although this can take many forms, the two main changes to the overall system that are explored below are the use of hybrid and dissipative elements. Following these two examples, there is a brief discussion of a method for analyzing specific parameters of a cellular automaton for their intrinsic computational abilities, which is particularly effective when there is no quiescent state and therefore no visibly obvious method of passing information.

The first change, and the one which is most closely related to the design of the framework in the next chapter, is the ability to define a cellular automaton where each individual cell is defined by a different set of parameters. This is known as a hybrid system, and can be directly related to the vertical axis of the computing spectrum shown in Figure 1.1. Typically hybrid systems will be limited artificially to only use a few different sets of parameters, such as only changing the transition rules, or using only a few cells in 1D, each with different parameters. A simple example of this type of restriction is created when using the method for designing 1D linear hybrid cellular automata with a characteristic polynomial that is described in [10]. In this case, the cells are limited to only being able to change their transition rule compared to other cells, and can only choose from a restricted set of rules that enable the particular method.

The second change, which is directly used in the hardware application that is described in Chapter 5, is the ability for the state of the cells in a cellular automaton to not only be influenced by their local neighbouring cells, but also by the external environment. These are known as dissipative cellular automata, and can be simply described as systems where a form of external information can influence the states of the cells in the system in real time. A particular form of dissipative cellular automata are described in [15] which are also asynchronous: the cells update independently at arbitrary times. In this work, the ratio between how often cells are updated and how often the external environment affects the cells is used as a classifier to group various systems together. The goal of this work is to be able to artificially create a desired pattern globally by only imposing it on a small local set of cells.

Other than these changes, very intriguing work has also been done that combines the fields of nonlinear dynamics and computing theory into a new field called computational mechanics [34], which characterizes the patterns and structures that occur in natural processes through formal

computational models. Instead of taking the traditional approach in computing theory of finding a cellular automaton to perform a specific function, computational mechanics seeks to decompose the behaviours of a particular cellular automaton, or a group of them with similar parameters, into their core patterns using higher and higher levels of abstract descriptions [35]. These descriptions generally start with the following foundational levels: a background of one or more static or periodic patterns known as regular domains; the borders between these domains, which take the form of particles; and the interactions between these particles, which have the ability to perform useful computations using the particles' direction and speed as the data.

A foundation has been laid in this section of the core aspects of the field of cellular automata, along with subsections on both the history of cellular automata up to this point and some of the more recent work that has been influential in the development of this thesis research. Building on this foundation, the rest of this chapter describes the main extensions of cellular automata that are used in this research: hardware implementations of cellular automata in the next section, and the use of genetic algorithms to search for particular sets of parameters in Section 2.3.

2.2. Hardware Implementation

There are many different ideas of what actually constitutes a hardware implementation of a computational task, and these ideas are largely based on past experience and the particular field of computation. In this section, the focus will be on two forms of hardware implementations that are the most common and useful in terms of cellular automata: embodied and embedded systems. A comparison of these two types of hardware, and the key aspects that make them more useful than other hardware for cellular automata implementations, is discussed below.

Although there are a few conflicting definitions of embodied and embedded systems among the loosely connected fields of robotics, artificial intelligence, and cognitive research, the definitions that will be used throughout this dissertation are as follows. An *embedded system* is an assembly of hardware on which a computational task is carried out defined by low-level software, and using a much lower level of hardware architecture compared to modern day personal computers. *Embodied systems* are a subset of these embedded systems that also have a form of physically dependent representation and that can often sense and interact with their surroundings [36]. Both embedded and embodied methods of implementation introduce beneficial aspects in the design, development, and use of hardware in the field of cellular automata.

In the first subsection, a number of common forms of cellular automata hardware are described in general with a discussion of the benefits and drawbacks of their implementation on particular types of hardware. This mainly focuses on the embodied and embedded systems that have been discussed in general above. In Subsection 2.2.2, a selection of recent research is described, most of which takes advantage of a few of the benefits of cellular automata hardware implementations to get the most out of their use, either as a control system or an artificially intelligent machine.

2.2.1. Cellular Automata Implementation

The most common implementations of cellular automata generally consist of a program running on a form of serial computer, often using a parallel-core processor, that simulates and then displays the output of a cellular automaton on a monitor. This form of implementation is a prime example of a system that exists in the extreme lower left corner of the computing spectrum from Chapter 1. The next most common implementation method, which is sometimes labelled as a hardware solution, effectively just replaces the serial computer with a form of supervised parallel computer, whether it consists of multiple serial computers in a network or a specially designed parallel system using custom boards. Regardless of the specifics, this solution again is generally applied to the cellular automaton as a whole and resides in the lower left corner of the spectrum.

The first form of implementation that is definitely a hardware solution, a type of embedded system, is when a configurable piece of hardware (e.g., an FPGA board) is programmed using a specific set of cellular automaton parameters. This way, the hardware has been specifically designed to allow for the large numbers of direct connections necessary to perform the computation of that cellular automaton as quickly as possible. This, most often, also resides in the lower left corner of the spectrum. However, there are implementations where a configurable piece of hardware has been designed to act as a single cell as well, moving the system along to the lower right corner of the spectrum with a single processor for every cell in the system.

Within the scope of cellular automata, embodied hardware entails a form of hardware modularization that causes the computation of a set of cells or cellular automata to be spread among a number of spatially distributed, modularly connected systems, each with its own processor and local information storage. An embodied system of this kind can have a range of computational abilities from a few processors, each computing one or more large cellular automata, to vast numbers of very simple processors each computing the outcome for a single cell or small group of cells. The architecture of an embodied hardware system is very different

from the other methods generally used in the computation and display of cellular automata in a number of ways that are broadly introduced below.

Since most cellular automata are based around a synchronous update method, the main issue with their implementation on distributed hardware is the difficulty of keeping the various modules of the system synchronized. This can be very difficult at the upper extreme of distribution, where each cell is purposely blind to the rest of the system, without resorting to connecting all of the cells to a common signal. However, in a situation where individual cellular automata, or even just large groups of cells, are running on modular hardware as small, locally-synchronous systems that share boundaries with other modules, there are a number of different ways that the system as a whole can be updated asynchronously without losing any information. The simplest of these methods, one that was briefly considered in the hardware application in Chapter 5, would be to include a flag on each of the modules that let any connected neighbouring systems know when the module has finished updating and is waiting for data.

Building on these aspects of hardware implementations of cellular automata, a selection of recent research has been described in the following subsection. The projects discussed are a system that relies on the predictable nature of a subset of cellular automata to operate window shades on a building, a sensor-driven hierarchical cellular automaton that provides dynamic lighting, and an interactive wall that allows any number of custom systems to exist including the Game of Life.

2.2.2. Recent Work

There are a range of different forms of hardware-implemented cellular automata, even a small subset of which could fill an entire book. In light of this, a selection of specific hardware implementations are discussed in this subsection that deal with a broad range of the particular aspects of cellular automata that directly impact the design of any hardware. The examples below cover a set of applications that include 1D and 2D cellular automata, dissipative and hierarchical systems, interactivity and presence sensing, and modular distributed systems.

The first example of a hardware cellular automaton implementation is the use of the state progression of a 1D cellular automaton to control digital window shades [9]. Each row of the building's window shades displays the state of the cellular automaton at the time step after the row above it, starting from specific initial conditions in the top row that are seldom changed. This is done in a way similar to how 1D cellular automata are typically displayed, as a 2D image with time as the vertical dimension and starting with initial conditions at the top as shown in

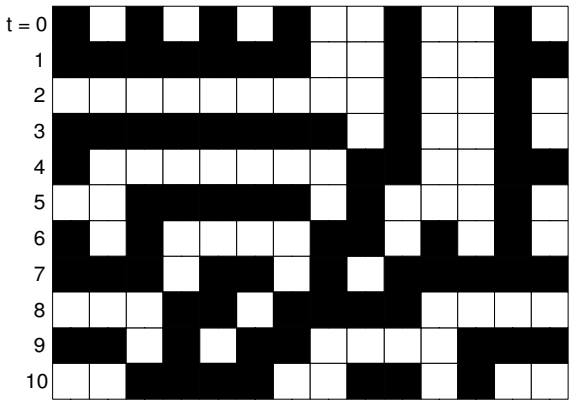

Figure 2.11. Illustration of the Elementary 'Rule 45' cellular automaton using periodic boundaries, with initial conditions in the top line and each progressive line showing the state in the following time step.

Figure 2.11. Since the computation has been implemented at the individual cell level, there would typically be the need to somehow synchronize the state changes of the cells using a global clock. However, in this case, since each row's behaviour is based solely on the states in the previous row and the states change infrequently, there is no need for synchronization across the cells. For example, if a cell takes longer than its neighbours to update, the cells below the slow cell may be in the wrong state temporarily. Once the slow cell updates to the correct state, the cells below it will also correct themselves. Since the initial conditions do not change on a regular basis, the states will remain static for large periods of time.

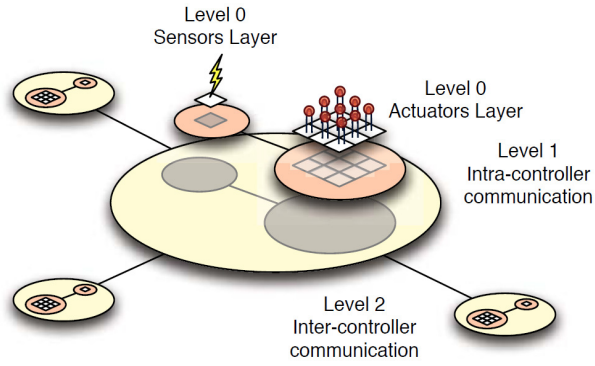

Figure 2.12. Organization of the hierarchical cellular automata in the second example. Note the multiple levels of systems involved. Image reproduced from [38].

Another example (shown in Figure 2.12) is in the use of a set of asynchronous, dissipative, hierarchical cellular automata implemented on custom embodied hardware that are used to provide artistic, sensor-driven lighting in a tunnel environment in [37] and [38], and later extended to be used as modular interactive lighting tiles for indoor environments in [39]. The first application makes use of a set of parameters that allows memory of past states, reaction to sensor inputs, and the evaporation of a cell's state to its neighbours to enable a diffusion of information throughout the system, represented as a smooth, dynamically changing lighting pattern. In the second application, these tiles are extended to be used as modular lighting elements in a home and is built on a similar framework and hierarchy as the previous application.

Touch-sensitive screens and embedded systems are sometimes joined to create a more specialized form of hardware that is on the verge of becoming embodied, as shown in this final example. To illustrate these types of systems, a modular system created by the Logic Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne (EPFL), known as the

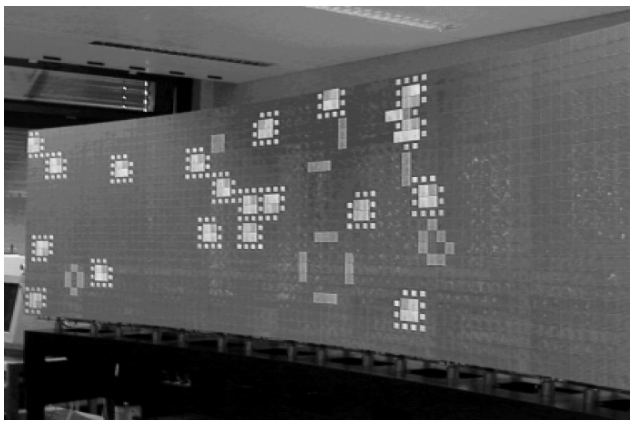
Figure 2.13. Picture of the BioWall in the Logic Systems Laboratory. Image reproduced from [40].

BioWall, is used. The BioWall is built from touch-sensitive modules that each contain an LCD and an FPGA that are specifically developed for a particular set of parameters. An application on this BioWall using a combination of the Game of Life parameters and a form of self-replicating blocks (shown in Figure 2.13) was developed in [40], and is used as a demonstration of a dynamically hybrid application. In this implementation, the Game of Life is applied as normal except for when a square block of four cells are created. When this happens, the block of cells and their neighbours adopt new transition rules and a new neighbourhood shape which, in combination, support the self-replication of the block based on input from touch sensors in each cell. This application of new rules within an environment of other rules can be considered as a slightly more complex form of a hybrid system than those with static but different rules in each cell.

This overall section has provided some background on hardware implementations, including definitions of embedded and embodied systems, and how cellular automata are impacted by the use of hardware in their representation. This specific subsection has also described some specific examples of hardware applications of cellular automata, with a wide range of different types of systems used to demonstrate a subset of the possibilities that exist. Next, a particular form of evolutionary computing known as genetic algorithms are explored and described, both in general and with a focus on their implementation in searching for cellular automata parameters.

2.3. Genetic Algorithms

Genetic algorithms are a type of evolutionary algorithm within evolutionary computing, a sub-field of computational intelligence based mainly on the ideas and concepts of biological evolution. They have been developed as a search method for large or non-linear search spaces, and work especially well compared to other search methods when there are many local minima [41]. Like other evolutionary algorithms, they apply the mechanisms of evolution (reproduction, mutation, crossover and selection) to search a population of candidates for the “fittest” individual(s) of a given generation. In the case of genetic algorithms, the population is made up

of groups of *genes*, called *chromosomes*, which directly or indirectly represent the controllable parameters of a system. These genes can be very detailed and specify every individual aspect of a system or can be very broad and provide only a high-level of control. The actual method of how genetic algorithms work is described in the next subsection, with details of how the various evolutionary mechanisms are applied to the chromosomes.

There is a lot of existing work in the combination of genetic algorithms and cellular automata, mainly because of the size and non-linearity of the parameter search space and the near-impossibility of a manual search. Subsection 2.3.2 provides a general foundation on this combination, and mainly focuses on a discussion of how a cellular automaton can be represented as a chromosome using various genes to directly modify the system parameters such as transition rules or neighbourhood size. This is followed by a description of a selection of recent work on the combination of cellular automata and genetic algorithms in Subsection 2.3.3.

2.3.1. Genetic Algorithm Structure

A population of these chromosomes are created at random throughout the search space, sometimes distributed in previously determined high-potential areas of the search space, and are evaluated in terms of their 'fitness' in satisfying the goal of the search. This fitness is calculated for each chromosome using a fitness function, and is used as a quantitative rating system of how well the given chromosome does at satisfying a desired goal. This goal could be computing the outcome of an algorithm, finding the best performer among a group of competing behaviours, or even just locating the global maximum of a function.

Once the best chromosomes are determined in a given generation using the fitness function, they are selected as the parents of future generations. There are many different ways of choosing how to pick the 'best' chromosomes, with the most common methods being fitness above a certain threshold or in the top 10% of all members of that generation. Any chromosomes that are not chosen as parents are generally eliminated from the population to allow only the fittest to survive. This shows that the parameters specified by the genes and the definition of the fitness function are key elements to effectively applying genetic algorithms.

After each generation, two parents are selected and combined to create new chromosomes using a method called 'crossover', in which a subset of the genes of each parent are switched with each other. Typically the point at which crossover happens is chosen at random, but there are cases where it benefits the search to have the crossover occur at a specific point in the chromosomes

such as between two specific genes. Once these new chromosomes are created, each of their genes has a small probability that it will change to a different value, to mimic mutation.

The algorithm will continue indefinitely going through the cycle of finding the fitness, selecting parents, abandoning the unfit, and creating children until either a set number of generations or until there is a chromosome created that satisfies the fitness function within a specified tolerance. In many cases, there is no optimal goal or the task may have no solution so there needs to be a way of ending the evolutionary cycle. Generally this is done by keeping track of the best member in each generation and stopping if it has been the best for a set number of generations.

2.3.2. Evolving Cellular Automata

In the world of cellular automata, there are vast numbers of possible sets of parameters, which create a huge search space when attempting to produce or find a system that will solve some global task. This is further complicated by the fact that the search space is not locally similar, with minor changes in any of the parameters leading to anything from no change at all to a completely different set of behaviours. Due to these issues, the most promising automated search method of sifting through all of the possibilities of useful or promising sets of cellular automaton parameters seems to be through the use of genetic algorithms.

Not only do genetic algorithms provide the most promising search method for finding a system with a specific global behaviour, the methods of evolution and gene specification can also help to classify groups of cellular automata. How the mutation and crossover of genes happens can lead to a number of clustering mechanisms as a direct consequence with even a very broad fitness function. The combination of a few specific gene sequences may occur in all high fitness members of the population, allowing a more direct investigation into the nature of all cellular automata that fit the parameters. How the genes are built, and with which parameters, can also lead to grouping and classification in a similar fashion to that of the evolutionary changes. Other than a few specific parameters that define how the selection, crossover and mutation aspects will occur, the main considerations involved with implementing a genetic algorithm on a given task are how to choose the genetic attributes and what the fitness function will be measuring.

Due to the variety of different types of cellular automata, there are a range of different potential genetic attributes that can be used to find the solution to a given problem. The tessellation used can be included, as well as the size and shape of the neighbourhood. Boundary conditions and transition rules are both used, and are typically kept the same for the whole cellular automaton,

although sometimes hybrid systems are used. Finally, the initial density of particular cell states is used when the task is not dependent on initial conditions. The transition rules and initial conditions are the most popular parameters to be used as genetic attributes, as most of the rest of the parameters are rarely used due to the types of tasks and libraries that are currently available.

In terms of how the fitness function is specified with relation to cellular automata, generally there are two different options, although there are rare cases where other options present themselves as part of a specific task. The first popular option is to examine the state of either all or a subset of cells after a specific number of time steps have passed to determine if a task-dependent condition has been met. This often takes the form of all of the selected cells being in a particular state, or using pattern recognition to find if a specific goal pattern has been reached. The other popular option is generally more difficult to compute, and is based on the steady state of the cells given that the initial state was a part of the task. Typically this will involve keeping track of whether or not any cells have changed in a given time step and continuing to progress through time until the cells have all reached a steady state. This option is usually used when trying to find a parameter set that implements a classification system for the initial conditions.

There are two very common tasks that are used to demonstrate and validate the performance of any cellular automaton library with genetic algorithms, both typically applied in 1D: the density and synchronization tasks. The density task attempts to find a rule in a two state cellular automaton that will result in all of the cells taking on, after a given number of time steps, the state that was initially the majority state. The synchronization task attempts to find a rule in a similar system that will result in an oscillation of the states of all of the cells in the system from one homogeneous state to the other, given any possible initial conditions.

Although there has been some use of human selection in genetic algorithms, a subset of interactive evolutionary computing [42], none of this work has been done in relation to cellular automata. The use of a human-in-the-loop fitness function, where a person chooses the fitness of the members of a population, is the main method of using people in genetic algorithms, although there are other applications where people are used to create the new population using a given selection of parents. In the next subsection, recent work on the combination of cellular automata with genetic algorithms is discussed that includes research using both of these tasks, as well as some other work using a number of more specialized tasks that are very problem-specific.

2.3.3. Recent Work

Genetic algorithms are frequently used in academia to attempt to find solutions to a wide range of problems in the field of cellular automata. Again, since this topic is so broad, only a small but broad selection of research is discussed in this subsection to allow an introduction to the field. There are a number of different groups that have worked with genetic algorithms in the field of cellular automata, and a few of these groups are discussed below along with their contributions.

The first group to be discussed is a group of researchers involved in the Evolving Cellular Automata Project, which was based at the Sante Fe Institute and at the Los Alamos National Laboratory. Although the project is no longer active, the group's contributions to the evolution of cellular automata are extensive and are described fairly exhaustively, along with the addition of computational mechanics to genetic algorithms, in [43]. To briefly summarize, Mitchell and Crutchfield, along with a number of other students and researchers, evolved a number of different rules for solving both the density task and the synchronization task in 1D. Once found, these solutions were analyzed in both traditional fashion and using computational mechanics to discover how the evolutionary process was finding the best candidates for each task. Building on some of this group's earlier work in genetic algorithms ([44]..[55]), some solutions to the density task were also found by extending similar methods into 2D systems by Inverso et al [56].

The next group to perform work in the use of genetic algorithms with cellular automata were Breukelaar and Bäck, at the Leiden Institute of Advanced Computer Science ([57]..[60]). Their main work is in the development of an improved approach to finding transition rules for 2D cellular automata using genetic algorithms. In [60], solutions are found for the density task, which they call the 'majority problem', a task where a chequerboard is created from any initial conditions, called the 'chequerboard problem', and a more general task where a rule is desired to change any initial conditions to a specific pattern of states, called the 'bitmap problem'.

Finally, the most recent group to work on the combination of genetic algorithms with cellular automata are Sapin, Bull, and Adamatzky at the University of the West of England, as explained in [61] and [62]. Their work is focused on multiple searches within an increasingly restricted set of rules that initially support any one of a number of specific 'glider' patterns. The initial search is used to discover a set of patterns that produce these 'gliders' at regular intervals, known as 'glider guns'. Once a set of 'glider guns' are found, another search is done of only the rules that support them to discover how to use these 'guns' to build basic logic gates, to find parameters that

support universal computation. The authors outline how they designed the genetic algorithms overall, as well as specifically how their different fitness functions were developed.

This overall section has introduced general methods and applications of genetic algorithms, as well as explaining how they are used with cellular automata. This specific subsection then provided a few examples of different applications of genetic algorithms in concert with cellular automata, including the search for solutions that directly benefit the field. Overall, this chapter has provided a background in cellular automata, and their combination with hardware and genetic algorithms, which will serve as a solid foundation on which the remainder of this dissertation is built. In the next chapter, the interconnection framework, the main contribution of this thesis research, is discussed and its relationships to the various elements explored in this chapter are described.

3. Framework

In this research, a framework is developed that defines a new method of organization for a group of cellular automata, and indirectly simplifies their implementation in software and hardware. This new method of organization has been labelled *interconnection* and is used to enable the design of cellular automata throughout the computing spectrum that was introduced in Chapter 1. In this chapter, the novel concept of interconnection is explored and extended into a set of definitions that will allow the creation at the high level of an interconnected set of simple systems which include cellular automata and their boundary conditions. This concept is initially explored in terms of the specifics of how to connect a few cellular automata together, followed by examples of more complex interconnected systems, and ending with a more formal definition of the overall framework using graph theory.

Using the foundation in the field of cellular automata that was laid in the previous chapter, an analogy can be made to introduce the core concept of this framework. In the same way that a specific cell is partially defined by its position and connections within a particular cellular automaton, a specific cellular automaton can be defined in part by its position and connections within a particular interconnected system. A visual illustration of a simple interconnected system example, made up of a square interconnected grid of cellular automata, is shown in Figure 3.1.

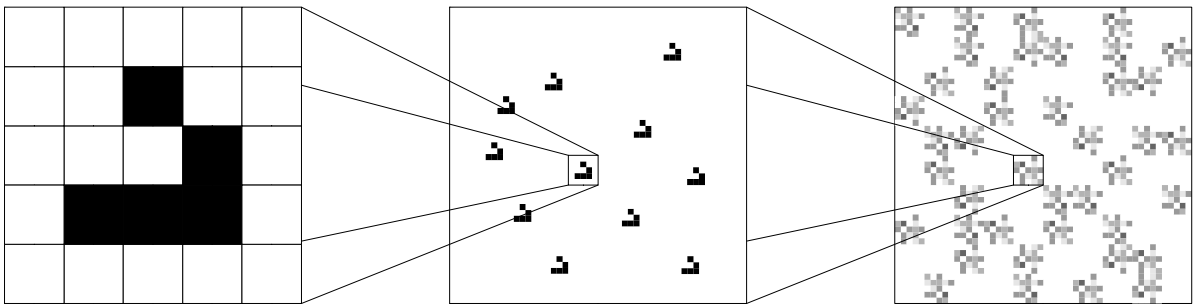

Figure 3.1. Illustration of an extension of cellular automata concepts to demonstrate an example of an interconnected system. On the left is a subset of cells from within the cellular automaton in the centre. This cellular automaton itself is only a single member of a set of interconnected cellular automata in a rectangular grid pattern, shown on the right.

In the following section, the concept of interconnection is explained in more detail and used to introduce a number of different types of interconnected systems that are designed specifically with cellular automata in mind. The methods used to perform these interconnections are described, including a comparison with similarly implemented methods in the field of cellular automata that were discussed in the previous chapter. Once these methods are explained, a

number of different systems of interconnected cellular automata are introduced, each of which enables a different form of exploration of the computing spectrum. Three specific forms of exploration are demonstrated using an example of a complex interconnected system to show how they will interact with each other.

With this basic understanding of the concepts in place, Section 3.2 extends the usefulness of interconnection to a larger set of systems by defining the core components of a graph known in this work as an interconnection graph. Briefly, the edges of an interconnection graph represent the directed transfer of information from one node to another, while the nodes can be any systems which are consuming or producing information. Although the nodes used in this dissertation are limited to cellular automata and some simple information producers, they are only a small subset of the possible systems that can exist in this graph framework. Using only this limited set of systems within the framework, a wide range of interconnected systems can be specified, designed and analyzed based on well-known concepts in the field of graph theory.

Once the concept of interconnection has been extended into the interconnection graph framework and some demonstrative examples presented, a library is described in the next chapter that implements this framework for a subset of the possible nodes. This library includes a number of aspects dedicated to interconnection, such as a set of methods for automatically interconnecting grids of cellular automata, and novel aspects that provide ancillary benefits to an interconnected system, including a mechanism for skipping updates for static regions of cells.

3.1. Interconnection

The main innovation explored in this thesis research is interconnection: the ability to connect the boundaries of a set of cellular automata. These interconnections are effectively an extension of the methods used in hardware implementations of large cellular automata to divide their storage requirements across multiple memory modules. In essence, each interconnection acts as a two-way dynamic information transfer point, where the states of the cells on the edges of each system are accessible to the edge cells of the connected system. This can be explained directly using an assumption that the two systems are combined into one larger cellular automaton, ignoring any differences in parameters, so that the edges of both systems are adjacent. If the neighbourhood of a given edge cell would include the edge cell(s) of the connected system under this assumption, then that given cell has direct access to the state(s) of those connected edge cell(s).

Providing this access to the edge cells of an interconnected system is accomplished by extending the methods normally used for periodic boundary conditions, where two opposing edges of a single system are connected, to allow the edges of two different cellular automata to be connected. These interconnections are made, using similar logic, between each cellular automaton and its neighbours as dictated by the overall system design. Therefore, in a fully interconnected system, each of the edges of a particular cellular automaton is connected to a specific edge of another. These interconnections can exist on as many sides as is necessary to build the overall map of cellular automata that is desired. Note that the shape of the cell matrix of a specific cellular automaton may be completely different from the shape of each individual cell.

The concepts behind interconnection have only been explored up to this point in the realm of hardware [8], and have generally been used simply as an abstraction of the arrays that are used in software implementations instead of being developed as a feature in their own right. With the addition of interconnection directly within the set of design parameters as part of this thesis research, the framework introduces the possibility to search for, design, and simulate cellular automata solutions comprising multiple, interconnected systems. Thanks to this concept, a number of interesting system designs can be investigated, including a few specific designs that are discussed below. The three specific designs that are discussed can be effectively labelled as hybrid sets, spatially abstracted sets, and sets with mixed dimensions.

First, within an interconnected system there can exist any number of individual cellular automata that can each use a different set of parameters (transition rules, neighbourhood, etc.) than those around it in the system to create a hybrid set. Unlike other interconnected systems, no additional overhead is needed to implement a system, other than what is needed for periodic boundaries, provided that there is a specified method of sharing edge states between the various different types of cellular automata. For the majority of these parameter differences, the edge-sharing method is simply based on how many cells the neighbourhoods for each cellular automaton need to have access to in the connected system. In the case where the two systems use different cell shapes, there are further definitions needed in terms of how the edge neighbourhoods are organized when some of the cells are different shapes. Regardless of the content of these definitions, only simple extensions should be needed to allow any potential combination that is desired. This creates the ability to design a system that can exist anywhere along the entire range of hybridity. An example of a complex hybrid system is shown in Figure 3.2, where the colours

of the cellular automata each represent a different set of parameters and the shapes of the cellular automata represent the various overall cell matrix shapes in 2D and 3D.

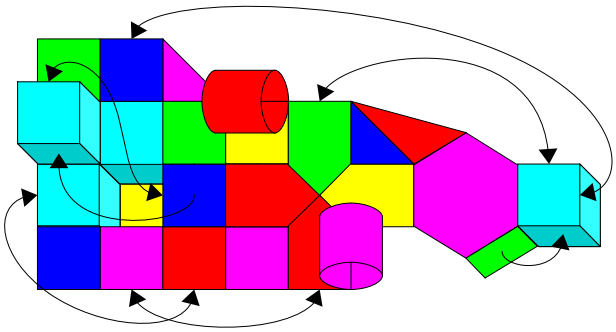

Figure 3.2. Example of a complex interconnected set of cellular automata. The colours represent different sets of parameters, while the arrows represent shared edges.

The second specific system design uses the fact that a particular cellular automaton can share anywhere from one to all of its edges with others which are not necessarily in its spatial neighbourhood, as shown by the arrows in Figure 3.2. Or, as an alternative, that there is the ability for a cellular automaton to connect one set of opposing edges into a cylindrical wrapping while the other edges can be interconnected to other systems or given more traditional boundary conditions. In this way, a number of unique system designs can exist where a cellular automaton can affect the cell states on the edge of another, without affecting the states of any other interconnected systems. It is important to note that in a case where an individual system does not share any of its edges, it is isolated and acts as an independent cellular automaton. These systems can broadly be labelled as spatially abstracted sets of cellular automata, as the cells can act as though they are spread across a range of surfaces and shapes that are impossible to physically recreate without moving into higher dimensions of space. In Figure 3.2, one of these impossible systems is shown as there is no physical way to spatially attach all of the interconnected edges along the arrows and maintain any useful spatial shape or surface.

The final system design to be described is one where the dimensions of the various cellular automata involved are different, the third visible aspect of the system shown in Figure 3.2. For example, the use of the entire system state of a 1D cellular automaton as one or more boundary conditions of a 2D system, since each edge in 2D is effectively the same structure as an entire 1D system. Since there are a number of very simply defined 1D cellular automata that exhibit very complex behaviour, their use as boundary conditions has the potential to produce very complex, computationally useful patterns in 2D. This can be viewed as a simple method of inputting algorithms that could potentially find solutions to a number of difficult tasks within a given system. As an analogy, consider that a wind tunnel is effectively a 3D visualization method for a 2D input. Note that this can be extended to using any (n-1)-dimensional system as the boundary conditions for an n-dimensional cellular automaton.

To create a solid framework that describes how to define interconnections consistently, a set of graph components have been developed and discussed in the next section. Using this graph framework, not only can sets of cellular automata be interconnected, but all of the possible boundary conditions can be specified. With each individual system or information source defined as a node, and each directed edge connection defining the path of information flow from one node to another, this graph can be used as a tool for the analysis and design of a wide range of interconnected systems.

3.2. Interconnection Graph

The previous section outlined the concepts behind interconnection, some details of how it works for cellular automata, and a number of examples of the types of systems that it enables. This section elaborates on a graph framework for defining any interconnected system that includes cellular automata, and other information sources, as an interconnected graph. As with any graph framework, there are two core components that need to be formally defined, nodes and edges, as well as the methods of connecting these components. In this framework, any system that can act as a producer and/or consumer of information is a node, while the flow of information between nodes is defined using the edges. The framework does not restrict the class(es) of information that flow(s) along the edges (e.g., discrete/continuous, scalar/vector, etc.). How these edges are defined and used to interconnect nodes is discussed below, followed by which types of systems can be defined as nodes, and ending with examples of a few nodes used later in this dissertation.

Within an interconnection graph, each edge shows the directed flow of information from a producer to a consumer. Each node can have multiple connection points on a given side, each of which may be a producer or consumer of a different class of information. Therefore, the edges of these graphs define how each specific class of information flows throughout the system, and between which particular nodes' connection points it is flowing. When two nodes each have a side with a pair of producer and consumer connection points that both use the same class of information and are connected together in both directions, a bidirectional edge can be used to clearly show this relationship and simplify the graph. This and other connection methods are explored with an example later in this section.

The nodes in these graphs are individual systems, which must manipulate and use information in a few very specific ways that are described below. Any given node can have multiple connection

points, each of which can use different classes of information and can be either a producer or consumer of that information. In some instances, such as with the edges of a cellular automaton, a particular type of node will require that a few of its connection points must consume information from a producer for the system to continue to function. This means that every node's required consumer connection points must be properly sourced with information by a suitably matched producer, or there is a chance of complete system failure. Although there is only a chance of failure, due to the robustness to information loss of unsupervised parallel systems, this requirement is applied in the graphs in general to ensure system stability. Note that there is no way that a producer connection point can require information be pulled from it, nor can there be too many consumers sourcing information from one particular producer since in this framework a producer point can always be infinitely copied.

For a particular system to be used as a node within an interconnection graph, it must satisfy a number of requirements. First, the system must have a mechanism for the transfer of information, whether that is as a producer or a consumer (or both), which dictates the role of each connection point on the system's node. Second, the format and content of the information that is being produced or consumed by the system must be the same as (and connected to) at least one other node in the graph that has the opposite role, to prevent the existence of unconnected nodes. Finally, the system must be able to perform its information transfer in one time step of the overall system, or at least to continue to transfer the same information repeatedly until new information is generated. These requirements are designed to ensure that all of the nodes in a graph can function and contribute to the overall system.

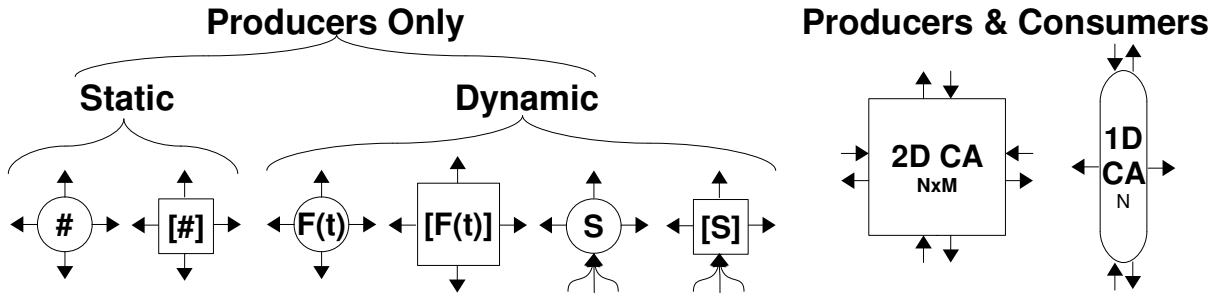

Figure 3.3. Examples of various types of possible nodes in the interconnection graph. Arrows pointing out of a node are producer connection points while those pointing into a node are consumers. Tripled arrows indicate external input. Rounded node edges signify a scalar connection point, while flat edges signify a vector connection point.

Given this set of requirements, there are a vast number of systems that can be used as nodes limited only by the desired complexity in the overall system, a subset of which are shown in

Figure 3.3. The only consumers of information that have been shown are the cellular automaton nodes, or CA nodes, as they are the primary system that is investigated in this thesis research using the interconnection graph. The other three types of nodes that that are shown in Figure 3.3 are all strictly producers of information, and each has both a scalar (circle) and vector (square) form. These nodes are labelled as follows: # for static nodes, F(t) for dynamic nodes, and S for sensor nodes. These node types are described below, but are only a subset of possible nodes.

In drawing these nodes, three forms of standard notational elements have been used. First, arrows that leave a node's edge are producer connection points for that edge while arrows that point towards a node's edge are that edge's consumer points. Second, a triple set of arrows pointing at a node but connected from nothing indicate that some form of external information is being used. Third, a rounded edge on a node indicates that a scalar value is transferred on any connections on that edge, while a straight edge indicates the same for a vector of values of a specific length. Any scalar connection point is capable of producing information for any consumer in the system, provided that the value produced is within the acceptable range for that consumer. Since only one scalar value is being produced by the point at a particular time regardless of the node type, that value is repeated to create a set of values of whatever size is necessary to connect to a particular vector consumer point length.

In a cellular automaton node, the number of connection points is entirely dependent on the shape and size of the cell matrix as it can vary widely from one system to the next. Also, the format of the information being produced and consumed on a given edge is always one dimension smaller than the cellular automaton itself, as it must match the size of the corresponding edge of the cell matrix. For example, a 2D cellular automaton produces a 1D array at each edge. Due to these complications, if the cell matrix is an odd shape or has different numbers of cells on particular edges, then a number of different information formats (and connection points) will be necessary.

Although each of the connection points on a cellular automaton node acts as a consumer and must take in information from a producer (as boundary conditions), not all of its points must also be active producers. This is due to a requirement in cellular automata of knowing how the neighbourhoods of any edge cells are defined to progress to the states in the next time step. However, there is no requirement that another system be monitoring the states of the edge cells of a cellular automaton. Since the connection points of a cellular automaton node can be connected to the points on the opposite side of the same node, a single system with periodic

boundary conditions in all directions can be specified using only a single one of these nodes and a pair of bidirectional edges as shown in Figure 3.4a.

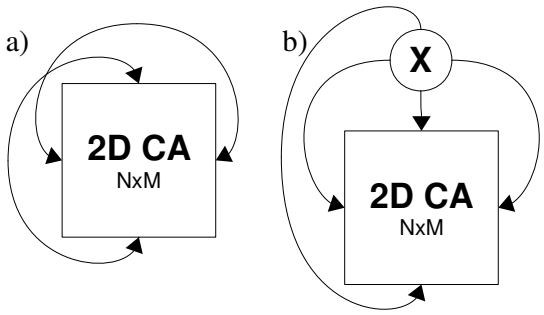

Figure 3.4. Interconnection graphs for the two main types of 2D cellular automaton that are currently used in academia: a) a system with periodic boundaries, b) a system with static homogeneous boundaries.

A static scalar node is the simplest type of node available in this framework, as it has a constant value and can produce information for any consumer in the system. The static vector nodes are only slightly more complex, as they still have a constant set of values but can only produce information for consumers of the same size as themselves. Using cellular automaton nodes and static nodes, it is possible to specify every classical set of cellular automaton parameters that has been used up until the development of this framework. For instance, to define a cellular automaton where all boundary conditions are static, homogeneous and identical, only two nodes are necessary: the cellular automaton node, and a static scalar node. The interconnection graph (shown in Figure 3.4b) is complete when all of the consumer connection points of the cellular automaton node have been connected from the static scalar node's producer points. Note in this graph that the scalar producer node is producing two vectors, of sizes *N* and *M*, for four consumer points.

The dynamic and sensor nodes are usable in a similar way as the static nodes in terms of how their connections are made with consumer points. However, how the values that are passed along to those consumers are produced is very different. In dynamic nodes, the values are based on the output of some form of function that changes over time, and may take into account as a part of the function, any internal part of the interconnected system. This differs from the sensor nodes due to the information being internal to the graph, whereas in a sensor node at least some of the information used comes from an external source. These three types of producer nodes are some of the most general types of nodes that can be defined, as they can inherently represent any system that does not require direct input from other parts of the interconnected system but can inject information into it. The applications of using nodes such as these are explored as a part of implementing this framework in the next chapter.

The graph for an example of an interconnected system is shown in Figure 3.5 that makes use of an assortment of the node types and producer/consumer pairings that have been described. There

are a few interesting connection methods in this particular example that require further description. The bidirectional connection between opposing sides of a cellular automaton node represents the wrapping of those edges, which is exactly the same as the periodic boundary conditions as described earlier and used in Figure 3.4a. This can be seen connecting the top and bottom edges of the left-most 2D cellular automaton node, making it act as though it were wrapped around a horizontal cylinder. The connection of the producer to the consumer on the same edge of a cellular automaton node, as shown at the top of the middle 2D cellular automaton node, can be thought of as a mirror condition: the cells on that edge will see copies of their own state and neighbours' states as if looking in a mirror.

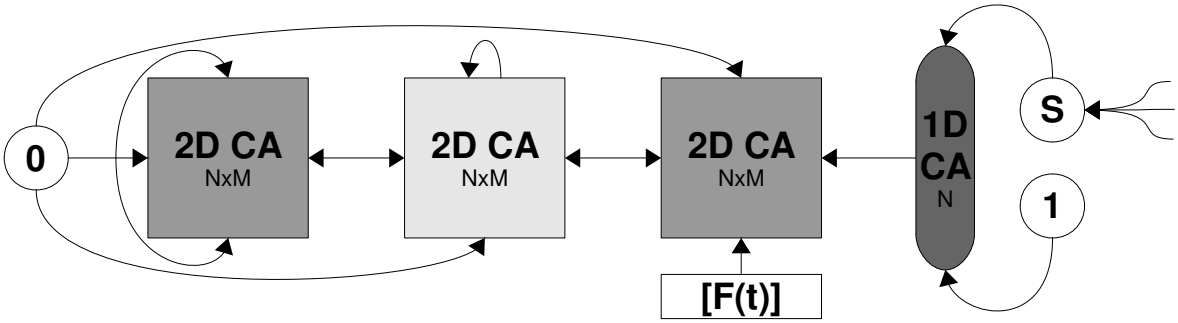

Figure 3.5. An example of an interconnected system drawn using the graph framework. In this example the shade of a particular CA node represents a specific set of system parameters. This means that the left- and right-most 2D CA nodes both have the same set of parameters, while the rest of the CA nodes each have different sets of parameters.

Note that this system also uses two of the specific design aspects from the previous section: hybrid sets of parameters and mixed dimensions of cellular automata. In this graph, hybrid sets of parameters are represented by the shades of the nodes, and the left- and right-most 2D cellular automaton nodes share the same parameters, as indicated by having the same shade. The mixture of dimensions is particularly evident where the right-most 2D cellular automaton uses the output of the 1D node as its boundary condition, requiring the size of the 1D system to be the same as the vertical dimension of the 2D system. It is fairly simple to confirm that every consumer point is connected to a corresponding producer, making this a valid interconnection graph, in addition to a few of the cellular automaton node producer points remaining unused.

In this chapter, a framework for the interconnection of cellular automata and other simple systems has been defined. This framework also simplifies the implementation of cellular automata in both software and hardware by allowing modularization and hybridity to span the full range of the computing spectrum. The core concept of this framework, interconnection, has been introduced and explained with a number of examples, and the graph components needed to

define interconnected systems and facilitate their design have been developed. The specifics of how a graph in this framework is constructed and built have been discussed, a number of examples of nodes within these graphs are given, and an example of an interconnected system has been described using these nodes. To clearly demonstrate the use of this framework, as well as to successfully accomplish the overall library design criteria for this dissertation, a library has been developed that is described in the next chapter. This library not only implements the graph framework from this chapter but also a number of other novel aspects that directly serve to achieve the criteria as set out in Section 1.2. Following the description of this library, an application that makes use of the library is discussed in Chapter 5 which includes the use of an interconnection graph to describe the overall system.

4. Implementation: Library

Informed and influenced by the architecture of the interconnection framework as described previously, a library for cellular automata has been designed. This library was also designed as a prototyping and simulation environment to aid in the development and design of a hardware application of cellular automata, which will be described in detail in Chapter 5. After a brief discussion of general implementation details, there are four sections dedicated to specific aspects of the library: implementation of interconnection and other novel aspects in Section 4.1, one specific instantiation of this library architecture in software in Section 4.2, computational results of the use of these novel concepts in this specific software instantiation of the library in Section 4.3, and finally a discussion of a few of the library's applications in Section 4.4.

This library has also been designed to satisfy all of the criteria as set out in Section 1.2. These can be summarized as having a unified objective of fulfilling the need for a high-level system that implements the interconnection framework, can be instantiated on both hardware and software, has an open and extendable architecture, and will fully implement the standard set of parameters in the field of cellular automata. To enable full implementation of the standard parameters, there exists in this library the ability to create a single large cellular automaton and manipulate any of the standard system parameters that are desired. This ensures that the ability to execute any tasks designed with other systems in mind is maintained.

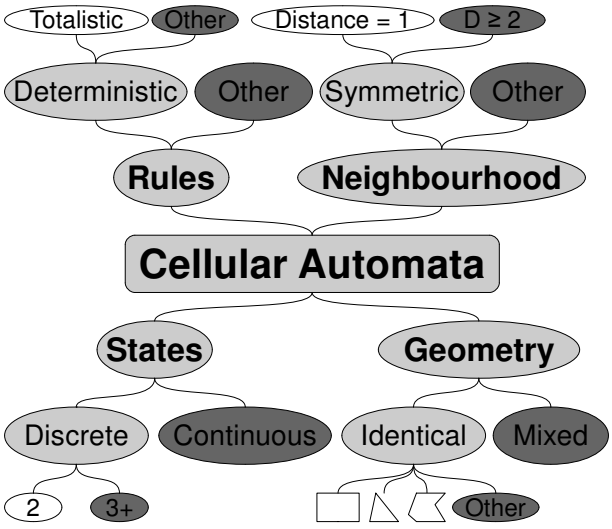

Figure 4.1. Concept map of the limited set of system parameters that are available within the library. White nodes are usable parameters, while dark grey nodes are not. Light grey nodes indicate partial implementation.

Nonetheless, there are a number of parameters of this library that have been artificially limited for various organizational and computational reasons that are discussed subsequently, as shown in Figure 4.1 (a reproduction of Figure 2.9 for ease of reading). For the sake of scope limiting, this library has been designed using inner-dependent, outer-totalistic transition rules with two states per cell. This allows both a wide range of possible rules and an intrinsic method of transcribing the rules in a simple, human-readable format as defined in Section 2.1. It is

important to highlight that any two-state outer-totalistic, or even general totalistic, rule can still be specified as they are both subsets of the inner-dependent, outer-totalistic rule set. How these other forms of totalistic rules are converted into inner-dependent outer-totalistic rules is discussed in detail in the next paragraph. By using the broadest definition of totalistic rules, all forms of totalistic rules can be used which allows the library to maintain the maximum functionality without losing simplicity.

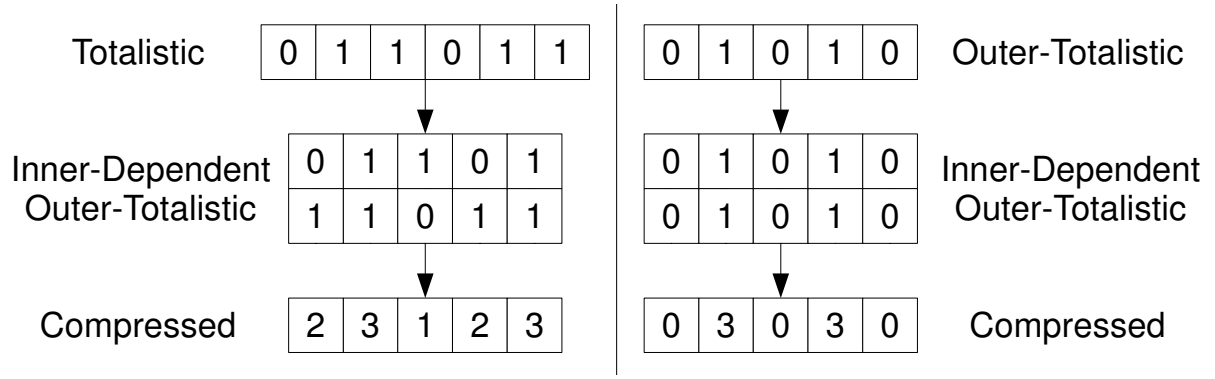

Figure 4.2. Conversions from sample totalistic and outer-totalistic rules into compressed inner-dependent, outer-totalistic rules. Both of these rules are valid for any two-state cellular automaton with five cells in a neighbourhood. Note that a 'five cell neighbourhood' is the equivalent of 'a cell with four neighbours'. Totalistic rules use the entire neighbourhood (from 0 to 5 cells in the '1' state), while outer-totalistic rules use the neighbours only (0 to 4 cells).

Examples of how these conversions work from other types of totalistic rules to the corresponding inner-dependent outer-totalistic rule are shown in Figure 4.2 (using the format introduced in Figure 2.6) for a two-state cellular automaton with five cells in its neighbourhood. To implement general totalistic rules the simple look-up table for the rules must be created, as shown on the left side of Figure 4.2. Given this simple look-up table, each element of the inner-dependent outer-totalistic rule matrix will have the same value as the element in the table that corresponds to the sum of both the inner cell index and the outer total index. To remove the inner-dependence of a rule but maintain the outer-totalistic behaviour, the look-up tables for each of the possible inner cell states must be equal as shown on the right side of Figure 4.2.

To avoid complex neighbourhood mechanics, and retain the original cellular automata concept of simple identical cells, the possible cell shapes were limited to only homogeneous and infinitely tessellating (covers the plane) shapes. These requirements reduce the potential cell shapes that can be used in 2D down to just three: rectangular, triangular, and a subset of hexagonal forms. The subset of hexagonal shapes requires that at least two opposing sides of the hexagon are parallel and congruent. If these restrictions were not in place, then any number of other potential shapes could be chosen where each cell might have a different neighbourhood depending on its

location within the tessellation, as discussed in Section 2.1 when using Penrose tiles and all but one pentagonal form. These tilings, as well as the one pentagonal tiling that does have a single neighbourhood mapping, also require an extension to this library's underlying cell organization to work. Though all of these tilings have been shown to support modified forms of cellular automata [14], they were not included in this library. It can be noted that 1D cellular automata are a subset of 2D systems, and can be implemented using 2D systems by setting static 0-state boundary conditions on the right and left edges of a single-cell width 2D system. However, due to their use of fully specified rules, the majority of 1D cellular automata cannot currently be defined in this library. This is not seen as an issue as their relative simplicity and limited single-cell edge effects mean that they are not the target of this thesis research.

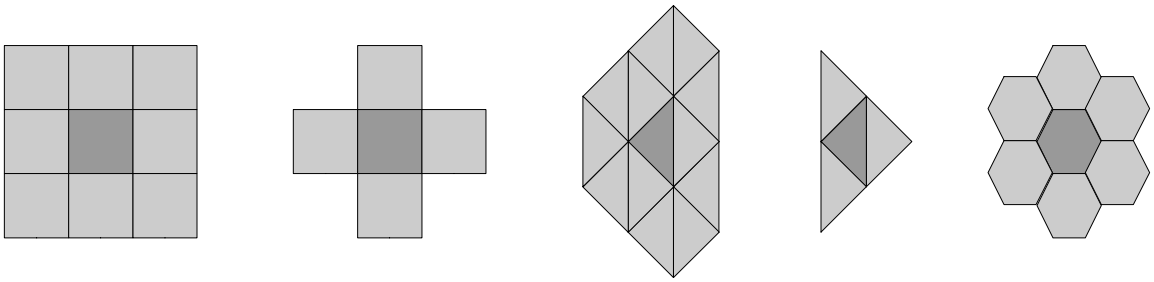

Figure 4.3. The five types of neighbourhoods available in this library. The light grey cells are always considered in the same way in the three types of totalistic rules. However, the dark grey cells are considered differently: in general totalistic rules, dark grey cells are identical to light grey cells; in outer-totalistic rules, they are ignored; and in inner-dependent outer-totalistic rules, they dictate which set of outer-totalistic rules to use on the other cells.

Building on these three regular tessellations, only the Moore and von Neumann neighbourhoods have been included in this design. They are the two best known and widely used types of neighbourhoods, and are available across all three of the available shapes within this library. In addition to limiting the neighbourhood types to these two popular options, the size of the neighbourhoods has been restricted to only including neighbours that are one cell away from the central cell, so that only the directly orthogonal and radial neighbours of the central cell are involved. As is evident from Figure 4.3, this still allows five different neighbourhoods to be used across the three different cell shapes and three different types of totalistic transition rules. It is interesting to note that the von Neumann and Moore neighbourhoods in the hexagonal tessellation with neighbours within a distance of one cell are exactly the same neighbourhood due to the lack of any corner-connections in a hexagonal environment.

One of the extended benefits of using the limited set of system parameters as discussed above is the ability to build upon as much of the previous research in 2D systems as possible while still

limiting the scope of this library to a manageable size to maintain simplicity. The majority of the cellular automata that are discussed and investigated in Chapter 2 are 1D, and those that are not tend to deal with very specific 2D systems such as Conway's Game of Life and its relatives known as the “Life-Like” rules [14]. There is very little work on the broad exploration of general 2D cellular automata, and effectively none of that little work uses non-square lattices. Of the work that does exist, the typical focus seems to be on the universal computation and self-replicating behaviours that first started the field (such as in [60] and [63]).

The ability to build on the existing 2D research is mainly due to the ability to simulate Game of Life and Life-Like rules, almost all of which can be specified using the limited set of system parameters available in the current library (shown in Figure 4.1). The only Life-related research that is not accessible while using the current library limitations is on systems that use complex neighbourhoods, three or more states, or non-totalistic rules, all of which can be included if necessary with fairly simple extensions to the library architecture. This ability to create specific extensions for a given problem demonstrates an inherent achievement of the extendable architecture desired, which allows the core components of the cellular automata library to be faster than typical generalized libraries without sacrificing the ability to generalize if needed.

Built upon this core subset of system parameters, there are a number of novel aspects that have been developed for use in this library that are discussed further in the next section. Many improvements have been made available within the design of this library compared to other known cellular automata libraries, such as [12] and [64], including a detailed specification of boundary conditions and the ability to easily extend the library to accommodate many different forms of research. The two primary improvements that have had the most impact on the success of this library are the implementation of interconnection, based on the framework from the last chapter, and the addition of a mechanism to skip static cells in updates. A comparison of the computational benefits and costs of these aspects within the library is made in Section 4.3, based on the efficiency of calculating the progression of particular cellular automata.

Given the fundamental design of this library described in this section, as well as the novel approaches that are discussed in the next section, a brief exploration of a few potential applications is discussed in Section 4.4. The applications encompass a number of topics, both new and more common to the field, before focusing on the two main applications that further differentiate this library from other available systems: using the library alongside interactive

systems and genetic algorithms. Both of these applications have been included in the hardware demonstration that is discussed in the next chapter.

4.1. Novel Aspects

There are a number of improvements that have been made in this library in comparison to most traditional approaches in the field of cellular automata, both in terms of computational speed and in terms of design freedom. These traditional approaches generally use an array of state variables to represent the cell matrix, and require embedding prior knowledge of the boundary conditions within the cell state update algorithm [64]. To achieve the improvements in this library, a tradeoff has been made where individual cells are represented by basic data structures instead of simple state variables. This has been done not only to simplify shared memory use, but also to allow the cells to inherently have access to their neighbours' states, as two of the three core elements in the structure of each cell are the cell state and a list of the neighbouring cells. The other element enables a mechanism for skipping static cells called *sleep*, which will be explained further in the second subsection. There are few other systems that use structures for the cells because of the overhead needed and the memory footprints that exist in very large systems. However, the benefits of these more complex data structures in the majority of cases when using this library vastly outweigh the issues that may arise from their use. A thorough discussion of the various improvements that tip the scales in favour of these data structures follows.

The first major improvement, and the feature used to implement the interconnection framework, is the ability to specify how each of the different boundaries will behave on a given cellular automaton, down to the individual cell level. This freedom on the system edges allows for both the traditional choices of having a static boundary condition for the system as a whole, as well as the ability to have any number of edges, or small sections of edges, be specified independently. These newly independent boundary cells can be defined in a number of ways that include as inputs to the system, information transfer points between systems, or simply using a temporally periodic pattern of states instead of just a single static homogenous state.

Building on the ability to specify each boundary cell uniquely, an interconnection graph can be created by manipulating the boundaries of individual cellular automata implemented. This is achieved by making the edges of each pair of interconnected cellular automata act as dynamic information transfer points. Using this mechanism the framework can be directly implemented,

allowing its benefits to apply to the library. As an extension, the library also has a component that can automatically build a specific interconnection graph to facilitate boundary initialization. The details of these interconnection mechanisms are further discussed in Subsection 4.1.1.

One feature that any computation on an even slightly parallel system should take advantage of is the use of threading to divide the computational task into smaller, parallel pieces. However, there is an advantage built into this library that makes threading far more powerful than in other setups. Since the cells are represented by shared data structures, there is no need for any direct communication between interconnected systems after initially setting up the boundary conditions. Based on this mechanism, each of the nodes in an interconnection graph can be assigned to a thread that requires no communication with any other thread. This means that the threads, as used in this library, allow a substantial savings in computational time on any system with multiple processing units. It must be noted that threading is a separate concept from the splitting that occurs in an interconnection graph, and that both threading and interconnection will actually slow down computation on a single-processor system.

The second major improvement is an efficient method for skipping over unchanging cells, which is considered a sparse matrix technique in cellular automata optimization [64]. In this library, this is accomplished with the inclusion of a 'sleep' flag within the data structure of each cell. If, in a given time step, neither a cell nor any of its neighbours change state, then the cell will turn on the sleep flag to indicate that it is sleeping. Sleeping cells are skipped in the system update until they are woken by a changing neighbour, which leads to a much shorter computation time in any transition rules where there are more static areas than dynamic ones. The specifics of how the sleep mechanism works and its benefits are discussed in Subsection 4.1.2.

In Section 4.2, the specific implementation design of the library is discussed in detail to ensure a full understanding of the various components that are used to achieve these novel aspects. The overall design of the library is shown, along with the various interfaces and general structures that are used to allow both the core implementation of cellular automata as well as the implementation of the novel aspects described later in this section. A specific instantiation of the library in software is also introduced, and illustrative comparative results, in terms of how the two improvements above affect the efficiency of the progression of a few particular cellular automata over standard implementations, are given in Section 4.3. These comparisons are included to give a sense of the costs associated with the inclusion of both of these novel aspects

of the library to be weighed against their general benefits. There is also a specific exploration of how the variations in the global dynamics among cellular automata will change the specific behaviour in terms of the costs and benefits in efficiency of progression when using sleep.

Finally, a major addition to this library compared to more traditional approaches, which typically run independent of their environment, is the ability to interactively modify both individual and groups of cell states, the parameters of a particular cellular automaton, and the progression of time through a graphical interface. As most cellular automata tasks are interpreted as a form of visual feedback, this allows a user to create an entire system from the base cellular automata right up to components of the user interface from within this library. This is further explored, along with other applications of the library, in Section 4.4.

4.1.1. Interconnection

As previously introduced, there is a mechanism within this library that explicitly allows the implementation of any interconnection graph and its various components. In particular, the shared data structures representing cells are accessible through the library neighbourhood mechanics to as many interconnected systems as are needed to reproduce the graph interconnections. This mechanism is extended through the inclusion of a component for automatically building a limited range of interconnection graphs to prevent the need to manually specify boundary conditions for every edge cell.

The implementation of interconnection in this library is mainly based on the idea that the edge cells of the corresponding sides of two interconnected cellular automata act as though they were neighbours in one larger cellular automaton. Due to the use of data structures in storing cells, and that each cell therefore knows exactly which other cells are its neighbours without caring about which system each cell is a part of, the extension from assigning a local cell as a neighbour to assigning a cell from another cellular automaton as a neighbour is fairly minor. Thanks to this simplicity of assignment, the only issue that arises in the implementation of interconnection is that the states of the cells on the edges of a cellular automaton may need to be accessed concurrently by multiple threads or processors. Note that concurrent access to data brings with it the added complexity of having to consider mutual exclusion and deadlock prevention.

Only the limited subset of interconnection graph nodes from Figure 3.3 in Section 3.2 are implemented in this library, primarily to allow an exploration of the interconnected systems that are geared towards cellular automata. The implementation of other nodes is enabled through the

extendability of the library, and allows the creation of any interconnection graph that is desired. Provided that the interconnection itself works, additional functionality has been added to the interface for this library that enables the automatic interconnection of anywhere from just two cellular automata up to a multidimensional array, given that the edge connections between any pair are bidirectional. In the array format, two cellular automata are considered to be paired if their locations within the array are adjacent. This component also allows the selection of various directions of periodic boundary conditions on the outside of the interconnected systems that can occur along with the automatic interconnection to build a variety of interconnection graphs.

4.1.2. Sleep

The addition of a 'sleep' flag to each cell, as introduced above, is a major contributor to the efficiency of updates in this library. Using this flag, each cell is labelled as either: awake, ready for state updates based on the transition rule; or asleep, maintaining its state and ignoring transition updates. The flag is a part of each cell's data structure, as described in the next section, allowing it to be changed by both the cellular automaton that it is a part of as well as any other systems it might be connected to through interconnection. Although only the local updates can trigger sleep, the ability of a changing interconnected cell to wake a local cell is vital to the accurate application of all interconnected systems' transition rules.

A given awake cell will be flagged as asleep, after all updates have been applied in a specific time step, if neither it nor its neighbours have changed state in both the current update and the previous update. Once a cell is flagged as being asleep, it will be skipped over by the transition rule updates and will remain in its current state until one of its neighbours is changed. At the end of the same time step that one of its neighbours changes, a sleeping cell is woken up so that in the next step it regains the ability to change state based on the states of its changed neighbours. In this way, there is no possibility for a transition update to miss a sleeping cell that was supposed to have changed in a given time step.

Although similar systems are sometimes used in other cellular automaton simulators (generally labelled as sparse matrix techniques), the particulars of the implementation of sleep as it exists in this library seem to be unique. The unique features of this method are primarily: the use of a localized flag within the cell, and the changes in sleep being triggered purely based on whether or not a given cells' neighbours and local state have changed. The primary advantage of this localized method of skipping static cells is that there is no need for a supervisory level of control,

which directly allows implementations of individual cells, or small groups of cells, on distributed hardware systems. These localized features also allow the system to be used with a much wider range of system parameters than most other efficiency methods that have been developed[4], which are generally written in a way that specifically assumes the use of a particular subset of system parameters, since there is no limitation in this library on the parameters that are used.

The inclusion of a sleep flag in each cell, and the sleep mechanism that it enables, creates a number of direct advantages in both computational aspects and global behaviour analysis. In terms of computation, there are direct effects on the time it takes to apply the transition function to the cellular automata in question. In systems that have large regions of static behaviour, or which have a quiescent state, the ability of cells to sleep vastly reduces the computation time. This is primarily due to the majority of computation time used in cellular automata being dedicated to computing the outcome of the transition rules, often with a constant repeated result for whole regions of cells. However, in systems that are very active throughout the entire cell matrix there is an expensive overhead of checking and waking cells with little benefit in time gained in exchange. These varied effects are investigated for a few specific well known sets of cellular automata parameters in Section 4.3.

On the topic of global behaviour analysis, a few different characterization tools arise from the use of sleep. For example, if a cellular automaton exists where a region of cells sleeps for a few time steps without being woken, then the specific pattern of cell states in that region can be classified as static, or a *still life*. In examining the dynamic number of sleeping cells, classification can be made of various types of activity for a particular system that takes into account the effect that the parameters may have. Finally, with a range of different initial conditions an investigation can be made of how the systems react in general to random input, and how much randomness is maintained or created over time. This is directly related to Wolfram's work on the relationship between externally supplied and internally generated randomness [32].

4.2. Implementation Details

In this section, the details of how the library can be implemented purely in software are discussed. This consists of a description of the particular objects that have been developed as part of one specific instantiation of the library design. The class structures and associated methods for

4 Published implementations of detailed cellular automaton code are difficult to come by in the literature. Comparisons with “other methods” or “existing systems” is the result of a web survey of posted cellular automata tools and code. For several examples of such websites see [12], [65], [66], and [67].

each cell are described, as well as those associated with each cellular automaton. These classes are used in the construction of a connector class which has the ability to automatically interconnect cellular automata. Finally, the elements of the graphical user interface are discussed, along with the various methods that allow user input through both a keyboard and a mouse.

Each cell in the library is defined by a class (*Cell*, shown in Figure 4.4) which contains specific information: the states of the cell in the current and previous time step (*state* array), a list of the data structures of the cell's neighbours (*neighbours*), and a flag that indicates if the cell is awake or not (*awake*). Using this information, the *Cell* class contains a method (*applyRules*) that allows it to calculate an index into a compressed inner-dependent, outer-totalistic rule table (*rules* in the parent *CA* class) to find its next state. It also contains a method (*wakeNeighbours*) that is called once all of the cells in the parent *CA* have updated, which determines whether or not it should be waking its neighbours based on its state changes. There is also a second class (*Cell2D*) which implements the *Cell* class when using 2D cellular automata, mainly to allow the cell to check what row (*row*) and column (*col*) of the *CA* it is located at (using *ego*). The two classes also contain a number of elements and methods (*x*, *y*, *showCellTile*) to assist in the creation of the graphical user interface (GUI) to enable the display and manual manipulation of cell states.

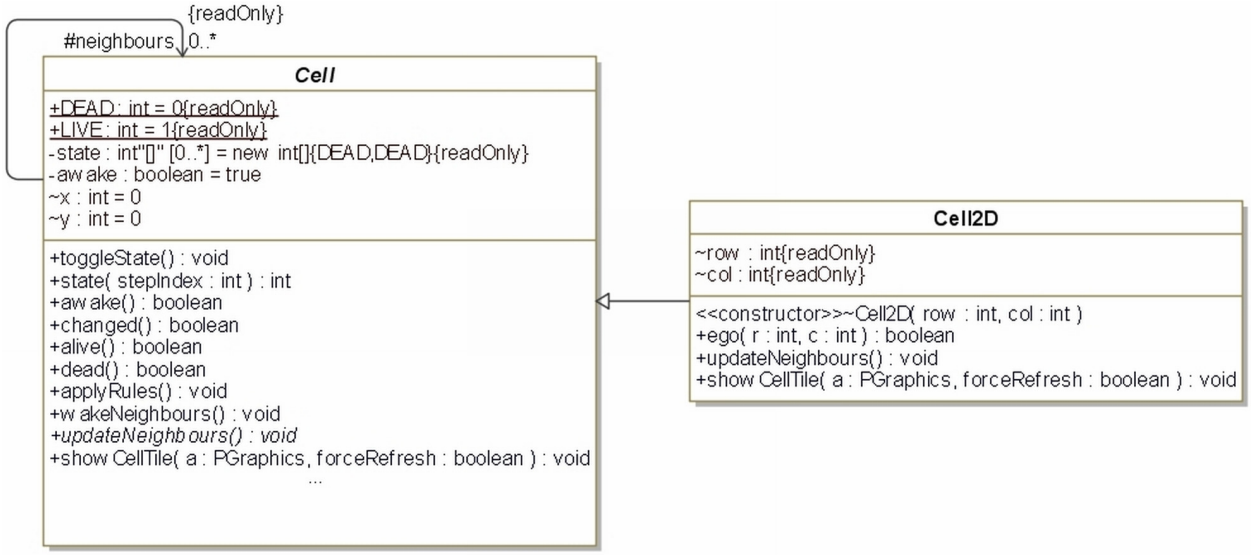

Figure 4.4. A UML diagram of the *Cell* and *Cell2D* classes. *Cell2D* implements the *Cell* class.

A pair of much larger classes also exist that define each cellular automaton (*CA* and *CA2D* in Figure 4.5), along with a number of smaller helper classes that simplify interconnection (*Stepper*), define constants (*CA2DConstants*) or specific 2D patterns of cell states (*Stamp2D*), and allow specific shapes of cells to exist(**CA2D*). The *CA* class contains the specific transition

rules for the system (*rules*) and an *n*-dimensional array of its cells (*cells*), along with a number of elements that facilitate the graphic representation of the cellular automaton (*_x*, *_y*, *cellShape*, *cellSize*, etc.). In addition to the methods associated with the GUI (*drawCell*, *show*, *makeTile*, **Background*, etc.), the *CA* class also contains a number of abstract method specifications for setting (*fillCells*, *randomize*, *switchClickedCell*) and updating (*updateCells*, *updateSleepers*) the states of its cells based on the particular demands of an implementing class, such as *CA2D*. The *Stepper* class is a singleton which allows multiple cellular automata to update synchronously.

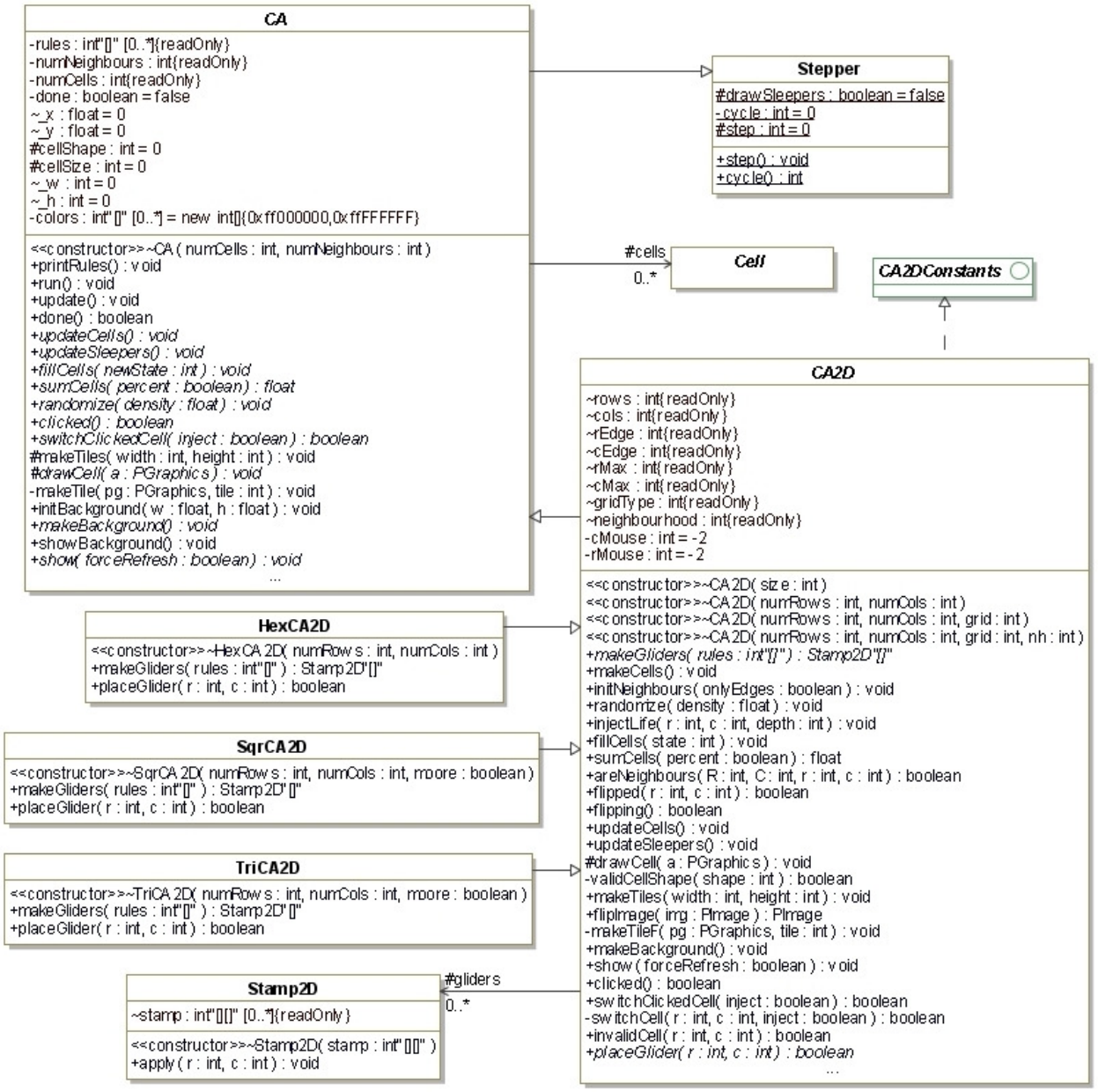

Figure 4.5. A UML diagram of the *CA* and *CA2D* core classes, including the various smaller helper classes that they are connected to through inheritance and implementation. The *Hex-*, *Sqr-*, and *TriCA2D* classes all implement the *CA2D* class, which in turn implements the *CA* class. The *CA2D* class also inherits constants from *CA2DConstants*.

The *CA2D* class implements all of the abstract methods of the *CA* class for a general 2D cellular automaton, as well as some of its own methods to create the 2D cells (*makeCells*), set the list of neighbours for each cell (*initNeighbours*), and to allow the insertion of random cell states (*injectLife*). The class also has an array (*gliders*) of specific 2D patterns (*Stamp2D* objects) which are created (*makeGliders*) and applied as cell states (*placeGlider*) using abstract method specifications for an implementing class, of which three exist (*HexCA2D*, *SqrCA2D*, and *TriCA2D*). Similar to the *CA* class, the *CA2D* class contains a number of elements (*cMouse*, *rMouse*) and methods (*flip**, *clicked*, **Glider*, etc.) that assist in its graphical representation.

Using the *CA2D* class, an automatic interconnection class (*CA2DConnector* in Figure 4.6) was developed using an overloaded method (*connect*), which can take as an argument either a single *CA2D* object or a 1D or 2D array of *CA2D* objects, along with a few simple arguments for wrapping and orientation, and create the bidirectional interconnections as the arguments specify. The *wrap* argument dictates whether to wrap the cell matrix edges periodically in both directions, while the *wrapTtoB* and *wrapLtoR* arguments do the same for the two possible directions individually. The *orientation* argument specifies whether the objects in a 1D array are spatially indexed as a horizontal or vertical set, while for a 2D array it specifies if the array is indexed using rows then columns or columns then rows. Note that the CA2DConnector class currently can only create interconnections between CA2D objects with the same cell matrix dimensions.

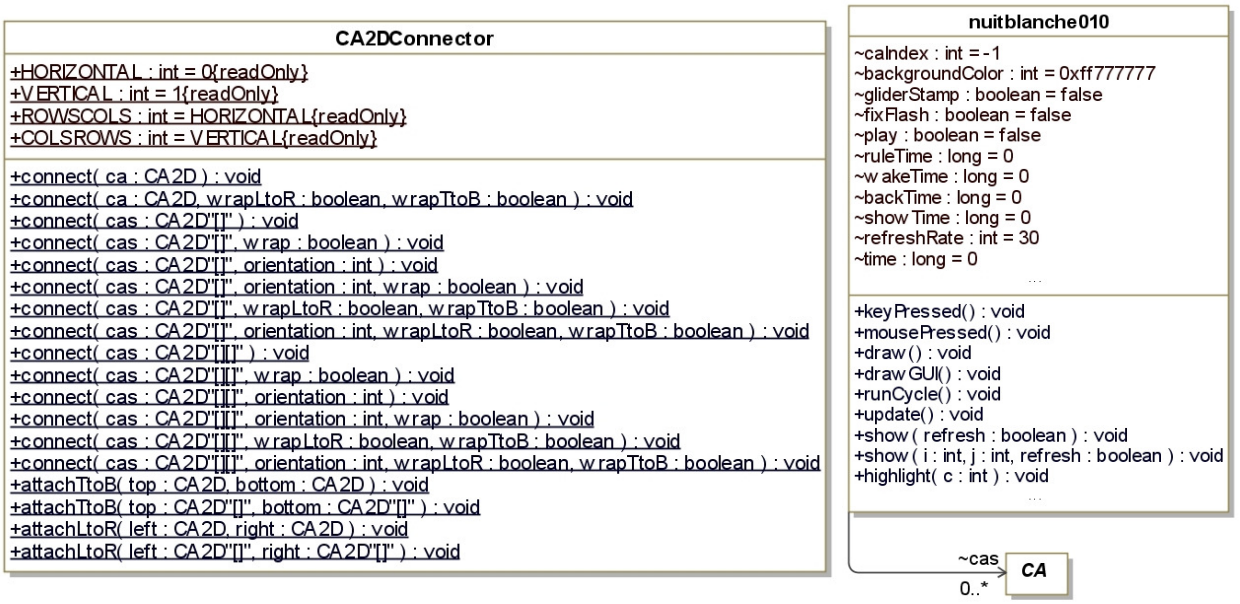

Figure 4.6. A UML diagram of the *CA2DConnector* class and the GUI class (labelled *nuitblanche010*). The connector class is designed to interconnect a single pair, or a 1D or 2D array, of *CA2D* objects in a simple bidirectional grid pattern. The GUI class allows the keyboard and mouse interactions and controls the update speed.

The GUI is primarily made up of methods and elements in previously described classes, as well as a main GUI class (*nuitblanche010* in Figure 4.6). In addition to the GUI elements (*fixFlash*, *backgroundColor*, *caIndex,* etc.) and methods (*draw**, *show*, *highlight*), this class controls the updates of the entire interconnected system (using *runCycle* and *update*), along with a number of methods that allow direct interaction with the system in real-time using a keyboard (*keyPressed*) or mouse (*mousePressed*). This interaction is mainly used to manipulate the progression of time (*play*), but can also be used to assign new parameters to a given *CA*, toggle the state of a particular *Cell* in the cell matrix, or even to directly randomize a limited set of cells or set down a glider pattern if one is known (*gliderStamp*). The interface, like the library, is fully extendable to allow the definition of any possible command that is desired from any supported input device.

Although the high level design established at the beginning of this chapter could be instantiated using a number of different software languages and hardware elements, the version of the library described here was developed using the Processing language and environment on a personal computer, and on its completion will be posted to the Processing website [68]. Processing is a high-level interpreted language which was developed to provide access to powerful computing and interface functionality for novice programmers, specifically for the arts and design communities. The selection of Processing as the implementation language for this instantiation enables direct access to the realm of cellular automata for these non-specialist groups.

This software includes all of the components of the library, both novel and standard, to facilitate its use in a wide range of different potential tasks from the fields of art to research to industry. Not only does this software enable the validation of the results of this dissertation, it allows the construction of systems that have specific applications in mind from the beginning of the design process. It can act as a simulator for a distributed system, and can be extended to model a given distributed system graphically with a user interface, thanks to its instantiation as a Processing library. The direct benefits of this instantiation as a Processing library include the ability to create graphics in 3D, make full use of a computer's mouse and keyboard as input devices, and even interact with the Internet or physical hardware when combined with other simple libraries.

As should be evident given both the high-level architecture from earlier in this chapter and the specific software classes described above, this library satisfies all of the design criteria that were set out in Section 1.2. The interconnection framework has been implemented, albeit for a specific subset of nodes, including an automatic connector class for basic grid-like systems. The entire

high-level library design has been built, demonstrating its usability in a purely software environment. The architecture is completely open and, thanks to the freedom of the Processing language and environment, can be easily interfaced with a variety of other systems. The software has been designed to be able to accommodate all of the parameters in the restricted set from Figure 4.1. Finally, the software can be extended to use more parameters, have other systems integrated into all levels of abstraction, and has already been extended to allow the injection of random cell states and the insertion of predefined patterns of cell states.

Using the software library that has been described in this section, a number of computational results have been found. This data was collected through the simulation of different types of interconnected systems of cellular automata, which are introduced and discussed in the next section. These simulations make use of most of the structures and methods that have been discussed above, with a specific focus on the effects of sleep and interconnection on the computational efficiency of various sets of popular cellular automata parameters and different interconnected systems, respectively.

4.3. Results

This section provides and discusses the results of a number of profiling experiments that were designed to determine the effects of this library on the efficiency of system updates in cellular automata. In these experiments, the total number of cells in the system was kept constant, independent of how many cellular automata there were, and the initial conditions of these cells consisted of a randomly chosen set of states with a density of 50% (half of the cells in the on state). To ensure consistency across these experiments, an easily repeatable set of system parameters was desired that would facilitate the validation of these results, aside from those based on interconnection, using other libraries.

Due to its popularity in existing 2D cellular automata research, John Conway's Game of Life was chosen as the primary parameter set. As described in Section 2.1, it uses binary cell-states, a 2D square tiling, a contact-based Moore neighbourhood, and specific inner-dependent outer-totalistic rules. The Game of Life has very complex global dynamics, including both a quiescent state and moving patterns, which allow it to act as a universal computer.

To determine what kinds of effects different parts of the library might have on the efficiency of updates, an experiment was carried out that was designed to separate their contributions to the

overall system efficiency. Each of the novel aspects of the library was enabled and disabled, alone and in combinations, to discover both their individual and concurrent effects. Based on these experiments, most of the aspects of the library did not have any measurable effect on the efficiency of the system. However, both the interconnection and sleep mechanisms made significant changes to the efficiency of the system, both independently and when combined together. The results of a few simulations with combinations of sleep and interconnection can be seen in Figure 4.7, with 'Split' meaning that one large cellular automaton is split into smaller identical parallel systems in the same way as the left transition of Figure 1.2.

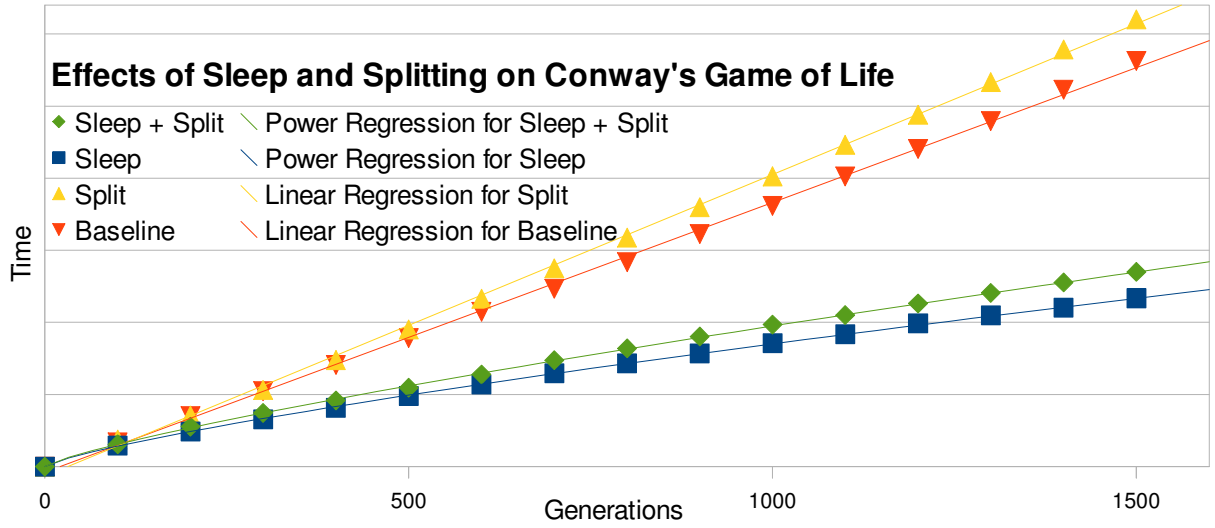

Figure 4.7. The effects on computation time when using the 'sleep' flag and splitting a large cellular automaton into an interconnected grid of identical cellular automata.

As is evident by these initial results, the only novel aspects of this library that have any measurable effect on the efficiency of the system are those directly involved in the cell updates. This result is to be expected, as the majority of computing resources used by cellular automata are dedicated to cell updates. On top of the overhead of the standard cellular automaton implementation, interconnection requires additional overhead for two main reasons: to provide each system access to any neighbouring systems' edge cells, and to allow each cellular automaton in an overall interconnected set to individually update themselves. This knowledge, when combined with the seemingly identical drops in efficiency seen in both split systems in Figure 4.7, seems to indicate that interconnection creates a constant overhead. Upon further analysis, it can be seen that the addition of interconnection shifts both the linear baseline and the non-linear sleep data by a constant factor, consistent with a constant overhead. A discussion of interconnection's influence on the efficiency of updates follows in Subsection 4.3.1.

The effects of sleep on efficiency are somewhat more complex. Although it also has an associated overhead, there is no clear method of predicting what it will be. This unpredictable nature is due to the variations of which cells require updates to the sleep flag from step to step. In isolation, the overhead reduces the efficiency of updates based on the unknown global dynamics. There is also an added overhead once interconnection is introduced, to allow sleeping cells on the edge of a particular system to wake when their interconnected neighbour cells change. The potential gain in efficiency by using the sleep mechanism can offset this entire overhead, but it is also based on the global dynamics, making the effects difficult to state with certainty. A full discussion of the effects of sleep on system update efficiency is in Subsection 4.3.2.

4.3.1. Interconnection

To determine whether or not the overhead due to interconnection is actually a constant drop in efficiency, as hypothesized previously in this section, a number of experiments were carried out. Since the overhead appears to be based solely on how the interconnection relates to cell updates, these experiments were designed to compare the different independent variables involved. The variables in question are the number of edge-connected cells and the number of individual cellular automata in the overall system. In these tests, the total number of cells and all parameters that are not involved with interconnection remain constant.

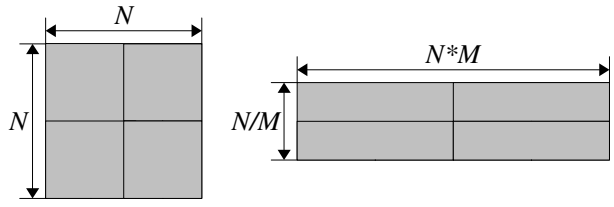

Figure 4.8. Diagram showing how the number of edge cells was varied without affecting the number or pattern of cellular automata in an interconnected set.

The number of edge-connected cells can be modified directly without affecting other variables by using a consistently interconnected and constant number of cellular automata in a pattern and varying the overall dimensions of the set (as in Figure 4.8). In this experiment, the change in dimensions varies the shape of the set of cellular automata from square to a thin rectangular strip while maintaining a constant number of cells, causing the number of edge cells to vary across a large but finite range. Note that M must always be an integer divisor of N to ensure that any configuration that fits this definition has the same number of cells. The results of these variations have been summarized in Figure 4.9, with each set of data points representing a particular number of cellular automata in a given pattern. As seen in this graph, the number of edge cells in the system appears to have a constant linear effect on the time it takes the entire system to update for a given pattern and number of

cellular automata. This effect ranges from approximately an 8% increase with two systems up to approximately a 15% increase for larger numbers of systems.

As the number of edge cells appears to have a constant effect on efficiency for a given data set, the impact of the number of individual cellular automata can be analyzed by comparing the different data sets. Other than the case of two systems, as the number of cellular automata increase, they appear to have a beneficial impact which gives the appearance of a reduction in the amount of overhead for a particular number of edge cells. This is likely an effect of the multiple cellular automata being evaluated in parallel when possible, one of the main benefits of interconnection. In the case of only two cellular automata, the reduction of the multidimensional array organization of the systems involved into a single array is thought to be causing the reduction in computing time, but this has not yet been confirmed.

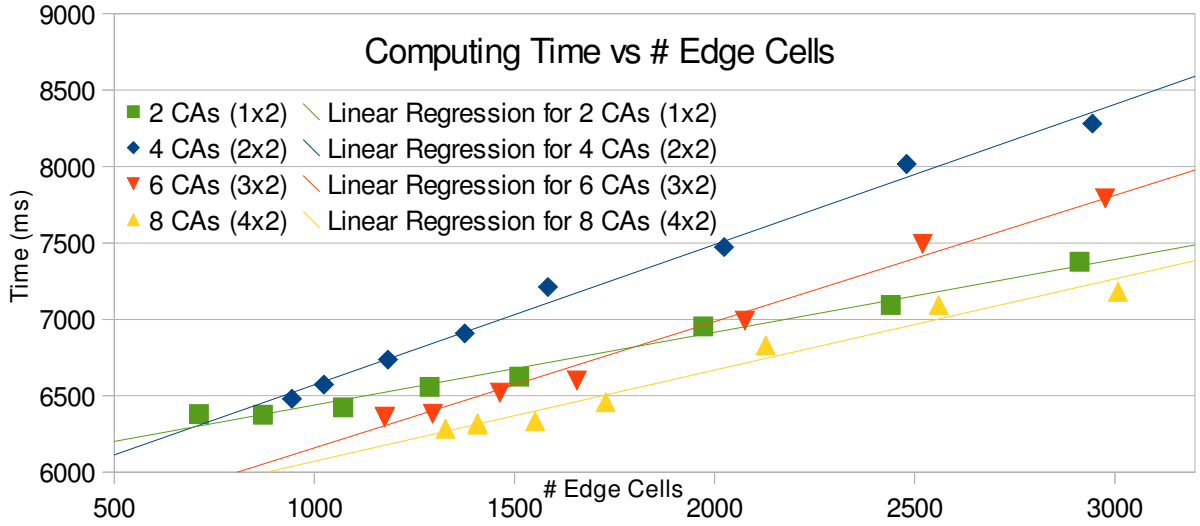

Figure 4.9. Effect on computing time of varying the number of edge cells in an interconnected system. Each set of data points represents a constant number of cellular automata in a specific pattern. The times have been averaged over ten executions of each particular data point, and each time found is for 1500 generations.

As is apparent from the results above, the drop in efficiency based on interconnection is linear and constant for a given number of cellular automata, creating an approximately 8% to 15% longer computation time. This value seems to be dependent on the number of cellular automata that are in use, as well as how well the implementation of the library will handle parallel tasks. In the next section, experiments on the impact of sleep are carried out.

4.3.2. Sleep

Experiments were carried out to separate the conflicting effects of sleep: the overhead of checking and setting the sleep flag, and the benefits of applying the sleep mechanism to skip

over static cells. Since both of these effects are directly related to the global dynamics of a given cellular automaton, a number of well known parameter sets were chosen (from listing at [69]) to be used in these simulations as shown in Figure 4.10, again in the same format as was used in Figure 2.6. The sets were specifically chosen to ensure common parameters with each other and the Game of Life other than their transition rules, to allow the known complex dynamics of each system's transition rules to contribute useful data to the sleep characterization. These common parameters are: square cells, a contact-based Moore neighbourhood, two discrete states, and inner-dependent outer-totalistic transition rules. These experiments were all carried out using a cellular automaton with a 240×240 cell matrix and periodic boundary conditions, with a data point every 100 generations up to 1500, averaged over ten executions of the same system. Also, the following conventions are used for sleep: *removed*, where all traces of sleep are removed from the code; *disabled*, where the sleep flag is updated but transition rules are applied whether it is set or not; and *enabled*, where sleep is actively preventing sleeping cell state updates.

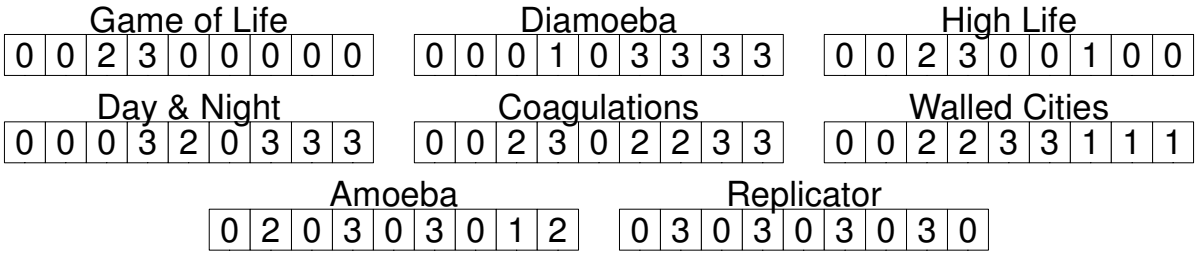

Figure 4.10. The rules that are used in the simulations for sleep. In all of these systems, the only parameter that differs from the parameters in Conway's Game of Life is the transition rules, so only those are shown here.

In general, the global behaviours of the rules used fall into three different levels of activity in cell state changes over time: consistently high, slowly decreasing, and rapid decay. The most active group consists of three of the eight chosen parameter sets: Amoeba, with areas of chaos that are continually changing size; Walled Cities, where rectangular regions of random chaos are enclosed in static lines of cells; and Replicator, where every initial pattern is copied eight times every 32 generations. Only one of the chosen sets has a slowly decreasing level of activity, Coagulations, as it produces many slowly changing patterns that gradually stabilize under periodic boundary conditions. The final group, those with rapidly decreasing activity levels which should benefit the most from the application of sleep, contains the remainder of the parameter sets. The global behaviour of the Game of Life has been described earlier in Subsection 2.1.1, and High Life is simply a slightly more active version of it, but the remaining two sets with rapidly decreasing activity levels are Diamoeba, which creates various forms of

diamond shaped static patterns of state '1' cells with changing edges, and Day & Night, which creates large regions of static patterns in both states with changing edges.

To isolate the drop in efficiency caused solely by the sleep overhead, the actual application of the sleep mechanism was disabled to ensure no hidden gain in efficiency. When the benefits of sleep have been disabled, the time increase shows how much is taken to check and set the sleep flags. As shown in Figure 4.11, this change in efficiency seems to be based on the overall activity level of the rule used. This means that although the overhead is limited to a given range, it seems to be directly dependent on the global dynamics of the specific rule. This overhead appears as a 25% to 75% longer computation time, which must be completely overcome by the benefits of sleep to justify including it as a component of the library. It is important to note that the rules used are chaotic, in that even very similar initial conditions can lead to completely different behaviour, and therefore attempting to fit a curve to the averaged sleep data has little to no meaning. However, it is possible to cluster the rules into high- and low-activity groups (high/low percentage of cells changing state frequently), as shown by the highlighted areas of Figure 4.11.

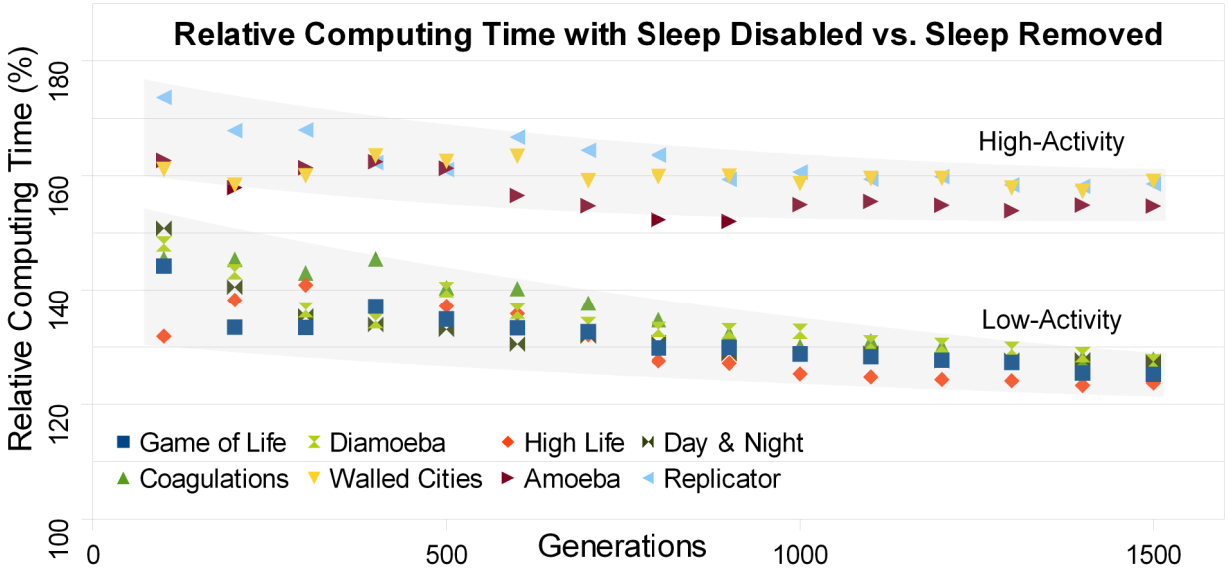

Figure 4.11. The relative computing time used when sleep is disabled, compared to a normalized 100% baseline when sleep is removed, for the various popular parameter sets as labelled. Other than their specific inner-dependent, outer-totalistic transition rules, these parameter sets are identical to those of Conway's Game of Life. Note that all of these rules are chaotic and non-linear, and the relative computing time changes with every set of initial conditions.

Taking this dynamics-dependent drop in efficiency into consideration, further simulations were carried out with sleep enabled and compared to simulations with sleep completely removed from the library. As shown in Figure 4.12, the overall effects of the sleep mechanism on computing time varies widely based on the global dynamics of the transition rules that are used. In general,

these results are to be expected based on the methods used to implement the sleep mechanism, however a number of interesting results for the specific rules are described below.

Based on the typical state progression in each of these rules, sleep seems to majorly benefit only those rules which have many static patterns, a quiescent state, or both, as expected. This group of rules, as shown in the bottom group highlighted in Figure 4.12, should always have sleep enabled. The rules which have very active cells consistently over time still suffer drastic negative drops in efficiency when sleep is enabled which closely match the overhead of these rules in Figure 4.11, only improving over those times by 15-20%. This means that for any rule with a high level of activity over time, sleep should be completely removed from the update process. However, for rules like Coagulations, where a mass of activity dies off slowly while leaving behind static regions of cells, it appears that to maximize efficiency throughout the entire run-time of the system, sleep must be removed until the critical point when it will start to speed up the system. These massive differences in efficiency effects of the sleep mechanism, not only between rules but even over time in one particular rule, lead to a foreseeable need in the near future for an intelligent decision making algorithm to discover in which situations that sleep needs to be activated to optimize computing time, and when it should be removed altogether.

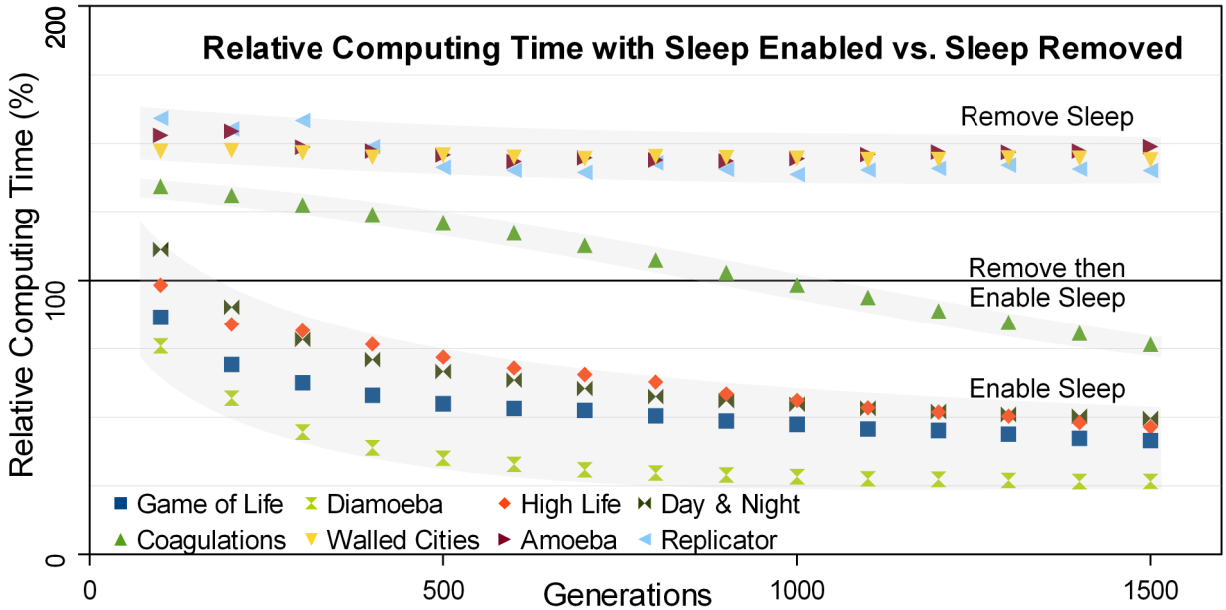

Figure 4.12. The relative computing time used when sleep is enabled, compared to a normalized 100% baseline when sleep is removed, for the various popular parameter sets as labelled. Other than their specific inner-dependent, outer-totalistic transition rules, these parameter sets are identical to those of Conway's Game of Life. Note that all of these rules are chaotic and non-linear, and the relative computing time changes with every set of initial conditions.

By taking the shape of this sleep efficiency graph for a given rule and comparing it to a set of common rules with well known global dynamics, general statements can be made about the types

of global dynamics that are present in the new rule. A number of comparable rules, or a linear combination thereof, could be used to effectively filter and classify a potentially useful new rule. This classification and filtering mechanism is applicable in a wide range of applications, including finding universal systems, searching for particular global behaviour, or even as a fitness function in genetic algorithms as these are all generally related to how the sleep mechanism will perform on particular sets of parameters.

The changes in computing time when using the novel aspects of this library, particularly interconnection and sleep, have been discussed along with the results of a number of simulations to verify these results. Although there is an overhead associated with both interconnection and sleep, their use introduces a range of benefits that greatly outweigh their respective overheads. In the next section, applications of this library are discussed along with specific types of systems that directly benefit from both of the novel aspects that have been analyzed in this section.

4.4. Applications

Building on and using the library that has been defined in this chapter up to this point, a wide range of applications present themselves. A few of these applications are extensions of common tasks in cellular automata research, while others are introduced entirely by the novel aspects of the library and framework. In particular the addition of cell-based boundary specifications, including the interconnection mechanism, creates the potential for completely new tasks as well as new solutions to common tasks. These applications can range from an investigation of the effects of the separation or combination of various popular cellular automata systems, through to the use of the library within the fields of interactive systems or evolutionary computing.

A direct application in the implementation of these systems is derived from the introduction of interconnections between distinct cellular automata, and arises due to the simplicity in software and hardware of using these interconnections to distribute computation. The ability to separate a very large cellular automaton into arbitrarily small homogenous subsections, each sharing boundaries and using the same rules, allows the computation of much smaller relative parts without losing the global behaviour that is desired. In this way, the interconnections between cellular automata are a directly applicable method of determining how to separate computational tasks among many parallel systems to satisfy the primary criterion of this library.

The cell-based boundary specifications can also create the ability to use any number of edges of a cellular automaton, or group thereof, as inputs to or from an interconnected system. A given input to a *n*-dimensional cellular automaton only needs to take the form of a (*n*-1)-dimensional array of cell states, and can be based on anything from internal statistics to external sensor data to be a part of the interconnection graph. There is also the ability within this library to directly change the state of any cell in the system, so that direct human input can be used in real-time to insert a pattern or change an ongoing computational process. This leads directly to the use of this library within the realm of interactive systems, as discussed in Subsection 4.4.1.

Building on the foundations of 1D hybrid cellular automata where each cell has a different set of rules, there is also a direct application in the use of interconnected systems, each having different rules to solve various tasks. This would allow the benefits of hybrid systems to grow from simple 1D problems to the vast realm of 2D problems in computer graphics and image manipulation where currently, general 2D cellular automata can perform only simple pattern recognition. It could even allow existing 2D character recognition cellular automata to be interconnected and made more efficient, with a high-level guide dictating which set of different pattern recognition rules to use for each section of the problem.

Finally, using the interconnection mechanism as a directly specifiable parameter of a set of cellular automata enables the use of the presence of edges in the interconnection graph as a genetic parameter within the realm of genetic algorithms. Not only can suitable parameters of a given cellular automaton be determined to solve a task, but using this library the interdependent parameters for multiple systems, and how they are interconnected, could be discovered. By being able to change how the systems are interconnected, the potential search field can be expanded by many orders of magnitude. Applications of this library when combined with genetic algorithms are introduced in Subsection 4.4.2.

4.4.1. Interactive Systems

In general, interactive systems involve a form of input from a person that affects a perceptible change in the system. The field of interactive systems, outside of its sub-fields of human-computer and human-robot interaction, is primarily focused within the art and architecture realms. There are many examples of interactive art, with a large majority of them taking the form of technologically enabled artistic pieces generally known as tech-art such as *Aurora*, the installation that is described in Chapter 5. In terms of architecture, a few simple forms of

interactive systems are intelligent building shading systems that are based on human occupation, and lighting systems that are motion sensitive. These are similar to the examples of the current uses of interactive cellular automata in this field that have been described in Subsection 2.2.2. In comparison to these existing examples, the library that has been developed allows a much more complex set of interactive elements and overall interactive systems to exist.

The freedom of specification for each edge of a cellular automaton in this library allows for a number of different types of inputs to modify and influence the state of those edge cells. This input can come in many different forms so long as it can be modified or manipulated to appear as a set of cell states. The states can be dynamic, static, random, or even based on an entirely separate cellular automaton of their own, as introduced at the end of Section 3.1. Also from that section, there is the possibility of using a single input source as the boundary condition on either multiple edges of a single cellular automaton or the edges of multiple cellular automata within a larger, interconnected system.

The input could also be from an external source such as other software, other hardware, sensor values, or even direct human input. As an example, a sensor that directly outputs its value as a set of cell states could be used, similar to how ambient light levels are used to create initial conditions for the 1D shading system from Subsection 2.2.2 [9]. Or, from the simple prototype in [9], it could be based on a manual manipulation of the states of the edge cells while the cellular automaton progresses in real time. If the system were connected to the Internet, the input could potentially be driven by anything in the world.

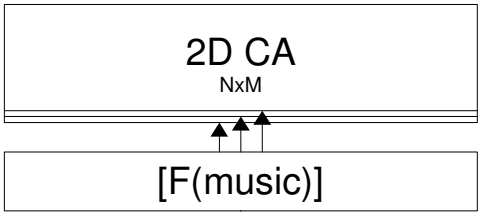

Figure 4.13. Diagram of the interconnection graph for how a dynamic node such as an equalizer could affect the bottom rows of a cellular automaton node.

With the addition of a small extension to the library, multiple layers of an edge could be based on various inputs. Using this extension, the bottom few rows of a cellular automaton, or interconnected set thereof, could behave as an equalizer in response to music (Figure 4.13). As long as the injection of cell states is done so it does not affect the rules, it could be done by an entirely separate system using a communication mechanism. If the rules for a set of cellular automata support moving patterns, or gliders, this would lead to various patterns of gliders sporadically rising from the bottom cells as they change to the beat of the music in the form of a digitally choreographed ballet.

In addition to the use of edges as inputs, the library allows the state of individual cells to be changed directly. It is further extended to allow the creation and insertion of any predefined pattern centred at a given cell within a system. In the graphical simulator built on the software library of Section 4.2, these placements can be chosen through direct human input. For example, if a system is using rules that support a particular glider, then the pattern for that glider can be specified as a valid pattern for that rule and stored in the software. Then, at any point during a simulation that glider can be placed anywhere in the lattice based on which cell is selected.

4.4.2. Genetic Algorithms

As discussed in Section 2.3, there is a broad base of work combining genetic algorithms with cellular automata to solve tasks and find various sets of parameters to satisfy different objectives. However, most research on combined systems in 2D is once again focused on the search for universal computation or self-replication. There are any number of reasons why this might be the case, but based on the current limitations of available cellular automata libraries one of the contributing factors is likely that there is no library currently available that has an open enough architecture to allow any other research. This library has been designed specifically with applications in genetic algorithms in mind, and can therefore reduce these limitations.

The use of this library with genetic algorithms is simplified by the addition of a number of high-level functions that provide access to the kinds of information that are generally desired for task-based evolution. The first set of high-level functions relate to statistical manipulation of the system, and include the ability to fill a given system with a specific density of cells and to find the density at any particular time step. Another set of functions uses the sleep flag to determine various global behaviours, as described at the end of Subsection 4.1.2. These include how active the system's cells are at a given time step, and whether or not there appears to be a quiescent state in a given rule. This allows a genetic algorithm to search, for example, for a set of parameters which achieve a specific desired activity level, if that is the overall goal.

Other than the potential for a much faster computation of the fitness function using the threading and sleep features of the library, there is little change to the typical methods of combining evolution with cellular automata for any of the traditional tasks. Note that although sleep may help or hurt depending on the task, the ability to compute the fitness of every chromosome in parallel using threads leads to a faster system overall on any form of parallel system, even those which simply have a parallel processor in a serial architecture. To show that no core functionality

of a typical cellular automaton has been lost with respect to evolutionary computation, a compact genetic algorithm component was also developed as an extension to the library.

To validate this component, a demonstration task was needed to replicate the results of other work in the combination of cellular automata and genetic algorithms. After reviewing the literature, the density task (all cells reaching a common state based on their initial density) seems to be the most popular option for evolutionary computation. The majority of work in this field was done at the Sante Fe Institute, by Mitchell and Crutchfield and their students [43]. However, only a small part of this work was done in 2D, and even that was not done using totalistic rules. Further work on the density task was done in 2D using totalistic rules by Inverso et al [56], so their density task solution is used as the standard to which comparisons will be made.

To perform the density task, a fitness function was created based on the density of cell states after 100 time steps from a known random initial density. The chromosome used was simply a direct reproduction of the transition rules. The fitness of a chromosome was calculated after applying its rule to a cellular automaton, running the system for 100 generations, and then evaluating the fitness function, as shown in Figure 4.14. The task was run using both one large cellular automaton and the same system split into four interconnected modules (2×2). Using both of these methods, the genetic algorithm achieved the Inverso rule after the same number of evolutionary steps. However, the system took slightly longer to compute when using the 2×2 split system, as expected based on the results in the previous section of this chapter.

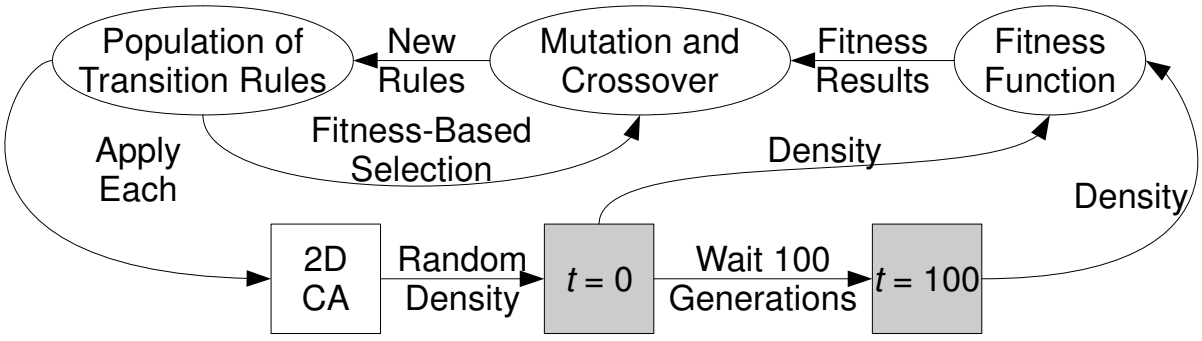

Figure 4.14. Diagram of the application of the density task fitness function on a population of transition rules. When using an interconnected system, the 2D cellular automaton was simply replaced with four connected 2D systems.

In addition to being able to replicate the findings of other work using this library, the ability to interconnect a number of cellular automata leads to many more possible genetic attributes. There is the opportunity for each of the individual cellular automata to have a selection of different transition rules, neighbourhoods, or even tessellations be a part of the chromosome that is

evolved. The inclusion of these interconnections as genetic components, and the distribution of different parameters among interconnected cellular automata, creates a vastly different search space from that which is typically explored. These diverse new possibilities have the potential to solve a wider range of problems than standard systems using only a single cellular automaton.

To demonstrate the freedom of these new genetic attributes, an extended version of the density task was created using a chromosome that was a concatenation of rules for each individual cellular automaton (shown in Figure 4.15). Additional crossover points were added between these concatenated rules to create the potential for maintaining whole rules when new chromosomes are created using the crossover mechanism. The search space for this chromosome, and therefore the task, is exponentially larger than that of the standard density task.

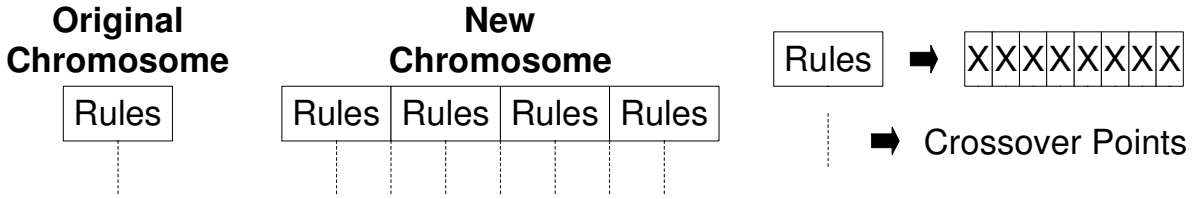

Figure 4.15. Diagram of how a rules-only chromosome is changed from the standard application of genetic algorithms with cellular automata to their use with a set of four interconnected systems.

In this extended version of the density task, the integrated genetic algorithm was still able to find the Inverso rule even though it had to effectively find the same rule four separate times. On average, this method took 5 to 8 times more evolutionary generations (and therefore real-world time) than the standard method. However, this extended system could be used to solve far more interesting problems that depend on scales of knowledge somewhere between the local neighbourhood and global tasks usually used, such as a version of the chequerboard problem where the alternating blocks of homogeneous cell states are more than one cell in size.

In addition to these attributes, the set of interconnections between particular nodes in an interconnection graph can be used as a parameter in the evolution of an overall system. By including the capability of arbitrarily networking various cellular automata together, the library allows the genetic algorithm to act like another member of the evolutionary computing family known as genetic programming. With a genetically determined network of interconnections, each cellular automaton acts as a time-dependent function. The input to this function is the pattern of cell states that cross the interconnections at each time step, while the output crosses back over the same interconnections concurrently. This introduces many new and interesting potential tasks

as well as providing new solutions to more common tasks, as it takes on the attributes of genetic programming [70]. A selection of these tasks and solutions are discussed further in Section 6.2.

In summary, this chapter has discussed the library that has been developed to satisfy the criteria as set out in Section 1.2. This library has implemented the interconnection framework from the previous chapter, as well as added a few unique components including interactivity and sleep. The instantiation of the library in software has been discussed, with UML diagrams of all of the core classes used to implement the high-level architecture laid out at the beginning of this chapter. The key components of the library have been analyzed in terms of computational efficiency using this software instantiation, and the global dynamics of the particular cellular automaton the sleep mechanism is applied to have been determined to result in a direct impact on efficiency. Finally, a number of applications have been introduced as a brief outline of the potential uses of the library. In the next chapter, the development of a hardware application based on this library is discussed as a demonstration of both the library itself and as a concrete use of the framework from Chapter 3.

5. Application: Aurora

In October of 2010, an artistic installation, named 'Aurora', was created to fill the atrium space of the Royal Conservatory of Music in Toronto, Ontario for an event called “Nuit Blanche” (see Figure 5.1). Aurora won the People's Choice award, and was designed alongside the artistic and creative team at Philip Beesley Architect Inc. It was conceived of as a 1 m wide cloud of light, sound and movement that reacted to visitors and was 10 m tall and 25 m long. To achieve this concept, a set of 18 modular embodied interconnected cellular automata were developed that each contained 144 (24×6) cells, for a total of 2592 (18×144) cells in the piece. Each of these cells controlled a super-bright white LED and a vibration motor using their state.

Figure 5.1. Image of Aurora installation from the 2nd floor. Note the scale of the space based on the people.

The initial motivation for creating this installation using cellular automata was simply to gain a greater understanding of the computational complexity of the patterns generated, primarily

influenced by an instance of art based on the Game of Life [71]. Once an investigation of the state of the art in the field of cellular automata was carried out, it was discovered that the entire installation would need to be built as if it were one large cellular automaton with common parameters throughout. Since there was a desire for the behaviour of the different parts of the installation to be unique, or at least different from other adjacent behaviours, the need for a hybrid mix of cellular automata parameters became clear. However, to be able to use a hybrid mix of parameters within a single cellular automaton, it quickly became necessary to create a framework that would allow a level of hybridity that did not exist in any other implementations, either in hardware or software. This led to the initial development of the conceptual framework that is the focus of this thesis research, and a large portion of the library as well. Once these two core systems had been developed, their application in the design of this installation began.

The design requirements for this installation consisted of four key aspects: no centralized control or storage, the visitors had to be able to affect the piece, the behaviour had to change over the course of the installation, and there had to be some form of distinguishable movement of light. To avoid the need for any centralized systems, a parallel module was designed for each cellular automaton that connected to the other modules to form the interconnected system desired. Presence sensors were included that hung below the installation to allow visitors to affect the cells in their location through the interaction and sensor aspects of the library and framework, respectively. The behaviour of the cellular automata was designed to change over time by using a genetic algorithm to evolve new parameters at a set frequency. Finally, to guarantee that the lights in the installation showed movement, the parameters that were chosen had to support some form of moving cell state patterns.

In the next section, the interconnection graph for Aurora is defined and the overall system behaviour is discussed in terms of how it fits within the framework from Chapter 3. Following that, Section 5.2 describes the specific details of the embedded implementation of the library from Chapter 4, with a focus on how the various components of the library have been changed so that they work well in this particular hardware application. A description of a few post-installation investigations that have been briefly initiated can be found in Chapter 6 as a subsection of Future Work, that includes discussions of a method of tracking visitor interest and a preliminary look at its use in human-in-the-loop fitness functions for genetic algorithms.

5.1. Interconnection Graph

The interconnection graph for the embedded system that was designed for Aurora is shown in Figure 5.2. As is evident, there are three different node types used in the graph: 2D cellular automaton nodes, dynamic sensor vector nodes, and a static scalar node. The 2D cellular automata nodes used are each made up of 144 (24×6) cells, and share a number of common parameters: two states, square cells, and a size 1 Moore neighbourhood. The range of shades of the cellular automata nodes indicate that each of the nodes can have a different set of transition rules. The dynamic sensor vector nodes each provide a vector of six values based on the input of a set of six presence sensors hanging below the installation. In addition to providing the lower boundary condition, these sensor nodes can randomly affect the states of cells up to three rows away from the boundary, depending on the sensor data. The static scalar node has a value of zero and is connected to the top of all of the cellular automata, as well as the extreme edges of the installation, to ensure that the cell state activity in the installation does not become high enough that the movement of cell states appears to simply be random noise. This effect of having a boundary absorb cell state activity is achieved through the use of transition rules that have a quiescent state that is the same as the used boundary condition, so the boundary is therefore unable to inject any non-quiescent cell states into the overall system.

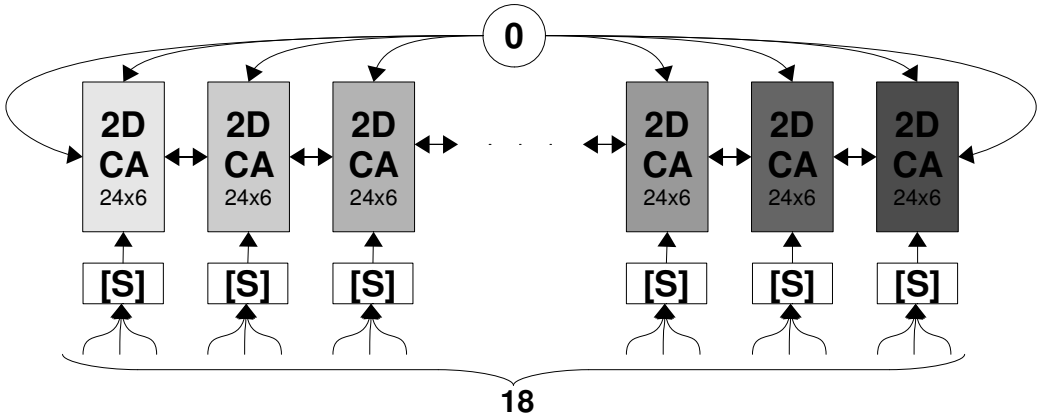

Figure 5.2. The interconnection graph for Aurora. As before, different shades represent different parameter sets.

The range of transition rules that are used in the installation are chosen from a restricted set of possible rules that all support the 'glider' pattern and movement from the Game of Life cellular automaton. This is done to allow 'glider' patterns to be created anywhere, move across multiple cellular automata in the installation, then either crash at the top and sides of the piece or dive into the visitor-affected cells at the bottom. The simplicity of the 'glider' design also enables the visitor-affected cells to easily create new gliders driven solely by their presence under the

installation. Based on the restrictions needed to support the Game of Life 'glider', the transition rules that were available are known to have many static patterns, so the sleep mechanism was enabled to reduce the time needed to do the state updates at each time step.

5.2. Hardware Library Implementation

To control this massive system without a centralized computer, a hardware-based version of the library described in the previous chapter was developed, as shown in Figure 5.3. This hardware application of the library was created using custom hardware designs and the open-source Arduino platform [72]. It incorporates a set of embodied cellular automata, each with their own computing and sensing elements, that allows local computation of each time step without any outside information. There is also a method for interconnecting one cellular automaton to the next using bidirectional cables hooked directly to shift registers to enable boundary interaction without needing a high-level communications protocol. This allows the cellular automata to be distributed across a huge space (250 m^2) and avoids using centralized systems. After the initial programming, the communication between controllers is limited to a global update message used as a heartbeat to ensure that the entire installation is switching its cell states at the same time.

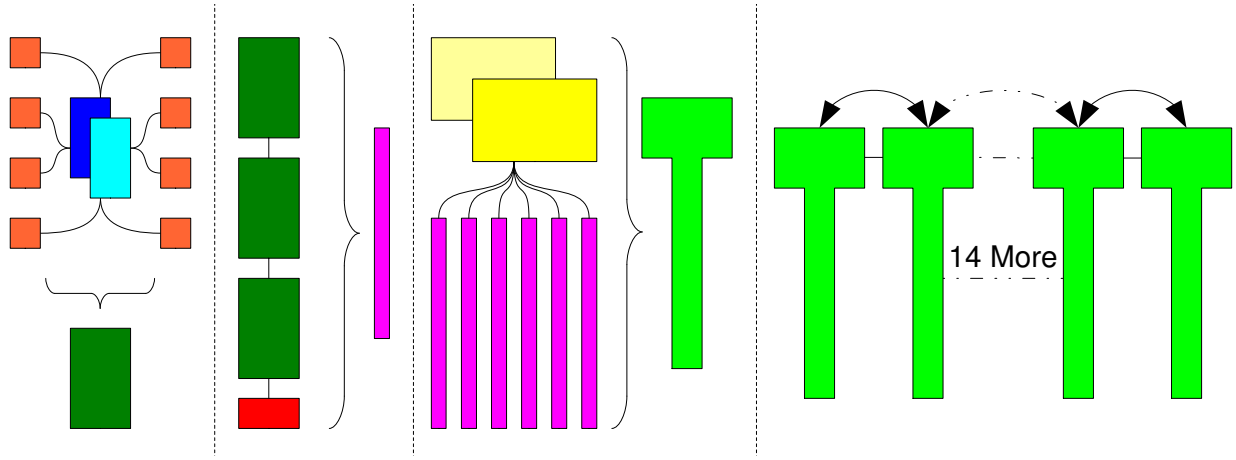

Figure 5.3. A block diagram of the hardware components used in Aurora. On the far left is a breakout unit (dark green) made up of the two boards (blue) that control the eight cells (orange) it is assigned. Just right of these, three breakout units are daisy-chained together, along with a sensor board (red) at the bottom, to form a cell column (purple). In the centre, six of these chains are connected to a controller unit (yellow) to form a single embodied cellular automaton system (green). On the right, 18 of these systems are connected using a communication link and a bi-directional cable to create the overall installation. The communication link is used on startup to program the initial parameters of each controller unit, then only as a global synchronizing heartbeat during operation. Total number of cells: 18 systems * 6 chains/system * 3 breakout units/chain * 8 cells/breakout unit = 2592 cells.

Due to the relative cost of basic logic circuits compared to simple microcontrollers, the level of embodiment in this work is at the individual cellular automaton level. This means that the set of cells that make up each individual cellular automaton are using the same rules and will be

updated synchronously, while the edges of these cellular automata will be cross-connected to allow the edge cell state information to pass asynchronously without sharing rules or computational resources. The hardware consists mainly of two parts: a controller unit (shown in yellow in Figure 5.3), the central processing and communications component for each module in the system; and a breakout unit (shown in blue in Figure 5.3), to separate the serial data from the controller unit into parallel signals for each of the breakout unit's eight cells' actuators.

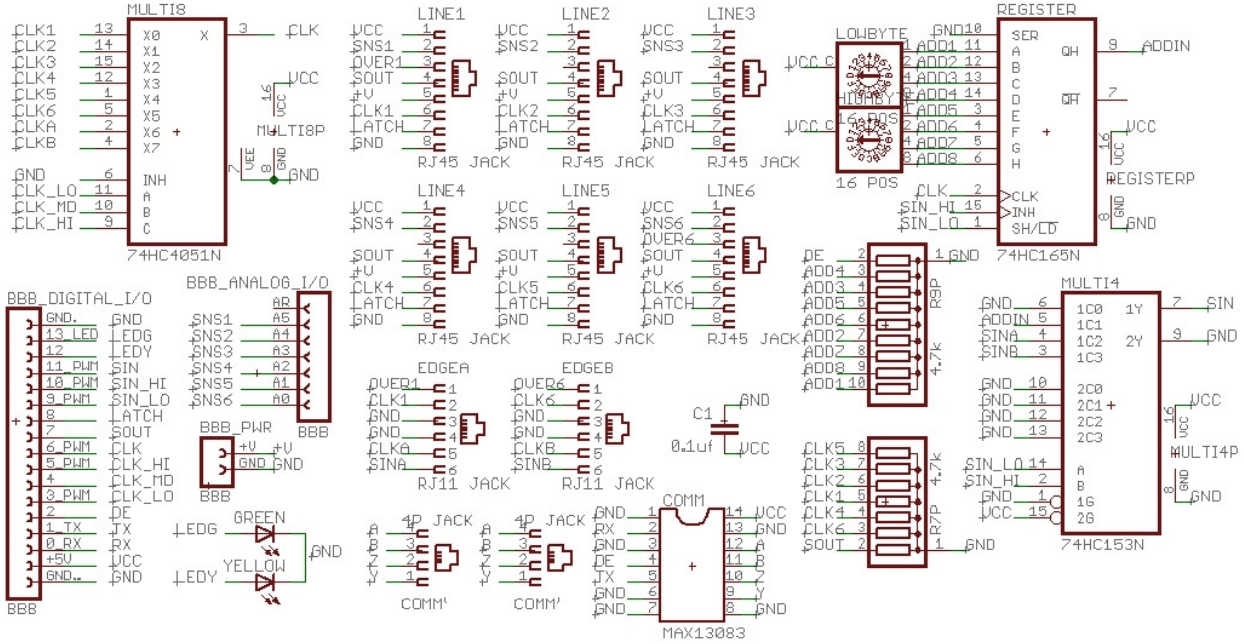

Figure 5.4. Schematic diagram of the custom hardware in the controller unit. This custom hardware, along with a Bare Bones Board from Modern Device, makes up the pair of yellow boards shown in Figure 5.3.

The controller units consist of custom designed hardware directly attached to an Arduino clone made by Modern Device, known as a Bare Bones Board (BBB) [73]. The custom hardware (schematic shown in Figure 5.4) allows the BBBs to each control a set of six cellular automaton cell columns of arbitrary length, read data from their neighbours on edge cell states, and communicate using RS485 with a manually changeable address. To achieve the data independence gained by using data structures in the library, shift registers are used as external memory units that can be written to by the local controller and read by neighbouring controllers. The control units each have the ability to keep track of local state changes as well as incorporating the states of the edges of both of its neighbouring controllers so as to reproduce the interconnection abilities of the library. The contents of the shift registers are controlled using a set of multiplexed serial output channels, while reading from neighbouring controllers is done using a similar smaller set of multiplexed serial input channels.

The breakout units each control eight cells, and consist of two custom boards (schematics shown in Figure 5.5) which connect to each other using a set of header pins: a memory board and a high-current driver board. The memory board has connectors for communication with the controller and with further daisy-chained breakout units, as well as a latching cascading 8-bit shift register that contains the state of all eight cells. In addition to a data pass-through for sensor signals, the memory boards can be chained to extend each column by eight cells at a time. The driver boards use the control signals from each memory board in combination with a high current source driver to power the individual cells. Thanks to this modular driver board, the cells can potentially power anything from simple LEDs to large motors by simply switching out the board for one of the right power.

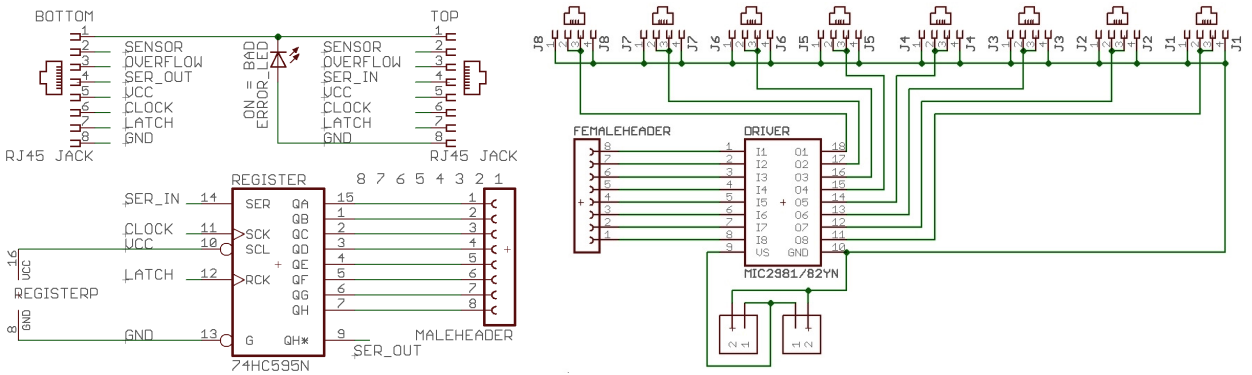

Figure 5.5. Schematics of the memory board (left) and the high-current driver board (right) that connect to form the breakout units. These two boards make up the pair of blue boxes from Figure 5.3.

In addition to this core hardware, a set of hanging SHARP GP2Y0A21YK infrared proximity sensors allow each embodied cellular automaton to sense the people that are moving below it. Each of the six columns of cells in a particular cellular automaton has a sensor hanging below it, and the presence of a visitor below a particular column of cells can affect the states of the bottom three cells in that column. With the movement of visitors below the installation, the changes in the states of the bottom cells creates the appearance of a boiling mass of light and vibration. Thanks to the restriction of the rules to only those that support the Game of Life 'glider', this boiling also occasionally will emit 'gliders' as if consuming energy from the movement of the visitors themselves to produce internal movement.

The outputs of the sensors were also initially going to be used to track the locations of the visitors within the installation to determine their interest in particular cellular automata. This was then to inform a fitness function for a genetic algorithm used to evolve the transition rules over time, with the idea that more interesting rules would attract more viewers, and the entire

sculpture would eventually evolve to rules that the visitors found interesting. However, due to the time constraints that existed on the design and installation, as well as the fact that the entire event only ran for a total 12 hours, the supervisory component that enables genetic algorithms to work was not incorporated into the actual construction of Aurora. This meant that there was no tracking of people within the installation, no genetic algorithm that needed a fitness function, and therefore no evolutionary aspect to the piece. To maintain the dynamic behaviour, the rules of the cellular automata were changed randomly at random intervals.

In this chapter, the use of the library, and by extension the framework, has been described in terms of an embodied cellular automata application in the form of an art installation. The specific design requirements for the installation dictated a need for specific aspects of cellular automata to exist that had not been developed until the creation of the framework in this research. How the framework is applied, and the interconnection graph that describes this application, has been laid out fully to demonstrate their direct use in a hardware application. The components of the library that have been adapted into hardware versions have been discussed, as well as the rest of the hardware that is used to enable the parts of the library that remain in software. In the next chapter of this dissertation, the conclusions of this work are described, including a discussion of how the main goal of this work was achieved and how well the library design criteria were successfully met. Along with these conclusions, a few topics are introduced as areas where future work should occur using the framework, the library, and some of the additional individual design elements that have been developed as well as some initial investigations inspired by 'Aurora'.

6. Conclusion

In the last few chapters, a number of novel topics have been discussed that contribute to both the field of cellular automata and to the larger field of distributed systems in general. A graph framework has been defined for the interconnection of information processing nodes, with a particular emphasis on the use of cellular automata as these nodes. A library that implements this framework, along with a number of other novel aspects, has been discussed and designed which will enable the application of these novel aspects in the research and industrial worlds. An application has been described that makes use of the library, and demonstrates a number of interesting uses of its novel aspects. The conclusions that have been reached throughout the time spent working on this thesis research are discussed below, while the next section will elaborate on some of the possible future work that this research enables.

The main goal of this thesis research has been achieved through the successful development of the interconnection graph framework in Chapter 3. The framework has enabled an exploration of the distributed computing spectrum from Chapter 1 by allowing cellular automata to behave as a representative member of the full range of parallel and distributed systems. This framework has also been developed to allow it to directly model the full range of these systems in a way which enables a theoretical analysis of the overall systems and their interconnections, in some cases using the well established field of graph theory. As an extension of this ability, any node within an interconnection graph can be replaced with another node, either built from a simpler or more complex system, that mimics the information manipulation of the first, without having to use the original underlying system.

To implement this framework, as well as to enable a practical exploration of the computing spectrum, the design of a library was discussed in Chapter 4 that was created based on a set of design criteria from Section 1.2. These criteria were developed to guide the design of the library to allow the use of the framework in research and applications throughout the computing spectrum on both hardware and software. There are five different criteria that the successful design of the library is judged upon, made up of two required criteria and three ancillary criteria. The primary criterion required the implementation of the framework discussed above, which was achieved for a subset of possible node types that focus on the use of cellular automata as interconnected system nodes. The secondary criterion required the library to be designed so as to

allow implementation on both software and hardware, which was successfully done and demonstrated in the application from Chapter 5 on a mix of hardware and software. Since both of these required criteria were met, further work on the achievement of the three optional ancillary criteria was performed.

The first of the ancillary criteria is the ease of combining the library with external systems, which was achieved in the form of a full breadth interface that takes into account the need for both low- and high-level measurement and manipulation of the systems implemented. The second ancillary criterion is based on the reproduction of typically available parameter sets in the field of cellular automata, and is demonstrated through the implementation of the Game of Life and other similar parameter sets in the library. Finally, the last of the ancillary criteria is the creation of new functionality above and beyond that typically found in other cellular automata libraries that are available, of which there are many examples in this library including the sleep mechanism and a wide range of possible interactive elements. This last criterion was also the inspiration for the ease of extendability in the library, which effectively allows the addition to the library of any new components that might be desired.

To fully demonstrate the use of the framework and the application of this successfully designed library, Chapter 5 describes a dynamic and interactive art installation created using aspects of the library on both hardware and software. This installation required the development of both the framework and the library to be designed as it was, and made use of aspects of the library that demonstrated each of the five criteria that were discussed above. It also provides a case study for an investigation of a particular region of the distributed computing spectrum, as the hardware can support a wide range of different system architectures with only minimal software changes required. Although not implemented on the opening evening of the installation, the development and use of genetic algorithms alongside an embodied application of the library provides a small peek into a much wider world of potential opportunities for further research. In the next sections, a number of different possibilities in this realm of future research, including the potential opportunities in genetic algorithms combined with embodied cellular automata, have been introduced. The possibilities provided are only a small subset of the full range of interesting problems and analysis that are enabled by using the framework on its own, the library, and their combination with external systems.

6.1. Aurora Inspired Future Work

A number of interesting and potentially valuable avenues were discovered after the installation of the Aurora piece that are worth mentioning. These designs and systems were briefly investigated in response to various issues that were encountered in the course of developing Aurora. They cover a number of different aspects of embedded design and human interaction, and can be categorized into the following core topics: synchronization, varied cell shapes in hardware, human interest tracking, and human-in-the-loop fitness functions for genetic algorithms.

Due to difficulties during construction of the installation, a number of connections were not made in the power and communications hardware, leading to the majority of the installation having effectively no communication network. As the sole job of the communication network was synchronization, this introduced the potential for a given controller to update a second time before its current edge states could be read by a neighbouring unit. This appeared to be quite rare based on an attempt at visually tracking cell states, but the imprecision and wide variety of interactions between rules make any visual tracking attempts difficult. Based on these difficulties, a number of possible methods of maintaining synchronization without a global clock signal were briefly investigated to find one that would be applicable to the embodied hardware that was used in this installation. All of the methods that were found are extensions or modifications of existing work in the realm of asynchronous cellular automata, primarily using a doubled set of states to prevent a secondary update from happening in a given cell until all of the cell's neighbours had been updated for a first time. The examination and implementation is recommended of one of the various methods of designing asynchronous cellular automata [74].

Along with the typical square shape for cells, hexagons and triangles should also be explored as potential cell shapes in hardware, and various methods of implementation should be investigated in terms of how hardware interconnections would work with these different shapes. Although it would appear at first glance that the square lattice is bound to be the standard, the only reason that it has been generally chosen over a hexagonal lattice up to this point is its ease of representation in a computer simulation. The hexagonal lattice actually makes far more sense as a hardware cell shape due to all of its neighbours having identical relationships with the central cell. This gives it a common neighbourhood under both Moore and von Neumann neighbourhood types, as described in Section 2.1. While the von Neumann neighbourhood always creates a common relationship between neighbours, both the square and triangular lattices have multiple

neighbour relationships in the Moore neighbourhood: edges and corners for the square lattice, and edges and two different types of corners in a triangular lattice.

Although these different neighbour relationships do not seem at first to be too much cause for concern, the situation changes once they are analyzed in terms of physical implementation. The multiple types of neighbours directly correspond to an equal number of physical connections that need to be designed into any hardware. Regardless of whether every single cell is being individually connected or multiple cells are grouped together and connected *en masse*, the different types of connections could cause a number of challenging issues to arise during both design and construction. These challenges in connections are the most difficult part of implementing cellular automata on distributed hardware. However, with the development of the framework and the library, a number of these connection issues have been resolved by allowing groups of cells to be implemented on one piece of more powerful hardware. In light of this ability, the connections between the hardware components can be made at a higher level of organization than is required when using simple hardware at an individual cell level. Because of this design feature, the hardware used in Aurora can also directly implement both triangular and hexagonal lattice shapes simply by changing the software on the control modules, as the level of communication is between cellular automata as opposed to individual cells or groups of cells.

One possible evolutionary fitness function which was mentioned in this work is based on the interest shown by visitors to various parts of the installation. In addition to the work described in this dissertation, a method of predicting interest in regions of a public installation has been developed as an extension of this hardware system and is described in full in Appendix A. Briefly, a height map was built that covered the installation space, similar to a topographical map of the various altitudes of mountainous terrain, using the distributed set of sensors below each controller unit to find the changes in height over time. This height map was then fed through a pattern recognition system that used estimated classifications of measurements into groups, such as “heads” or “shoulders”, to make a probabilistic estimate of the locations of any potential people. The estimated centre of each person, and the probabilistic mapping of all potential people, was then used to create and update particle filters. These were used to find a better estimate of visitor locations, as well as an initial estimate on their directions of motion. Based on averages of these particle filters, an estimate was made of the location and direction of each person. This system was then augmented with an interest prediction algorithm that created a

map, which changed over time, of what regions of the installation space were interesting. This overall method of predicting the interest of visitors in an artistic installation is potentially a key part of an expanded look at work in the field of interactive genetic algorithms.

6.2. General Future Work

Building on the successful achievement of both the main goal of this work and the library design criteria, the use of the novel aspects of this thesis research in some future work will be discussed in this section. This starts with a description of a number of possibilities in analyzing the robustness of hardware applications of cellular automata, and moves into the need for an investigation of the effects of specific types of interconnected systems. Following these potential opportunities, the combination of this work with the field of genetic algorithms is outlined with a discussion of the possible new gene attributes that will allow the discovery of solutions to tasks and problems, both old and new, that cannot currently be solved. Finally, some work will be described inspired by the issues that arose during the construction of 'Aurora', including some initial investigations that were carried out after the installation had finished have been described in a subsection.

One future possibility for research involves using the framework and library to analyze the robustness of information processing in embodied cellular automata in the face of various forms of hardware failures. For example, determining what the effects would be when the connections between one or more pairs of cells is broken, and whether these broken connections will affect only a local area or the entire cellular automaton. In hardware, these broken connections could appear as a constant state, or vary either randomly or in a predictable pattern. As another example, what are the effects if an actual cell becomes stuck in a single state, changes states randomly or predictably, or only updates every few time steps. How would the cell be detected, and what could be done to fix them? What can be done in these cases of cell-level issues to maintain the integrity of the system needs to be addressed, as well as how both the local and global dynamics will change and if these changes are dependent on the specific set of system parameters or are generally applicable? Simulating these broken connections would be as simple as replacing an edge in a custom interconnection graph with a scalar producer that will mimic the cell state(s) as desired. Or, at the library level, by replacing a cell structure with another similar structure that will perform the same form of mimicry.

On a higher level, there are further questions regarding the necessary levels of connection between entire regions of a cellular automaton. What would happen to a cellular automaton that had a set of connections broken all within one row, or in the centre of the system compared to at the edges? What if all of the cells in a particular region were stuck in one state, or oscillating between a few different states either randomly or predictably? Are these hardware issues going to affect the information processing that is being done only locally, or will they completely change the outcome? All of these questions have need of answers, and only through the application of the framework will the majority of them be able to be answered at the level required for computationally reliable results.

As an extension of the library itself, an intelligent sleep modification algorithm could be designed that would use an adaptable parameter to define how many time steps a cell needs to remain in the same state before it will be flagged as sleeping. This parameter could start at some initial moderately low discrete value and change over the course of time so that in active patterns it increased to create very little computational overhead, while in mostly static patterns it would decrease to create the same savings as seen currently when set to two time steps.

In terms of specific applications, an investigation should be made of the vast realm of various forms of mixed systems of interconnected cellular automata, including the three specific designs that were discussed in Subsection 4.1.1: mixed dimension systems, where there is a mix of different dimensions of cellular automata; hybrid systems, where cellular automata with different sets of parameters are interconnected; and spatially abstracted systems, where the interconnections between cellular automata are impossible to construct in a spatially consistent way. For example, a mixed dimension system where a number of 1D cellular automata are used as the boundary conditions for a set of interconnected 2D systems could allow a simple form of human readable input to translate into vastly complex information processing. Or as an example of a hybrid system, cellular automata could be interconnected using different shapes, to determine the effects of various axes of symmetry between systems. All of these forms of interconnected systems could be analyzed for potential uses in information processing.

With the addition of genetic algorithms to this work, a wellspring of potential research is created that builds on the already existing work in their combination with more typical cellular automata as discussed in Subsection 2.3.3. There have been only very preliminary investigations made of how the use of interconnection affects the evolution of genes in genetic algorithms, in large part

due to the vast realm of possible genetic attributes that can include not only the parameters of every interconnected system but the actual interconnections between systems themselves. For instance, a chromosome could be specified using only the ability to modify the interconnections in an interconnection graph and the rules of each cellular automaton node. The solutions found using this chromosome will need to be compared to those found when using only the rules, or only the interconnections. As an example, a larger version of a chequerboard problem where increasingly larger blocks of cells are desired is an excellent candidate for this type of exploration. With a block size of only one cell, the solution is simply the same as with a single cellular automaton, but as soon as the size is increased there is likely a need for using interconnection to allow hybridity between cells. It also needs to be determined how these changes will affect the solution when the objective of the fitness function is based on a global outcome instead of a local effect. If the goal is to find the best individual set of parameters for one cellular automaton, it should be determined if using an interconnected set of cellular automata will help or hurt the search.

References

[1] E.R. Banks, "Universality in Cellular Automata", in Proc. FOCS, 1970, pp.194-215.

[2] B. Chopard, "Cellular Automata Modeling of Physical Systems", presented at Encyclopedia of Complexity and Systems Science, 2009, pp.865-892.

[3] P. Maji, C. Shaw, N. Ganguly, B.K. Sikdar, and P.P. Chaudhuri, "Theory and Application of Cellular Automata For Pattern Classification", presented at Fundam. Inform., 2003, pp.321-354.

[4] P.L. Rosin, "Training Cellular Automata for Image Processing", presented at IEEE Transactions on Image Processing, 2006, pp.2076-2087.

[5] S. Nandi, B.K. Kar, and P.P. Chaudhuri, "Theory and Applications of Cellular Automata in Cryptography", presented at IEEE Trans. Computers, 1994, pp.1346-1357.

[6] W.K. Mason, "Art from Cellular Automata and Symmetrized Dot-Patterns", presented at Computers & Graphics, 1992, pp.439-441.

[7] S. Wolfram, “Cellular Automata and Complexity: Collected Papers”, Addison-Wesley, 1994.

[8] T. Toffoli and N. Margolus, “Cellular Automata Machines”, The MIT Press, Cambridge, MA, 1987.

[9] M. Zawidzki, "A Cellular Automaton Controlled Shading for a Building Facade", in Proc. ACRI, 2010, pp.365-372.

[10] K. Cattell and J.C. Muzio, "Synthesis of One-Dimensional Linear Hybrid Cellular Automata", presented at IEEE Trans. on CAD of Integrated Circuits and Systems, 1996, pp.325-335.

[11] J. von Neumann, “Theory of Self-Reproducing Automata,” Edited and completed by A. W. Burks, University of Illinois Press, Urbana and London, 1966.

[12] “Mirek's Cellebration – 1-D and 2-D Cellular Automaton Viewer, Explorer, and Editor”, http://www.mirekw.com/ca/index.html.

[13] A. Tovar, N. M. Patel, Amit K. Kaushik, and J. E. Renaud, “Optimality Conditions of the Hybrid Cellular Automata for Structural Optimization”, AIAA Journal 45 (3), 2007, pp. 673-683.

[14] A. Adamatzky, “Game of Life Cellular Automata”, Springer Publishing Company, Inc, 2010.

[15] M. Mamei, A. Roli, and F. Zambonelli, “Dissipative Cellular Automata As Minimalist Distributed Systems: A Study On Emergent Behaviors”, in Proc. of PDP'2003. pp.250~257, 2003.

[16] E.D. Adamides, P. Tsalides, and A. Thanailakis, "Hierarchical Cellular Automata Structures", presented at Parallel Computing, 1992, pp.517-524.

[17] S.M. Ulam, “On Some Mathematical Problems Connected with Patterns of Growth of Figures”, Proc. Symp. Appl. Math, Vol. 14, 215-224, 1962.

[18] E.F. Moore, “Machine Models of Self Reproduction”, Proc. Symp. Appl. Math,. Vol. 14, 1962.

[19] E. F. Codd, “Cellular Automata”, Academic Press, Inc. New York and London 1968.

[20] J.D. Farmer, T. Toffoli, and S. Wolfram, (eds). “Cellular Automata: Proceedings of an Interdisciplinary Workshop”, Los Alamos March 7-11, 1983 (Physica D 10). North Holland. 1984

[21] E.R. Banks, “Information Processing and Transmission in Cellular Automata”, Ph.D. thesis Department of Mechanical Engineering, MIT (1971)

[22] E.R. Berlekamp, J.H. Conway, and R.K. Guy, “Winning Ways for your Mathematical Plays”, Academic Press, vol. 2, chapter 2, 1982.

[23] M. Gardner, "Mathematical Games - The Fantastic Combinations of John H. Conway's New Solitaire Game Life," Scientific American 223, 120-123, 1970.

[24] P. Chapman, "Life Universal Computer", http://www.igblan.free-online.co.uk/igblan/ca/.

[25] D.I. Bell, "HighLife - An Interesting Variant of Life", http://www.tip.net.au/~dbell/, 1994.

[26] D. Millen, "Cellular Automata Music", in S. Arnold, G. Hair (eds), Proc. of the 1990 Int. Computer Music Conf., Int. Computer Music Association, San Francisco, pp. 314-316, 1990.

[27] D. Millen, "Generation of Formal Patterns for Music Composition by Means of Cellular Automata", in A. Strange, (ed) Proc. of the 1992 International Computer Music Conference, International Computer Music Association, San Francisco, pp. 398-399, 1992.

[28] E.R. Miranda, "The Art of Rendering Sounds from Emergent Behaviour: Cellular Automata Granular Synthesis", in Proc. EUROMICRO, 2000, pp.2350-2355.

[29] J. Flury and D. Bisig, "Celerina- A Generative Music System Using Aesthetical Reduction Applied to Simple Cellular Automata", in Proc. FLAIRS Conference, 2006, pp.237-242.

[30] D.A. Ashlock and J. Tsang, “Evolved Art via Control of Cellular Automata”, in Proc. IEEE Congress on Evolutionary Computation, 2009, pp. 3338-3344.

[31] N. Packard and S. Wolfram, "Two-Dimensional Cellular Automata," Journal of Statistical Physics 38, 901-946, March 1985.

[32] S. Wolfram, “A New Kind of Science”, Wolfram Media Inc., Champaign, IL, 2002.

[33] M. Cook, "Universality in Elementary Cellular Automata," Complex Systems 15, Number 1, 2004.

[34] J. Hanson, “Computational Mechanics of Cellular Automata,” Ph.D. dissertation, University of California, Berkeley, 1993.

[35] J. E. Hanson and J. P. Crutchfield, “Computational Mechanics of Cellular Automata: An Example”, Santa Fe Institute Working Paper 95-10-95, 1995.

[36] H. Hamann and H. Worn, “Embodied Computation”, Parallel Processing Letters, Vol. 17 (3), pp.287-298, 1997.

[37] S. Bandini, A. Bonomi, G. Vizzari, and V. Acconci, "An Asynchronous Cellular Automata-Based Adaptive Illumination Facility", in Proc. AI*IA, 2009, pp.405-415.

[38] S. Bandini, A. Bonomi, G. Vizzari, and V. Acconci, "A CA-Based Self-organizing Environment: A Configurable Adaptive Illumination Facility", in Proc. PaCT, 2009, pp.153-167.

[39] S. Bandini, A. Bonomi, G. Vizzari, and V. Acconci, "A Cellular Automata-Based Modular Lighting System", in Proc. ACRI, 2010, pp.334-344.

[40] A. Stauffer and M. Sipper, "Emergence of Self-Replicating Loops in an Interactive, Hardware-Implemented Game-of-Life Environment", in Proc. ACRI, 2002, pp.123-131.

[41] M. Mitchell, “An Introduction to Genetic Algorithms”, MIT Press, 1996.

[42] H. Takagi, “Interactive Evolutionary Computing: Fusion of the Capabilities of EC Optimization and Human Evaluation”, Proceeding of the IEEE, Vol. 89, No. 9, 2001, pp. 1275-1296.

[43] J.P. Crutchfield, M. Mitchell, and R. Das, “The Evolutionary Design of Collective Computation in Cellular Automata”, Evolutionary Dynamics: Exploring the Interplay of Selection, Neutrality, Accident, and Function, Oxford University Press, NY, 2003, pp. 361-411.

[44] M. Mitchell, P.T. Hraber, and J.P. Crutchfield, “Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations”, Complex Systems, Vol. 7, pp.89-130, 1993.

[45] M. Mitchell, J. Crutchfield, and P. Hraber, “Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments,” Physica D, vol. 75, pp. 361–391, 1994.

[46] R. Das, M. Mitchell, and J.P. Crutchfield, “A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata”, in Y. Davidor, H.P. Schwefel, and R. Manner (eds), “Parallel Problem Solving from Nature”, PPSN III, in Lec. Notes in Com. Sci., Vol. 866, Springer-Verlag, Berlin, 1994, pp. 344-353.

[47] M. Mitchell, J.P. Crutchfield and P.T. Hraber, “Dynamics, Computation, and the 'Edge of Chaos": A Re-Examination”, In G. Cowan, D. Pines, D. Melzner (eds), “Complexity: Metaphors, Models, and Reality”, Reading, MA: Addison-Wesley, 1994.

[48] M. Mitchell and J.P. Crutchfield, “The Evolution of Emergent Computation”, in Proc. of the National Academy of Sciences, USA, 92 (23): 10742, 1995.

[49] R. Das, J.P. Crutchfield, M. Mitchell, and J.E. Hanson, “Evolving Globally Synchronized Cellular Automata”, in L. J. Eshelman (ed), Proc. of the 6th Int. Conf. on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann, 1995.

[50] M. Mitchell, J. Crutchfield, and R. Das, “Evolving cellular automata with genetic algorithms: A review of recent work,” in Proceedings of the First International Conference on Evolutionary Computation and its Applications (EvCA’96). Russian Academy of Sciences, 1996.

[51] W. Hordijk, J.P. Crutchfield, and M. Mitchell, “Embedded Particle Computation in Evolved Cellular Automata”, in Proc. of PhysComp96, Boston, MA, 1996.

[52] W. Hordijk, J. P. Crutchfield, and M. Mitchell, “Mechanisms of Emergent Computation in Cellular Automata,” in Proc. of the 5th Int. Conf. on Parallel Problem Solving From Nature—PPSN V, A. E. Eiben, (ed.) Springer, NY, 1998.

[53] M. Mitchell, “Computation in Cellular Automata: A Selected Review”, in T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari (eds.), Nonstandard Computation, pp. 95-140, VCH Verlagsgesellschaft, Weinheim, 1998.

[54] M. Mitchell, J.P. Crutchfield, and R. Das, “Evolving Cellular Automata to Perform Computations”, in T. Back, D. Fogel, and Z. Michalewicz (eds.), Evolutionary Computation, Oxford University Press, 1998.

[55] J. Werfel, M. Mitchell, and J.P. Crutchfield, "Resource Sharing and Coevolution in Evolving Cellular Automata", presented at IEEE Trans. Evolutionary Computation, 2000, pp.388-393.

[56] S. Inverso, D. Kunkle, and C. Merrigan, “Evolutionary Methods for 2-D Cellular Automata Computation”, 2002.

[57] R. Breukelaar and T. Bäck, "Evolving Transition Rules for Multi Dimensional Cellular Automata", in Proc. ACRI, 2004, pp.182-191.

[58] T. Bäck, R. Breukelaar, and L. Willmes, "Inverse Design of Cellular Automata by Genetic Algorithms: An Unconventional Programming Paradigm", in Proc. UPP, 2004, pp.161-172.

[59] T. Bäck and R. Breukelaar, "Using Genetic Algorithms to Evolve Behavior in Cellular Automata", in Proc. UC, 2005, pp.1-10.

[60] R. Breukelaar and T. Bäck, "Using a Genetic Algorithm to Evolve Behavior in Multi Dimensional Cellular Automata: Emergence of Behavior", in Proc. GECCO, 2005, pp.107-114.

[61] A. Adamatzky, “Collision-Based Computing”, Springer-Verlag, London, 2001

[62] E. Sapin, L. Bull, and A. Adamatzky, "A Genetic Approach to Search for Glider Guns in Cellular Automata", in Proc. IEEE Congress on Evolutionary Computation, 2007, pp.2456-2462.

[63] A. Wuensche, "Self-Reproduction by Glider Collisions: the Beehive Rule", Alife9 Proceedings, pp. 286-291, The MIT Press, 2004.

[64] “Cellular Automata – Optimisation”, http://cell-auto.com/optimisation/.

[65] “An Implementation of Conway's Game of Life”, http://dotat.at/prog/life/life.html.

[66] Carter Bays, “Cellular Automata Home Page”, http://www.cse.sc.edu/~bays/CAhomePage.

[67] “Hexatron: A Cellular Automaton”, http://www.hexatron.com/hexca/index.html.

[68] “Processing.org”, http://processing.org/.

[69] “Cellular Automata Rules Lexicon – Life”, http://www.mirekw.com/ca/rullex_life.html.

[70] J.R. Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection”, The MIT Press, Cambridge, MA, 1992.

[71] “ITP Algorithmic Art >> Life Tower”, http://itp.nyu.edu/sigs/algorithmicart/life-tower/.

[72] “Arduino – HomePage”, http://arduino.cc/.

[73] “Bare Bones Board Kit | Modern Device”, http://shop.moderndevice.com/products/bbb-kit.

[74] B. Schonfisch and A. de Roos, “Synchronous and Asynchronous Updating in Cellular Automata”, Biosystems, Volume 51, Issue 3, September 1999, Pages 123-143

Appendix A

Tracking Fields of Interest in Large Scale Art Installations

Submitted for course credit in ME 780 Topic 7: Autonomous Mobile Robots

Tracking Fields of Interest in Large Scale Art Installations

Brandon J. DeHart, *Member, IEEE*

***Abstract*—Finding interesting cellular automata through evolutionary computing depends on how well the fitness function can gauge interest. When standard tracking methods fail, a method is needed to track moving people in order to extract their interest. A low resolution height graph is generated and used to produce a probability map. Based on this map, particle filters are created to track fields of interest. An average success rate of 92% is found using this method.**

I. Introduction

This work has developed as an answer to a question brought forward through a number of large scale art installations done in concert with Philip Beesley Architect Inc. (PBAi) in various locations around the world. The main goal of these collaborations is the creation of interactive art as a breeding ground for new forms of design in architecture and engineering.

The first of these, the Hylozoic Series, has had a number of generations displayed all around the world. The most recent of these are installations in: the Mois Multi Festival, Quebec City, Canada; the Festival de Mexico, Mexico City, Mexico; and representing Canada at the Venice Architecture Biennale, Venice, Italy. The Hylozoic Series are described as "artificial responsive forests with organic movements, embedded intelligence, and ongoing chemical reactions."

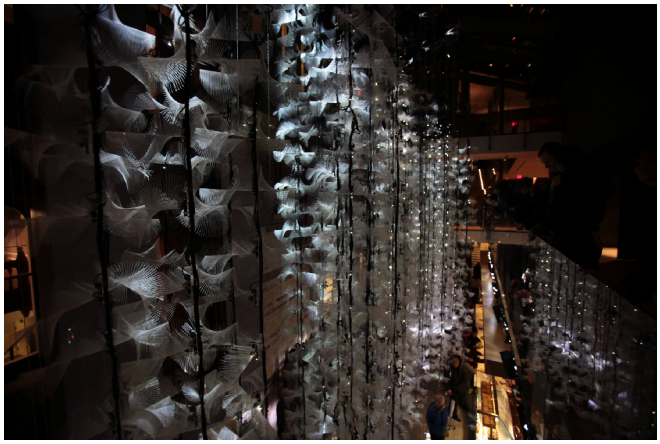

Fig. 1. A side view of the Aurora installation hanging in the atrium of the Royal Conservatory of Music in Toronto for Nuit Blanche 2010.

Another series of installations that was just debuted at Nuit Blanche 2010 in Toronto (see Fig. 1) is called the Aurora Series, and has been described as an "environment for human-aware artificial life within hanging columns of light, movement, and sound." The work in this paper is based mainly on developing needs in the Aurora Series.

Manuscript received December 4, 2010.

Brandon J. DeHart is a MASc candidate in the Electrical and Computer Engineering Dept. at the University of Waterloo, 200 University Avenue West, Waterloo, ON, Canada, N2L 3G1 (e-mail: bjdehart@uwaterloo.ca).

Some recent interest has been generated in using cellular automata (CA) for problem solving and real world applications ([1], [2]), due to the recent explosion of parallel processing systems. However, there is no guaranteed method of designing an environment, or a set of rules within a specific environment, that will allow the CA to accomplish some desired function. In the interest of solving this problem for use in artistic installations, a possible method for finding various CA that will be useful in a specific way is evolutionary computing.

Evolutionary computing, also known sometimes as genetic algorithms, is a well known optimization method for large and/or difficult search spaces such as that of finding good CA with which to solve real problems ([3], [4]). They consist of evolving and mutating a set of genes (specifying system parameters) towards a global goal, defined by a fitness function. The fitness function itself is a measure of how successful the set of genes are, and so finding the fitness function is the main goal of much of the work in this field.

In the case of the Aurora Series, each hanging column contains and runs a Cellular Automaton which interacts with those around it. In order to find which column patrons find the most interesting (and least interesting), the columns are used as a population (each column is a specific set of genes) within which to evolve and change CA rules with evolutionary computing. Thanks to the physically distributed nature of the genes throughout the installation, the fitness function will be defined as how interesting a specific region (and therefore set of CA) are to the people passing by. To find these interesting regions, it is necessary to first find what the people are looking at within the installation over time. It is this problem that is described in the following Section.

II. Environment

In order to find what regions of the installation people are actively interested in, there is a need to discover first where the people are within the installation. There is a lot of work dedicated to the problem of tracking people in an open space. However, to my knowledge, all of these methods use a style of sensing that is unavailable to use in the case of these installations.

The various reasons that these sensing styles will not work in this case will be discussed in Part A of this Section, along with a sensing solution that will work for this installation.

Due to the difficulty and inherent hardware problems that exist in large installations such as these, a physical test bed does not exist leading to the need for a functional simulation which is discussed in Parts B and C of this Section.

A. Real World

The majority of the work in tracking human movement ([5], [6], [7]) consists of attempting to recognize people in video sequences. Unfortunately, due to the large scale of the installations and the low-hanging elements involved, cameras from the top or the sides will not be able to see much detail other than peripheral information. These could be used as counters to inform the system of when people enter and exit the area of interest, but no more than that.

Some other work ([8], [9]) focuses on tracking people using heat, with an infrared camera, in order to ignore small physical obstructions such as in these installations. However, due to the use of high-heat activated Shape Memory Alloys actuators in this series, the body heat of people will be heavily masked from any angle.

Finally, a possibility that isn't mentioned in any previous work is to use floor mats with built in pressure sensors in order to track people's movements. In some installations this could work, but most of the instances of this series are placed in public spaces where there is no access or ability to cover the ground with anything.

Thanks to the problems listed above, the only real location that any sensing solution could exist is along the top of the space that the people themselves occupy. Since the installations are fairly large, this means some form of distributed sensing must take place. As the local electronics in a given region are nowhere near complex enough to handle video in addition to their other duties, a distributed set of simple sensors is needed.

This leaves a number of analog sensors as options, with the most economical and sensible being an infrared range sensor pointed directly at the floor from a known height in order to give a good estimate for the height of anything below. A distributed set of height measurements would also have the advantage of allowing distinction between specific people if they come together and separate again.

B. Simulated People

In order to develop and test this work, a simulated set of height measurements was created to allow for testing of a range of possible scenarios without the need for a physical installation to test in. These measurements were based on a set of simulated people moving in an area viewed from overhead by a distributed set of simulated range sensors.

Each of the people was randomly initialized with a set of values based on average size of a human as shown in Fig. 2. The ranges are: body width, 18” - 24”; head height, 5'4” - 6'6”; and shoulder height, 10” - 14” less than head height.

All people were modeled from above in 2 dimensions as: a head, represented by a circle; and shoulders, represented by an underlying ellipse. The head of each simulated person has a diameter of 10” and a color given by the head height. The shoulders have a minor axis diameter of 10” in the heading direction (forward) and a major axis diameter equal to the body width, while the color is given by the shoulder height.

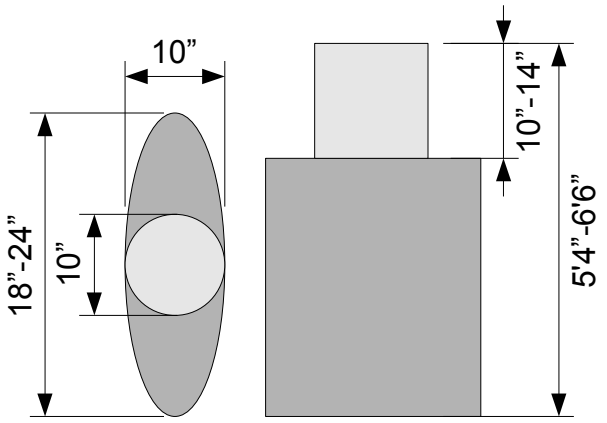

Fig. 2. Diagram showing the allowable size ranges of simulated people.

The state (***x***) of each person was governed by a simple model as shown in (1), where the position (*x, y*) is in inches and the heading (θ) is in radians. The inputs (***u***) are simply the velocity in the direction of the heading (v) in inches/second and angular velocity (ω) in radians/second.

$$\boldsymbol{x_t} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + v_t \cos(\theta_{t-1}) \\ y_{t-1} + v_t \sin(\theta_{t-1}) \\ \theta_{t-1} + \omega_t \end{bmatrix} + \varepsilon_t \tag{1}$$

The disturbance to the state (ε) is given as a multivariate Gaussian distribution with a diagonal covariance matrix with non-zero members listed in (2).

$$\Sigma_{xx} = \Sigma_{yy} = 0.01, \Sigma_{\theta\theta} = 0.0001 \tag{2}$$

The location and heading values were initialized at random from within the full set of possible values. As shown in (3), the input model is completely probabilistic to simulate a random human walking pattern in an open space.

$$\begin{gathered} \boldsymbol{u_t} = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} = \begin{bmatrix} v_{t-1} + dv_t \\ \omega_{t-1} + d\,\omega_t \end{bmatrix} \\ p(dv_t = N(0,1)) = 1\%, else\ dv_t = 0 \\ p(\omega_t = N(0,0.01)) = 1\%, else\ d\,\omega_t = 0 \end{gathered} \tag{3}$$

The velocity was initialized to a random value from 6” to 36” per second, while the angular velocity was randomly chosen from $-\pi/10$ to $\pi/10$ radians per second. These values were determined heuristically to simulate human motion.

C. Simulated Sensors

To simplify the simulation and boundary formulations, the installation used is a large square, with distributed hanging points located at the intersections of an overlaid rectangular grid. The grid consists of squares 10” wide by 10” long, in order to ensure that a person's head can never be detected on more than one sensor at a time.

Each simulated sensor outputs a height measurement as given in (4), with the variance on the noise taken from experience with a number of infrared range sensors.

$$h_{x,y} = h_{x,y:true} + N(0, \Sigma_h), \quad \Sigma_h = 0.0001 \tag{4}$$

Once the sensors have found their measurements for the given time-step, they are self-sorted into various classes. These classes take into account the possibility of noise in the measurements and so have slightly expanded boundaries compared to what the possible ranges of heights are for the related people. There are 4 classes as follows:

- Head, if h > 5'10";
- Head or Shoulder, if 5'2" ≤ h ≤ 5'10";
- Shoulder, if 4' ≤ h < 5'2";
- Nothing, if h < 4'.

These classes are used in order to facilitate all forms of measurement interpretation as discussed in the next Section.

III. Measurements

There are two main aspects of data interpretation that come from the sensor readings, both of which are discussed in this Section. The first is some basic sensor clustering into groups, which is discussed in Part A. The second aspect, in Part B, is the construction of a probability map using both the raw sensor classes and the sensor groups. In order to reduce computational time, only active sensors (those in a class other than Nothing) are considered in both sensor groups and construction of the probability map. Both the sensor groups and the probability map play important roles in the initialization, iteration, and error correction of the particle filters that are discussed in the next Section.

A. Sensor Groups

Sensor groups are initially created using a basic system which iterates through all active sensors and groups them together based on euclidean distance as given in (5).

$$d = \sqrt{(S_{1,x} - S_{2,x})^2 + (S_{1,y} - S_{2,y})^2} \tag{5}$$

The distance must be less than the maximum width of a person. If two or more active sensors are found within range of each other, they are both assigned to a group. This initial grouping is later used as a hub to which unassigned active sensors can be added if they move within range. This also allows dead sensors within the group to be removed.

Rejection policies also exist in the groups in order to attempt to ensure that sensors detecting two different people are not in the same group. The simplest of these consist of each group only being able to have one sensor in the Head class, and all sensors in the group needing to be within half the maximum body width from the group center.

Once the groups have been formed and no currently active sensors are without a group, the group centers are assumed to be very likely locations for people to be in.

B. Probability Map

At each time-step, the probability map is initialized with a low, non-zero probability throughout before any distinct probability regions are added. The regions themselves are integrated into the map additively so that the probabilities can stack with each other. Once all regions have been added, the values are normalized using the highest value region.

The regions are each created based on using either sensors directly or the sensor groups found in the previous Part of this Section. There are high probability regions created and centered around each of the active sensors, with different shapes and sizes of region based on the various classes the sensors can have. The probability of a given point in a region due to a particular sensor is strictly a function of its distance from that sensor as is seen in (6).

$$d_S = \sqrt{(S_x - x)^2 + (S_y - y)^2} \tag{6}$$

Each of the different active sensor classes create a different high probability region. If the sensor is in the Head class, then the region is a circle with radius 5" as shown in (7). This is due to the fact that if the sensor reads a Head, then the furthest the center of the person can be is 5" from the sensor, due to a head having a diameter of 10".

$$p(x, y | d_S \le 5) = 90\% \tag{7}$$

If the sensor is in the Shoulder class, the region is a ring with an outer radius of 12" and an inner radius of 5" as shown in (8). Since a definite shoulder reading can only happen if the sensor is outside of the head but still within half the maximum body width, the region looks like a ring.

$$p(x, y | 5 < d_S \le 12) = 90\% \tag{8}$$

Finally, if the sensor is in the Head or Shoulder class, then the region is a circle with radius 12" as shown in (9). As the measurement could be either a head or a shoulder, the region is a union of the head and shoulder regions.

$$p(x, y | d_S \le 12) = 90\% \tag{9}$$

In addition to the individual sensors contributing high probability regions, the sensor groups create regions of their own that always overlap with their associated sensors. These are also strictly a function of distance, but in this case it is the distance from a given point to the group as seen in (10).

$$d_G = \sqrt{(G_x - x)^2 + (G_y - y)^2} \tag{10}$$

There are only two distinct regions that are created from the sensor groups: one when the group has 1 active sensor, and one with 2 or more active sensors. In the case of 1 active sensor, similar reasoning to that of a Head or Shoulder class sensor applies to create a 12" radius circle, as seen in (11).

$$p\left(x, y | d_G \le 12,\ G_{size} = 1\right) = 90\% \tag{11}$$

In the case of 2 or more active sensors, the distance of the group center from any valid person's location will be a maximum of 7", giving a 7" radius circle as shown in (12).

$$p(x, y | d_G \le 7,\ G_{size} \ge 2) = 90\% \tag{12}$$

In order to clarify how the regions interact once they are integrated into the probability map, a couple of possible combinations are shown in Fig. 3. These are just two of many different possible combinations that can occur.

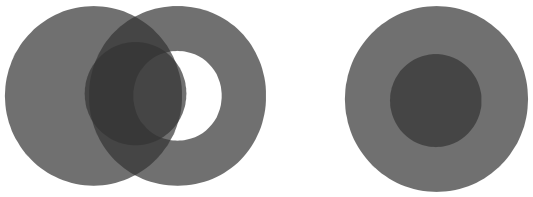

Fig. 3. Diagram showing two possible combinations of probability regions, where darker regions indicate higher probability. Left: a Shoulder or Head sensor and a Shoulder sensor side by side in a group together. Right: a Head sensor in a group of its own.

IV. Particle Filters

The use of particle filters instead of any of the other possible sensor filters for this work was mostly based on the non-linear nature of both the measurement model and the motion models used. Since these typically require a number of modifications that will remove guarantees of optimality, a solution was found which uses a set of particle filters. Also, since this work will at some point be implemented on a distributed set of microprocessors, an inherently discrete solution at multiple levels provides many methods of division of labour in order to allow real-time computation.

One of the main limitations in particle filters is the issue of particle deprivation. This can happen when there is little to no new information presented, and results in the particle filter arbitrarily focusing all of its particles in one small area. This focus can cause the filter to ignore future conflicting information. Unfortunately, in a number of different possible positions a person can be setting off none of the sensors in their region. Other times a person may stop, or be spinning in place, which will lead to a constant sensor reading. Both of these scenarios would be an issue if only one particle filter was being used to track all people in the installation.

In order to avoid this issue and make deprivation a useful attribute of the particle, one filter is created and assigned to each likely person given by the center of a sensor group. Since the filter will be narrowly distributed and the probability map in the vicinity of a sensor group will encourage an even narrower distribution, deprivation of the filters will actually lead to a better estimate of that particular person's state within the installation.

The remainder of this Section will deal with the following aspects of each particle filter: the initialization in Part A, updating and resampling in Part B, estimating the overall state in Part C, and any needed error correction in Part D.

A. Initialization

Each particle filter is initialized based on a new sensor group being found, and the filter is associated with the group in order to allow for later error correction. The locations of particles within the filter are normally distributed around the group center while the heading of each particle is randomly chosen from the full range as shown in (13).

$$\boldsymbol{x}_{p,0}=\begin{bmatrix} G_x+N(0,9) \\ G_y+N(0,9) \\ random(0,2\pi) \end{bmatrix} \tag{13}$$

The inputs to each particle are initialized in the same range as the input model for the people, as shown in (14).

$$\boldsymbol{u}_{p,0}=\begin{bmatrix} random(6,36) \\ random(-\pi/10,\pi/10) \end{bmatrix} \tag{14}$$

B. Update

The state update of the particles is done using the same kinetic model as the simulated people as given in (1,2) so it will not be repeated here. The change in inputs that the people will use are unknown though, so the input update for the particles in a given filter will have a zero-mean, normally distributed, additive disturbance as shown in (15).

$$\boldsymbol{u}_{p,t}=\begin{bmatrix} v_{p,t-1}+N(0,1) \\ \omega_{p,t-1}+N(0,0.01) \end{bmatrix} \tag{15}$$

Once the motion model update has been completed, each particle is given a weight taken from the probability map based on their estimated location. Based on these weights, a cumulative weight density function is created from all particles in a given filter. This function is uniformly sampled and evaluated in order to build a new set of particles from the old. This process is known as resampling and will allow for the old distribution of particles to more closely match the true state as modified by the newly found measurements.

C. Estimate

The resampled particles are used to find an estimate for each filter's overall state, given in the same way as the simulated people or particles. In order to do this, the state is assumed to be made up of independent Gaussian distributions, leading to the need only to find the mean and variance of each state variable. In the case of the location variables, this is fairly simply done with a basic average and simple variance as given in (16).

$$E[a]=\frac{1}{P}\sum_{p=1}^{P} a\,,\ Var(a)=E[a^2]+E[a]^2 \tag{16}$$

However, in the case of the heading variable the formulas in (16) do not hold thanks to the periodic nature of the values. In this case, a circular mean and variance are necessary. To accomplish this, the heading is assumed to be the angular component of a location on the unit circle in polar coordinates, which can easily be converted into a location in Cartesian coordinates using basic trigonometry. An average location can then be found in Cartesian coordinates which is converted back into polar coordinates.

The angular component of this polar location will be the correct average heading as shown in (17).

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_\theta \end{bmatrix} = \begin{bmatrix} E[x] \\ E[y] \\ atan2\left(E[\sin\theta], E[\cos\theta]\right) \end{bmatrix} \tag{17}$$

The radial component of the polar location will be a measure of how tightly grouped the headings were with a value of 1 being all equal and a value of 0 meaning uniformly distributed. In order to convert this value into a variance, it is subtracted from 1 as shown in (18).

$$Var(\boldsymbol{x}) = \begin{bmatrix} E[x^2]+\mu_x^2 \\ E[y^2]+\mu_y^2 \\ 1-\sqrt{E[\sin\theta]^2+E[\cos\theta]^2} \end{bmatrix} \tag{18}$$

The maximum distance of the estimated location compared to the simulated person being tracked was found to be 5", which is almost entirely due to the method of building the probability map and sensor groups using the assumption that a head is a 5" radius circle.

Even when a person stops within the installation, which is typically a problem for particle filters, the estimated location stays within 5" of the actual person. The only issue that arises from a stopped person is an increased variance on the heading, as without further sensor input there is no way to know if the person has stopped completely or is spinning.

Once an estimate is found, provided the error correction does not remove the filter from use, each filter's state estimate is used as the origin and direction of a field of interest as described in the next Section.

D. Errors

In order to ensure that the particle filter estimates are valid, various forms of error detection are applied. These errors, if detected, signal that the particle filter estimate is in one of three different states that all require that the filter be removed from use and reassigned in some way.

The first state occurs if an estimate has drifted more than 12" away from the center of its associated sensor group as shown in (19). When a filter estimate has moved this far away from its group, it is assumed that the filter is on a divergent path from that of the person being tracked. If a filter is found in this state, it is reinitialized using the location of its current sensor group.

$$\sqrt{(\mu_x - G_x)^2 + (\mu_y - G_y)^2} > 12 \tag{19}$$

The second state occurs when the estimate leaves the installation boundaries, which happens in the general course of use as tracked people leave the installation. As such, this is the most often detected error state. In order to detect when this is the case, one or more of the logical statements in (20) must be true. The variable E is a heuristically chosen value (5" is used) of how close to the edge an estimate should be before it is assumed that it will be leaving the installation.

$$\begin{aligned} &\mu_x < E && \&\ \cos(\mu_r + \pi) > 0 \\ &\mu_x > (x_{MAX} - E) && \&\ \cos(\mu_r) > 0 \\ &\mu_y < E && \&\ \cos(\mu_r - \pi/2) > 0 \\ &\mu_y > (y_{MAX} - E) && \&\ \cos(\mu_r + \pi/2) > 0 \end{aligned} \tag{20}$$

Finally, the third state occurs when the filter's associated group orphans it (this can happen when a person stops in a position between sensors) and the filter estimate is left in the middle of open space for a time. When this happens, the variances over the state become very large very rapidly and the total weight of all of the particles gets close to zero.

In both the second and third states, the filter in question is removed from use and placed into a list of free filters. These filters are then used when new sensor groups are created as new people enter the installation.

V. Interest Maps

The estimated interest map is built using the particle filters' state estimates for the locations and headings of people within the installation. Each estimate is only considered valid if the variance on the heading is less than 0.1, and the particle filter is associated with a sensor group.

If a filter passes both of these checks, then it is assumed that there is a trapezoidal field of interest projected from the estimated location in the direction of motion, the extremes of which are found based on the following assumptions: the viewing angle is 60°, centered around the estimated heading; the start of the field is 1' from the estimated location; and the depth of the field is 5' from the estimated location.

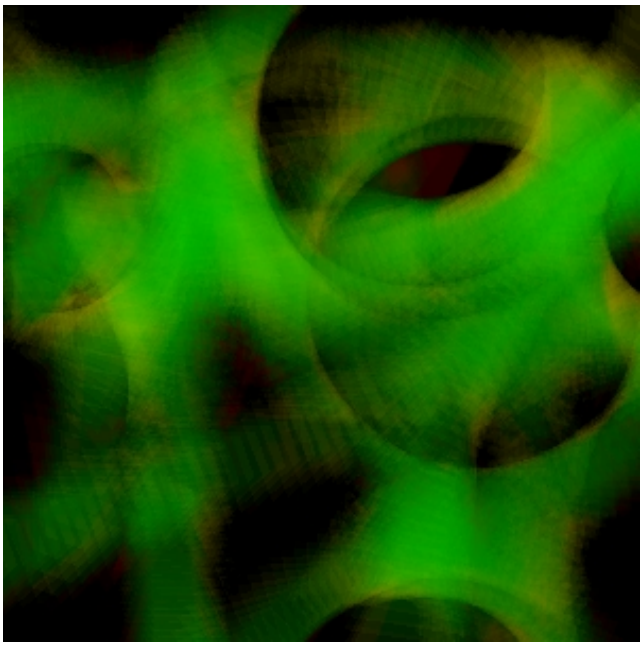

Fig. 4. Interest map comparison. In this image, the color representations are: green for true positive, red for false positives, yellow for false negatives, and black for true negatives.

The estimated interest map itself is made by integrating all of these predicted fields of interest over time. The fields are each treated as a medium probability region, which are then overlaid together with a slightly eroded copy of the previous map estimate in order to emphasize recent interest more than past interest. This estimated interest map will slowly change over time to show what regions of the installation the people currently find interesting and which ones they do not.

In a similar fashion, with the same assumptions, and using the simulated people's actual locations and headings, a comparison is made through direct subtraction of the actual interest map from the estimated one. The values on both maps are chosen so that errors can be distinguished into false positives (estimate shows interest when people do not) and false negatives (estimate shows no interest when people do), along with true positives and true negatives (estimate and real people agree on presence/lack of interest). These can be seen distinctly as shown in Fig. 4.

VI. Conclusion

As should be evident from Fig. 4, the performance of the work described in this paper surpassed expectations in terms of being able to track and predict the interest of people moving through a large scale installation. After a number of executions of the simulation and with various numbers of people, the average percentage of true interest points out of the overall interest map comparison was 92%. Also, of the errors, the false negatives were an average of 6% of the comparison, while false positives were only 2%.

Some future possibilities that are planned for this work include the incorporation of better pattern recognition in the measurement interpretation task in order to: create a more useful probability map, be able to initialize the filters with some form of subset of possible headings, and improve group rejection for when people are close to one another.

Another future improvement is the possibility of extending a single particle filter to track all people in a subsection of the installation and perform hand-offs between them in order to better correct for individual filter errors. This will allow for removing the dependence on the deprivation problem.

Finally, this work could definitely benefit from a number of improvements related to the interest fields themselves and the human motion data: a more realistic model of the human motion would improve testing and tracking of real people; using real data from an installation along with verification of the true motion would allow a much better correlation between measurements and truth; and a better idea of what the field of interest actually looks like based on a person's orientation would directly inform the fitness function.

Acknowledgment

I would like to thank Prof. Steve Waslander for some essential insights and extensions on this project that ensured on-time production and presentation of all deliverables.

References

[1] C. Burstedde, K. Klauck, A. Schadschneider, and J. Zittartz, “Simulation of Pedestrian Dynamics Using a 2-Dimensional Cellular Automaton,” Physica A, 295:507-525, 2001.

[2] M.Mamei, A.Roli and F.Zambonelli, ”Dissipative Cellular Automata As Minimalist Distributed Systems: A Study On Emergent Behaviors,” IEEE Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing, Genova (Italia), 2003.

[3] M. Mitchell, J. P. Crutchfield and R. Das, “Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work,” Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA '96). Moscow, Russia: Russian Academy of Science, 1996.

[4] A. Ugur, and M. Conrad, "Building Evolution Friendliness into Cellular Automaton Dynamics: The Cytomatrix Neuron Model," in Proceedings of the 1999 Congress on Evolutionary Computation (CEC99, Washington DC, USA, July 1999), Vol. 3, pp. 2071-2077, IEEE, Piscataway, NJ, 1999.

[5] L.Bazzani, D.Bloisi, V.Murino, “A Comparison of Multi Hypothesis Kalman Filter and Particle Filter for Multi-target Tracking,” 11th IEEE Int’l Workshop on Performance Evaluation of Tracking and Surveillance PETS 2009, Miami, FL, USA, June 2009.

[6] C. Chang, R. Ansari, A. Khokhar, "Multiple Object Tracking with Kernel Particle Filter," CVPR, Vol. 1, pp.566-573, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) – 2005.

[7] M. Perše, M. Kristan, J. Perš, G. Vučkovič, S. Kovačič, “Physics-Based Modeling of Human Motion Using Kalman Filter and Collision Avoidance Algorithm,” International Symposium on Image and Signal Processing and Analysis, ISPA05, Zagreb, Croatia, pp. 328-333, 2005.

[8] H. Nanda, L. Davis, “Probabilistic Template Based Pedestrian Detection in Infrared Videos,” Proc. IEEE Intelligent Vehicles Symposium 2002, Paris, France, 2002.

[9] E. Goubet, J. Katz and F. Porikli, "Pedestrian Tracking using Thermal Infrared Imaging", Mitsubishi Electric Research Laboratories, Technical Report, TR2005-126, 2005.