THESIS APPROVAL

The abstract and thesis of Davy Stevenson for the Master of Science in Computer Science were presented February 15, 2008, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS:

Melanie Mitchell, Chair

Karen Karavanic

Martin Zwick

DEPARTMENT APPROVAL:

Wu-chi Feng, Chair
Department of Computer Science

ABSTRACT

An abstract of the thesis of Davy Stevenson for the Master of Science in Computer Science presented February 15, 2008.

Title: Evolving Cellular Automata with Genetic Algorithms:

Analyzing Asynchronous Updates and Small World Topologies.

Langton states, "It has been frequently observed that the simultaneous execution of a single relatively simple rule at many local sites leads to the emergence of interesting and complex global behavior" [17]. It is this complex global behavior which is the main topic of interest for this thesis: can a network with only local connections between nodes run simple computations on each node in order to perform tasks that require global information processing?

Research studying the abilities and limitations of sensor networks has grown considerably in the last decade and cellular automata provide a simple framework with which to study these distributed systems. Cellular automata, especially when

extended to exhibit the small world property and update asynchronously, are an idealized model for studying the computation capabilities of a distributed sensor network.

The goal of this thesis is to investigate the effects of asynchronous updates and small world topologies on the evolution of rules to perform the density classification task. In analyzing the resultant rules, strategies including the previously known *default strategies*, *block expanding strategies*, and *embedded particle strategies*, as well as the newly discovered *messaging strategies*, were found. A theory behind the dynamics of the messaging strategy has been proposed, which explains how the rules take advantage of the small world links in order to more quickly disperse information through the network.

# EVOLVING CELLULAR AUTOMATA WITH GENETIC ALGORITHMS: ANALYZING ASYNCHRONOUS UPDATES AND SMALL WORLD TOPOLOGIES

by

DAVY STEVENSON

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Portland State University
2008

**Contents**

## List of Tables

**List of Figures**

**Nomenclature**

| | |
|---|---|
| CA | Cellular automaton |
| $N$ | Number of cells in a CA |
| $k$ | Number of possible states for each cell in a CA |
| $r$ | Radius of a CA |
| $\phi$ | Update function of a CA, also called a rule |
| $\eta_i^t$ | Neighborhood of cell $i$ at time $t$ |
| $\mathbf{s}^0$ | The initial configuration of a CA |
| $s_i^t$ | The state of cell $i$ at time $t$ |
| $T_{max}$ | Time limit given to the CA in order to perform the given task |
| $\rho_0$ | The density of the initial configuration |
| $\rho_c$ | The critical value for the density classification task |
| $C$ | The clustering coefficient of a network |
| $L$ | The degree of separation of a network |
| $p$ | The probability that any link in a CA will be rewired to a random cell in order to add the small world property |
| $q$ | The number of links rewired in a small world CA |
| $E$ | The number elite rules copied over to the next generation by the genetic algorithm |
| $G$ | The number of times the genetic algorithm is run for each case |

$I$ The number of initial configurations that the genetic algorithm tests each rule against

$R$ The number of rules in each generation of the genetic algorithm

$g$ The number of generations in the genetic algorithm

$m$ Number of locations within each child rule that are mutated

$\mathcal{F}_N^I(\phi)$ The fitness of rule $\phi$ on $I$ random initial configurations of size $N$ chosen using a density-uniform distribution

$\mathcal{P}_N^I(\phi)$ The performance of rule $\phi$ on $I$ random initial configurations of size $N$ chosen using an unbiased distribution

# 1 Introduction

With the expansion of computing power and the continued miniaturization of computing devices and sensors, technology is expanding to encompass network-centric computing in addition to the traditional centralized-control architecture of mainframes and personal computers. Examples of emerging network-centric devices include sensors for controlling homes, sensor networks used for surveillance and reconnaissance, communication networks and environmental sensor networks [4]. In many cases these networks continue to be controlled by a central unit, which will become increasingly burdensome as network sizes increase [6].

"It has been frequently observed that the simultaneous execution of a single relatively simple rule at many local sites leads to the emergence of interesting and complex global behavior" [17]. It is this complex global behavior which is the main topic of interest in this thesis: can a network running simple computations on each node perform tasks that require sophisticated collective information processing? It has been shown [5] that solutions with varying accuracies have been found for

certain cases of CAs, particularly locally connected, synchronous automata.

One problem facing network-based computing is the difficulty of performing certain tasks that are trivial with a centralized control unit. Two tasks often cited in the literature are the density classification task and the synchronization task [5, 20, 22, 26], which are defined in Sections  and . One example of research in this area involves using a genetic algorithm to evolve cellular automata to perform these tasks [5, 20, 22]. There are a few problems with applying this research to real-world applications, however. The first is that many networks of individual computing devices will not be synchronous in their updates — expanding the cellular automata to encompass asynchronous updates is necessary to address this problem.

One other issue is that the nearest-neighbor connectivity of traditional cellular automata is very limiting as the average distance between any pair of cells is large. Many organic networks with small average distances between nodes, such as brain networks, social networks, or the Internet, exhibit what is known as the small world property. Research has shown that networks with the small world property are more successful at both the density classification task and the synchronization task [26, 27].

This thesis extends previous research [5, 22, 20] on the evolution of cellular automata rules to perform the density classification task. In particular, the cellular

automata will be extended to update in an asynchronous manner and exhibit the small world property. A genetic algorithm will be run for each type of cellular automaton in order to evolve rules to perform the density classification task. These rules will then be evaluated for performance and the resulting rule strategies will be analyzed. The goal of this thesis is to investigate the effects of asynchronous updates and small world topologies on the evolution of rules to perform the density classification task.

## 2 Cellular Automata

A *cellular automaton* (CA) is a computational device originally proposed by John von Neumann [28]. It has been proven that some CAs are universal computers [31], though crafting rules in order to run specific computations is a difficult task and thus not often attempted. CAs have often been used as models in the areas of emergent computation and artificial life since they share many of the same properties as systems found in the natural world. Natural systems as diverse as brains, insect colonies, biological populations and immune systems all contain a large number of simple and homogenous components that interact without centralized control and with only limited communication [5].

CAs consist of a collection of $N$ cells, each of which is a finite state machine. Cellular automata can be defined in any dimension, though the focus here will remain on one-dimensional CAs. Each cell can be in any one of a finite number of states, the set of which is denoted $\Sigma$. Let $k = |\Sigma|$ indicate the number of possible states for each cell. A *cell* is uniquely specified in space and time by

its site number $i = 0, 1, \ldots, N-1$ and its state at time $t$, which is indicated by $s_i^t$, where $s_i^t \in \Sigma$. Each cell is connected to $2r$ other cells, where $r$ stands for *radius*, and in the standard locally connected case the neighbor cells consist of the $r$ closest cells on each side of a cell. Therefore cell $i$ is connected to cells $i-r, \ldots, i-1, i+1, \ldots, i+r$. The collection of the states of these $2r$ cells and the center cell itself $(s_{i-r}^t, \ldots, s_i^t, \ldots, s_{i+r}^t)$ comprises the *neighborhood* $\eta_i^t$ of cell $i$ at time $t$. The configuration of the CA at timestep $t$ is defined as the configurations of all the cell states, $\mathbf{s}^t = s_0^t s_1^t \ldots s_{N-1}^t$. The *initial configuration* of the CA is thus $\mathbf{s}^0$. Each cell follows a rule $\phi$ that is used to update the cell to its new state given the current state of the cell and its neighborhood, therefore $s_i^{t+1} = \phi(\eta_i^t)$.

## 2.1 Rules

A *rule* describes the update function of a cell given the state of itself and its $2r$ neighbors. Since a total of $2r+1$ cell values are evaluated (the cell and its neighbors), there are a total of $2^{2r+1}$ possible update values that must be described. A rule is thus a binary string of length $2^{2r+1}$ indicating a 0 or 1 for each possible state, which are ordered lexicographically.

The binary string encoding of a rule can be thought of as the 'chromosome' of a CA. It is populations of such binary strings that are evolved by the genetic algorithm. For even small values of $r$ the search space of the rules is extremely

large, $2^{128}$ for $r = 3$ for example, which is too large for any form of exhaustive search.

## 2.2 Asynchronous Cellular Automata

The standard CA updates in a synchronous manner — all the cells are updated according to the rule $\phi$ at exactly the same time. This provides for simple computation, but unfortunately the real world does not update in this way. Instead, parallel and asynchronous updates are the norm. When extending a CA to a real world application that is highly parallel and lacks global synchronization, as is often the case in current hardware configurations, the synchronous update step would add an additional burden. While it has been proved that asynchronous CAs can be used to perfectly simulate synchronous updates [1], these CAs require substantially more cells (anywhere from $O(N \log N)$ to $O(4N^2)$ where $N$ is the size of the CA) and extra time to finish the computation, which could be burdensome or infeasible. Instead, it would be ideal to find a CA rule that can perform collective computation with asynchronous updates. Unfortunately, studies of asynchronous CAs have typically been lacking for all but the most trivial issues [1].

The definition of an *asynchronous update* as is used in this research is as follows. During each discrete time step, each cell will update itself by polling the values of its neighbors and applying the update rule. However, the order in which the cells

are updated will be chosen at random. Each cell updates once in each timestep. If a cell's neighbor has already updated, the neighbor's updated value will be used when applying the rule, which is the biggest difference between synchronous and asynchronous updates.

Due to the fact that a cell uses its neighbors' current (and possibly updated) values during each update step, information may be lost. For example, suppose cells $A$ and $B$ are neighbors of each other. In timestep $n$, cell $A$ updates, and then cell $B$ updates. In timestep $n+1$, however, cell $B$ updates before cell $A$. In this situation cell $A$ will never see the first updated value of cell $B$, and that information is lost to cell $A$. Because of this information loss the overall performance of an asynchronous CA might be lower than the performance of a synchronous CA for some tasks. However, the difference in performance may be small enough that the added benefit of asynchronous updates as described earlier provides an overall gain.

## 2.3 Connectivity

The method used to connect the cells of a CA can have drastic results on the CA's performance on a given task. When taken out of the theoretical realm, the connection method also obviously affects the structure of the network and the physical layout. The most simple and common method of connectivity for a CA is

the *locally connected* method — in fact this method of connectivity is often part of the definition of a CA. In this case a cell is connected to the physically closest $r$ other cells on each side. This causes the automaton to form a regular lattice structure. While only one-dimensional CAs will be investigated in this thesis, this connection method can also be extended to two-dimensional CAs where, in addition to the cell itself, either the nearest four neighbors (north, south, east, west) are connected, which is called the *von Neumann neighborhood*, or the nearest eight neighbors (adding northeast, northwest, etc.) are connected, which is known as the *Moore neighborhood*.

The *clustering coefficient* $C$ of a network is defined as "the average fraction of pairs of neighbors of a node which are also neighbors of each other" [24]. $C = 1$ in a fully connected network, and $C = 0.6$ for a locally connected network with six neighbors, such as locally connected automata with $r = 3$. A random graph has a very low clustering coefficient[1], especially as the number of nodes grows. Studies have shown that many real-world networks show a much greater clustering coefficient than that of random networks [24]. It is important that a CA retain a high clustering coefficient, because these clusters of cells allow for more advanced computation to take place.

[1]For a random graph $C = z/N$ where $z$ is the number of neighbors and $N$ is the number of nodes in the network [24]

The *degree of separation* $L$ of a network is the average distance between any two nodes in the network. CAs that are locally connected exhibit long average distances between two random cells, which makes the transfer of information across the network slow. Very few network structures in the real world (social networks, sensor networks, neuron connections in the brain, the Internet, etc) show this highly localized connectivity. Instead, these networks allow for a much faster transfer of information due to a network structure providing a smaller degree of separation between nodes. This is embodied by the popular notion of 'six degrees of separation,' which states that there exists a chain of around six acquaintances between any two people in the world, even though the human population is larger than six billion [24]. While the actual number six might be an exaggeration, the underlying idea that a low degree of separation facilitates the movement of information through a network remains valid.

## 2.4 Small World Property

A network exhibits what is known as the *small world property* if it has a high clustering coefficient while also having a small degree of separation. These properties are retained even as the network size grows very large. It has been shown that a network exhibiting the small world property is much more successful at performing the density classification task than locally connected networks [26, 27].

The standard nearest neighbor connectivity of a CA can very easily be modified to generate a connection exhibiting the small world property. By taking each link between two cells and replacing it with a link to a random cell with some small probability $p$, the degree of separation is decreased dramatically. These shortcut links allow information to move through the CA at a much faster rate. It has been found that the degree of separation between nodes drops drastically after even a small number of reconnections, while the clustering coefficient remains almost constant even after many more reconnections are made. The degree of separation drops by half when $p = 0.001$ and drops by 80% when $p = 0.01$, while the clustering coefficient remains effectively equal to the clustering coefficient of a locally connected network for both cases [29].

In order to apply the reconnections while keeping the number of neighbors constant for each cell, some small adjustments must be made. If a link was simply rewired to a random cell, then the random cell would contain one too many neighbors, and another cell would contain one too few. Therefore the reconnections must happen in pairs in order to keep the number of neighbors of each cell constant. The following method is used in order to accomplish this task. Two links are chosen at random. Then the links are crossed — if the two links connected cells $(a_1, b_1)$ and cells $(a_2, b_2)$, then the new links connect $(a_1, b_2)$ and $(a_2, b_1)$.

One thing to worry about when randomly adjusting the links in a CA is the

accidental disconnection of the automaton into two or more sub-automata. This could happen if the number of links adjusted is greater than the number of neighbors a single cell has. This problem is thus avoided by keeping the number of adjusted links below the number of neighbors. In order to guarantee that the CA is never divided, a constant number of randomly chosen links $q$ will be rewired instead of using an actual random number generator to decide whether any one link will be rewired. This is still equivalent to rewiring each link with probability $p = q/Nr$, with $N$ and $r$ defined in Chapter .

# 3 Computational Tasks

When studying CAs two simple tasks are often used as a benchmark of success. These tasks are easy to describe and understand, and are generally trivial for systems with a centralized control unit. These tasks encompass the types of computation that would be useful for CAs to be able to perform and yet are not at all trivial for distributed networks.

## 3.1 Density Classification

Density classification is a task that is trivial in a system with a central control unit, but is difficult to design a CA to perform due to the limitation of local computation. In order to be able to correctly perform the density classification task with any high degree of accuracy, some form of collective behavior and information transfer must take place within the CA. The definition of the density classification task for binary CAs is as follows: if $\rho_0$ is the density[1] of $\mathbf{s}^0$, the initial configuration, then

[1]The density of a binary configuration is the number of cells in state 1 divided by the total number of cells. Thus a state of all 0s has a density of 0.0, and a state of all 1s has a density of 1.0. A configuration where exactly have of the cells are in state 1 and half are in state 0 has a

given $\rho_c$, the critical value, the CA must determine if $\rho_0 > \rho_c$ and if so all the cells should transition to a fixed point of all 1s, otherwise the cells should transition to a fixed point of all 0s. The CA is given a time limit of $T_{max}$ in order for the cells to transition into the fixed point. Therefore this task measures the density of the initial configurations. In this research $T_{max} = 2N$ and $\rho_c = 0.5$.

Due to the ambiguity of the desired result should there be an exact split, the number of cells in the CA is usually chosen to be an odd number. The current best rules for synchronous locally connected CAs have around an 85% accuracy rate. It has been proven that no two-state rule exists that can perfectly perform the density classification task [16], though solutions do exist for various subproblems; if the use of two CA rules is allowed then a perfect solution can be found [10], though critics of this method point out that the CA must be given the ability to count in order to know when to switch between the two rules [5]. It is not known what the maximum performance of a locally connected synchronous CA is on the density classification task.

## 3.2 Synchronization

Research has also focused on the task of synchronizing a CA (also known as the firefly task). The CA again starts in some initial configuration, and the goal

density of 0.5.

is to end up in a configuration alternating between the states of all 1s and all 0s. Therefore, the state of the CA at time $t$ should be all 1s, and the state at time $t+1$ should be all 0s. Accomplishing the synchronization task using a CA (particularly an asynchronous CA) would provide a stepping stone towards synchronizing sensor networks in a simple and reliable way. In this research the synchronization task was not studied directly, but would provide an interesting extension of this area of research.

# 4 Sensor Networks

Due to the fact that they exhibit parallel computing, CAs are becoming increasingly relevant as technology expands to include distributed computing models. The continued shrinking of computer components has brought the idea of a distributed sensor network into the forefront of technology. As the hardware devices used to form the nodes of a sensor network, which are called motes, become smaller and cheaper, the deployment of large numbers of motes with the express purpose of monitoring and perhaps affecting the surrounding environment will become more commonplace. Useful tasks for such sensor networks include geographical monitoring, environmental monitoring, military operations, helping the elderly, and emergency response [14, 30]. *Sensor networks* are systems of motes providing feedback from and possibly affecting the surrounding environment. Sensor networks may self-organize or co-ordinate autonomously. Due to the limitations of their small hardware footprint, they often have power constraints along with limited computing power and bandwith [30]. Research studying the abilities and limita-

tions of sensor networks has grown considerably in the last decade [14], though this research has been dominated by sensor networks controlled by a central unit. As sensor networks continue to grow, a central control unit will become a liability, as the complexity of the network will continue to grow with the number of motes [14].

CAs provide a simple framework with which to work with these distributed systems, freeing them from the requirements of a central control unit. The inherent simplicity of CAs would allow for even smaller hardware components, increasing the viability of a real-world application. CAs, especially when extended to exhibit the small world property, are an excellent way to study the computation capabilities of a distributed sensor network. Each cell of the CA does the computation of one mote, and by finding successful rules to tasks such as density classification the parallel aspect of the network can be taken advantage of and the central control unit can be eliminated.

While most of the research on CAs has focused on synchronized updates, sensor network implementations are rarely synchronized due to high power and computing requirements [14]. Thus, to successfully apply the CA rules to a distributed sensor network, studies on the capabilities of asynchronous CAs must be undertaken.

## 5 Genetic Algorithm

This research extends work by Mitchell et. al. [5, 22, 23] investigating the evolution of rules by genetic algorithms to perform the density classification task. First a repetition of Mitchell's previous results [5] with synchronous locally connected CAs was undertaken in order to validate the similarity of the approach. Then asyncronous locally connected CAs were studied. Next, CAs that exhibit the small world property were examined. Finally, CAs exhibiting both the asynchronous update step and the small world property were analyzed.

In accordance with previous research, the one-dimensional CA will have $N = 149$, $r = 3$, $k = 2$. The size of the CA lattice was chosen to be odd in order to avoid the issues surrounding an exact 50% density in the density classification task. Initially the neighborhood of a cell will consist of the cell itself and the three cells on each side of the cell. This will be termed the *locally connected case*. In the *small world case*, $q = 4$ links will be reassigned to a random cell in order to add the small world property to the CA. This corresponds to $p = 0.009$.

In order to simulate asynchronous communication, the cells will be updated in a random order at each timestep and the current state of the CA will be used to apply the update rule. This preserves the idea of a single update step, which allows for easy visualization of the time series, while still simulating the features of asynchronous communication. This update method will be used in the *asynchronous case*. This is compared with the *synchronous case*.

A genetic algorithm was used to evolve the rules in order to perform the density classification task. The genetic algorithm was run a total of $G = 300$ times for each case. Each run of the genetic algorithm contains $g = 100$ generations. The algorithm starts out with a population of $R = 100$ randomly generated rules. During every generation each rule is evaluated on a population of $I = 100$ randomly generated initial configurations. The rules and initial configurations are not chosen using an unbiased distribution[1], instead a density-uniform distribution is used[2]. It has been shown that attempting to run the genetic algorithm with an unbiased distribution of initial configurations provides for low genetic algorithm performance as the densities of such a distribution are clustered around 0.5 — the hardest cases for the CA rules to handle — and the evolution of the rules is slow [5]. Aiding the evolution by giving the rules tasks with easy densities around 0 and 1 speeds

[1]In an unbiased distribution each cell is assigned a random value of 0 or 1

[2]In a density-uniform distribution, a random density from [0,1] will be chosen and the cells will be assigned random values in order to meet this density.

things up and provides for a better pool of final rules. Similarly, an increase in genetic algorithm performance is seen if the rules are likewise not clustered around a density of 0.5 [5].

After running each rule on the initial configurations and recording the results of the fitness function $\mathcal{F}_{149}^{100}$ — which in this case is the correct classification defined in the density task (see Section ) — the rules are ranked in order of fitness. The fitness function used by the genetic algorithm is described in Section . After the rules are sorted by fitness the $E = 20$ highest performing rules are identified. These rules will be copied directly over into the next generation, which is a procedure known as elitism. The remaining $R - E = 80$ positions will be filled with the results of recombination between two randomly chosen parent rules from the elite group. Furthermore, after recombination $m = 2$ randomly chosen locations within each child rule will be mutated; if the location holds a 0 it will be switched to a 1 and vice versa.

## 5.1 Fitness and Performance

In order to calculate the ability of a particular rule to perform the density classification task it is necessary to have a concept of fitness and performance. The fitness $\mathcal{F}_N^I(\phi)$ of a rule $\phi$ is calculated by selecting $I$ random initial configurations of size $N$ using a density-uniform distribution and then running $\phi$ on each initial

configuration for $T_{max}$ timesteps. The fraction of the $I$ initial configurations that the rule $\phi$ classified correctly is then returned. An initial configuration is only considered classified correctly if a fixed point of all 1s or all 0s is reached by time $T_{max}$ and the answer is correct. No partial credit is given. A density-uniform distribution picks initial configurations with a uniform probability over $\rho_0 \in [0, 1]$. It has been shown [5] that choosing a distribution that selects initial configurations from a variety of densities allows the genetic algorithm to perform much more successfully. In this research the fitness function $\mathcal{F}_{149}^{100}$ is used.

In contrast, the performance $\mathcal{P}_N^I(\phi)$ of a rule $\phi$ is calculated by selecting $I$ random initial configurations of size $N$ using an unbiased distribution and again running $\phi$ on each initial configuration for $T_{max}$ timesteps and returning the fraction classified correctly. This is a much harder task for the rules as the densities of the initial configurations are all clustered around 0.5, which is the hardest case for the rules to classify. Thus performance gives a more accurate representation of a rule's accuracy at performing the density classification task given initial configurations around $\rho = 0.5$, and is also a lower bound on the performance of the rule. The performance function $\mathcal{P}_{149}^{10^4}$ is used in this research.

# 6 Results

Results were gathered through simulations run on four different cases: local synchronous, local asynchronous, small world synchronous, and small world asynchronous. Values used for the genetic algorithm can be found in Table 6.1.

## 6.1 Strategies

In studying the resultant rules evolved for each case, a few different strategies were seen. The default, block expanding, and embedded particle strategies have been documented by previous research [5, 20, 22] on the locally connected synchronous case. The messaging and unclassified atypical strategies were discovered when the connectivity of the CAs was extended to exhibit the small world property. The following subsections discuss the various strategies and display typical space-time diagrams for each. For each space-time diagram, the initial configuration is shown at the top of the diagram and time progresses in the downward direction. In a figure displaying two space time diagrams side-by-side, the diagram on the left

| | | |
|---|---|---|
| $N$ | Number of cells in the CA | 149 |
| $T_{max}$ | Number of time steps the CA is allowed to run in order to perform the given task | $2N$ |
| $k$ | Number of possible states for each cell | 2 |
| $r$ | Radius of the neighborhood of a cell | 3 |
| $\rho_c$ | The critical density of the density classification task | 0.5 |
| $G$ | Number of times the genetic algorithm is run for each case | 300 |
| $g$ | Number of generations in the genetic algorithm | 100 |
| $R$ | Number of rules evolved by the genetic algorithm | 100 |
| $I$ | Number of initial configurations each rule is performed on in each generation | 100 |
| $E$ | Number of elite rules transferred to the next generation | 20 |
| $m$ | Number of rule locations that are mutated in each child rule | 2 |
| $q$ | Number of links between cells that are randomly reconnected in order to provide the small world topology | 4 |
| $p$ | Percent chance that any particular link between two cells will be reconnected in order to provide the small world topology | 0.009 |
| $\mathcal{F}_N^I$ | Fitness of a rule when run on $I$ random density uniform initial configurations of size $N$ | $\mathcal{F}_{149}^{100}$ |
| $\mathcal{P}_N^I$ | Performance of a rule when run on $I$ random unbiased initial configurations of size $N$ | $\mathcal{P}_{149}^{10^4}$ |

Table 6.1: Table of parameter values used by the genetic algorithm.

expresses the low density case, while the diagram on the right expresses the high density case. The actual densities of the initial configurations can be found in each figure caption.

Even though the rule has $T_{max} = 2N$ time steps in order to perform the density classification task, only $N$ time steps are shown in order to limit the space taken up by graphs. The cases shown were picked in order to exhibit the desired behavior in $N$ time steps. In this work, the classification of the strategies was made by visual inspection of the highest fitness rule from each run of the genetic algorithm, for a total of 300 rules per case. Each rule was performed on six different randomly generated initial configurations with specified densities $\rho_0 = 0.4$, 0.45, 0.48, 0.52, 0.55 and 0.6, and the resultant space time diagrams were analyzed. This range of densities was chosen to expose the full range behaviors for each type of strategy. This method of strategy identification was used for all four cases. A breakdown of how many of each type of strategy was found for each of the four cases can be found in Table 6.2.

### 6.1.1 Default Strategies

Default strategies are the most simplistic rules that can be described as having a strategy. These rules simply iterate to all 0s or all 1s no matter what initial configuration is provided. These strategies therefore correctly classify approximately

| | Default | Block | Particle | Messaging | Atypical |
|---|---|---|---|---|---|
| Local Sync | 57 | 238 | 5 | – | – |
| Local Sync Previous [5] | 11 | 280 | 9 | – | – |
| Local Async | 7 | 276 | 17 | – | – |
| Small World Sync | 8 | 0 | – | 269 | 23 |
| Small World Async | 0 | 14 | – | 270 | 16 |

Table 6.2: Table of the number of rules exhibiting each strategy for each of the four cases. The line labeled Local Sync Previous indicates previous results [5] for the locally connected synchronous case. It can be seen that the number of rules seen in this research align well with previous results for the locally connected synchronous case.

50% of the initial configurations, either the $\rho < 1/2$ or $\rho > 1/2$ case, for a performance $\mathcal{P}_{149}^{10^4} \approx 0.5$. Default strategies were found in all cases except the small world asynchronous case. Space-time diagrams of default strategies all look remarkably similar, so only an example from the locally connected synchronous case is shown, and can be found in Figures 6.1 and 6.2.

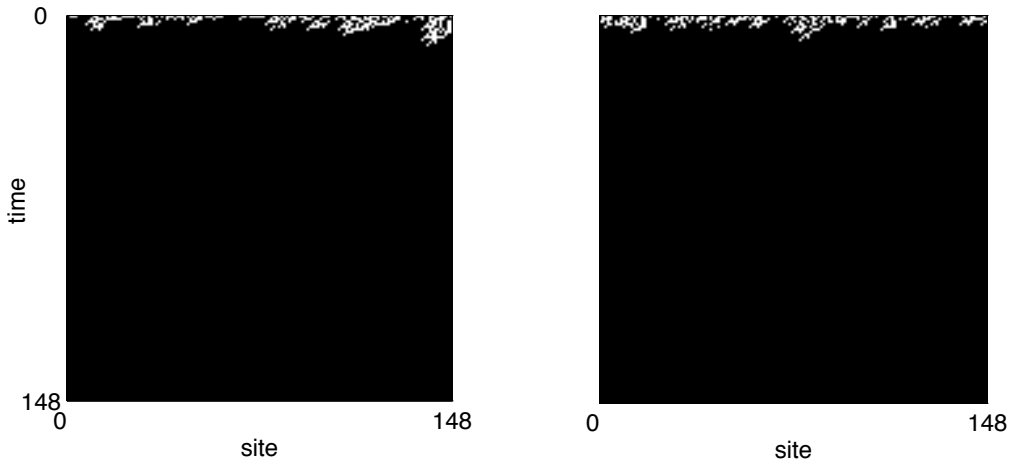


Figure 6.1: Default Strategy for a Local Synchronous CA with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.56$ (right). Only the right is correctly classified. This rule has $\mathcal{P}_{149}^{10^4} = 0.4893$.

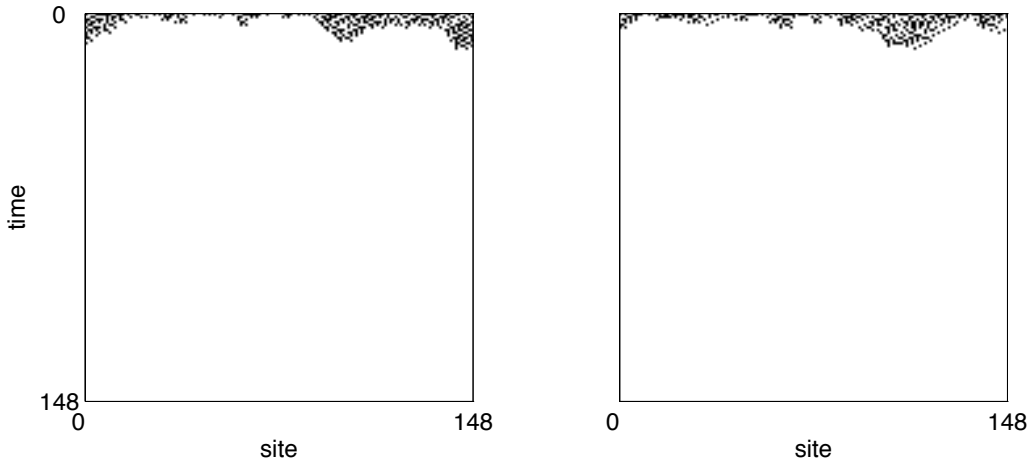


Figure 6.2: Default Strategy for a Local Synchronous CA with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.56$ (right). Only the left is correctly classified. This rule has $\mathcal{P}_{149}^{10^4} = 0.447$.

### 6.1.2 Block Expanding Strategies

Block expanding strategies attempt to improve upon the default strategy's performance by allowing the additional classification of extreme density cases in the opposite domain. Therefore a typical 1s block expanding strategy might default to all 0s except for cases with very high density, where it will iterate to all 1s. This is accomplished by looking for blocks of 1s and expanding such blocks if found. A 0s block expanding strategy does the opposite: it defaults to all 1s unless there is a block of 0s. Various block expanding strategies will have different requirements for how large a homogenous block must be before it becomes expanded. Some strategies might require a block of only 5 cells while another might require a block of 8. Mitchell et al. [5] found that the average block size for these types of strategies were typically near the neighborhood size of $2r + 1$, which appears to match with the block expanding strategies found in this research.

The rules that look for large blocks of adjacent homogenous cells are usually able to detect very high or very low density initial configurations as the chances of such blocks existing is correlated with $\rho_0$. This leads to an increase in fitness $\mathcal{F}_{149}^{100}$ and performance $\mathcal{P}_{149}^{10^4}$ as compared to the more simplistic default strategies. The results from Mitchell et al.'s work show typical block expanding strategies as having $\mathcal{F}_{149}^{100} \approx 0.9$ and $\mathcal{P}_{149}^{10^4} \approx 0.6$. It was also found that these types of strategies

do not generalize well to automata with larger N [5]. Figure 6.3 shows an example of a block expanding strategy for the local synchronous case.

Analyzing the block expanding strategies found in the asynchronous cases, a few differences from the synchronous strategies are found. In particular, due to the asynchronous update a well defined expansion rate is lost. This causes the expanding line to lose the straight edge found in the synchronous case and instead display a wavering line. This, of course, adds uncertainty to the estimation of density performed by the block expanding strategies. Thus asynchronous block expanding strategies have a lower performance as compared to the synchronous block expanding strategies, with an average $\mathcal{P}_{149}^{10^4} \approx 0.57$ as compared to the locally connected synchronous case's average of $\mathcal{P}_{149}^{10^4} \approx 0.63$. These values can be found in Table 6.3. Figures 6.4 and 6.5 show what block expanding strategies look like in both the asynchronous cases.

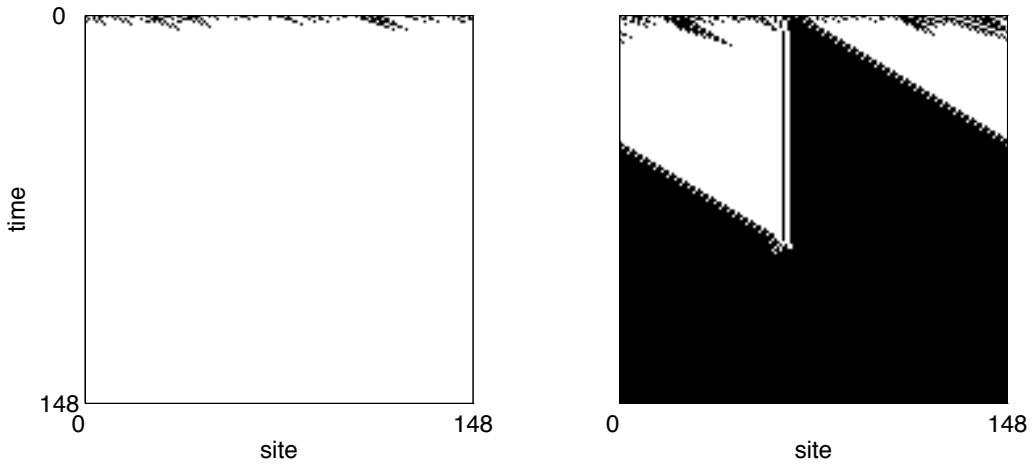


Figure 6.3: Block Expanding Strategy for a Local Synchronous CA with a $\rho_0 \approx 0.42$ (left) and $\rho_0 \approx 0.56$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.6423$.

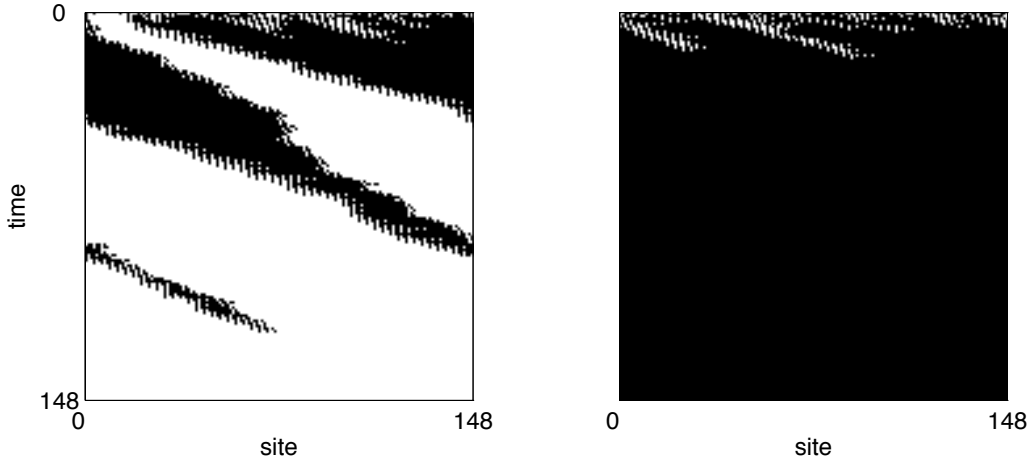


Figure 6.4: Block Expanding Strategy for a Local Asynchronous CA with $\rho_0 \approx 0.44$ (left) and $\rho_0 \approx 0.61$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.605$.

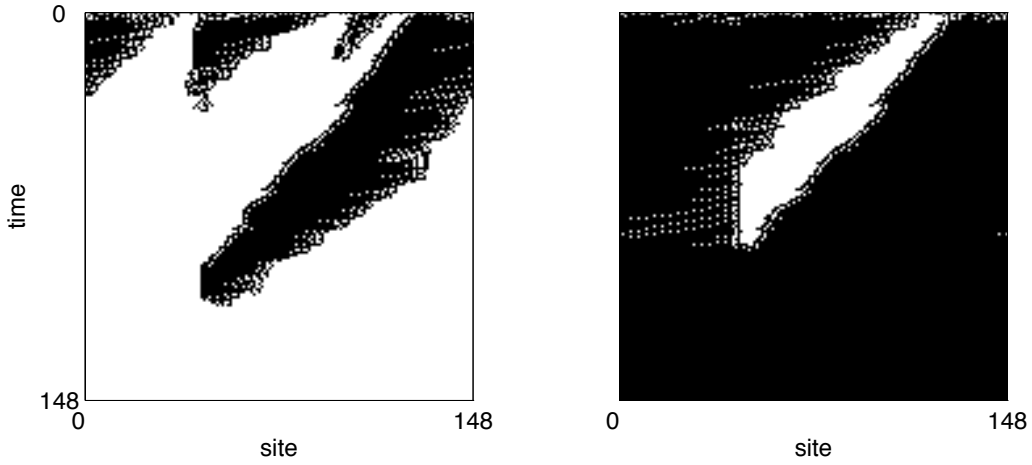


Figure 6.5: Block Expanding Strategy for a Small World Asynchronous CA with a $\rho_0 \approx 0.48$ (left) and $\rho_0 \approx 0.60$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.5662$.

### 6.1.3 Embedded Particle Strategies

So far the strategies described have been rather simplistic and have not exhibited the sort of coordinated behavior that is required for a higher level of performance in the density task. The strategies have only relied on local information and information transfer between different areas of the cellular automata has been non-existent. The strategies have also been overly reliant on the particular size of the CA and thus their performance doesn't scale as the CA grows larger.

Previous research by Mitchell, et a. [5] found a class of strategies that manages to coordinate and communicate across the cells in order to perform more complex calculations about the density of the initial configuration. These strategies take advantage of moving sections of boundaries between areas containing different patterns. These moving boundaries act as particles of information passing through the CA. When different boundaries collide, data transfer and calculation takes place in order to compute the likely density of the initial configuration. A more thorough explanation of the embedded particle strategies can be found in [5].

Embedded particle strategies were found in both the locally connected synchronous case and the locally connected asynchronous case. While the embedded particle strategies in the local asynchronous case continue to perform better than the block expanding strategies they have a maximum performance of only

$\mathcal{P}_{149}^{10^4} = 0.67$ as compared to the locally connected synchronous maximum performance of $\mathcal{P}_{149}^{10^4} = 0.74$. The asynchronous update adds an additional layer of complexity to the information transfer required by the embedded particle strategies — with the asynchronous updates, reliable transfer of data to the left and right is reduced due to the possible information loss described in Section . Asynchronous automata also lose the ability to make checkerboard patterns, which is one of the main information transfer methods used by particle strategies, as is seen in Figure 6.6. Not all fixed point patterns are lost, however, as the asynchronous automata are still able to create patterns of vertical lines.

Figure 6.6 shows example space-time diagrams of embedded particle strategies for the locally connected synchronous case, while Figure 6.7 shows space-time diagrams for the locally connected asynchronous case.

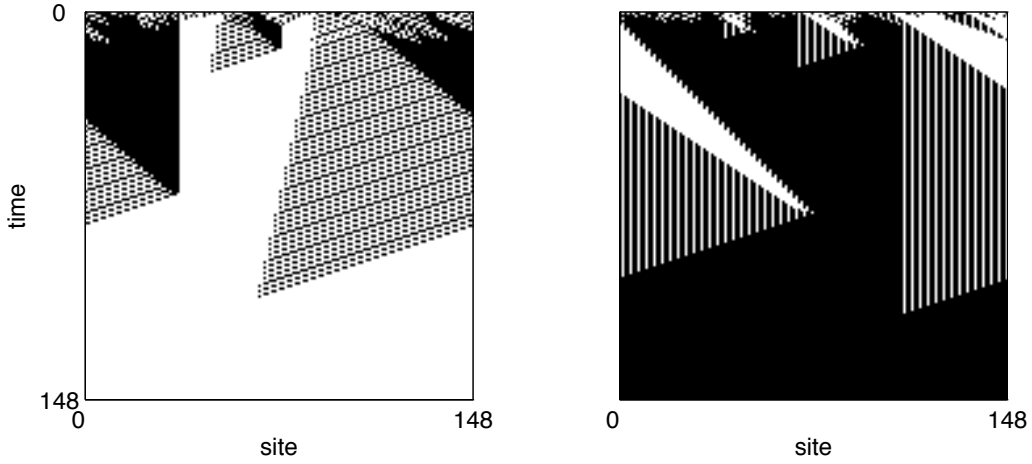


Figure 6.6: Embedded Particle Strategies for Local Synchronous CAs with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.51$ (right). Both are correctly classified. These figures are generated from two separate rules, the rule on the left has $\mathcal{P}_{149}^{10^4} = 0.702$ and the rule on the right has $\mathcal{P}_{149}^{10^4} = 0.7368$.

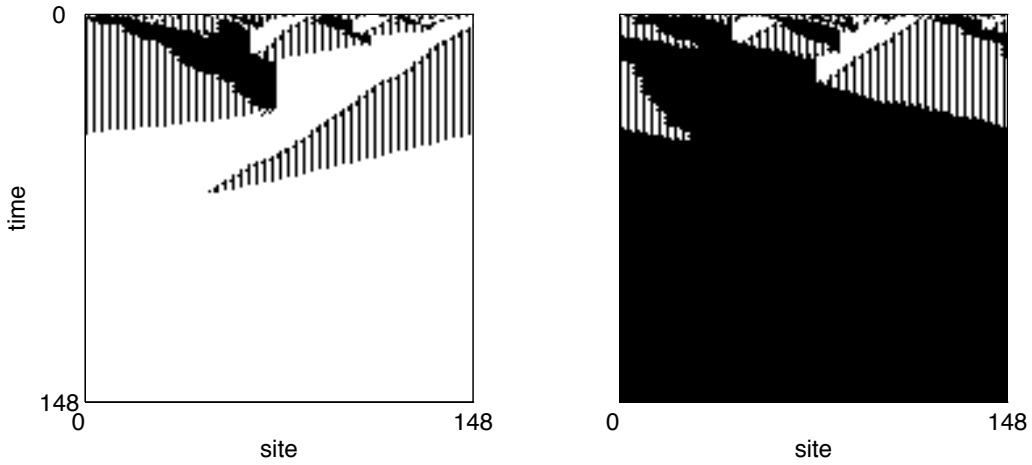


Figure 6.7: Embedded Particle Strategy for a Local Asynchronous CA with a $\rho_0 \approx 0.48$ (left) and $\rho_0 \approx 0.61$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.6727$.

### 6.1.4 Messaging Strategies

Messaging strategies are a form of strategy evolved specifically for CAs exhibiting the small world property. They exhibit diagonally moving bands of black and white. As these bands interact with the cells containing the small world links, these bands either remain as they were, expand or reduce in width, or even disappear completely. Example space-time diagrams of messaging strategies for both the small world synchronous and small world asynchronous cases are shown in Figures 6.8–6.9. A theory explaining how the messaging strategies work can be found in Section .

The failure cases for the messaging strategies include accidentally reducing the wrong bands, or having so many bands of equal width that the reduction does not happen quickly enough, such as shown in Figure 6.10.

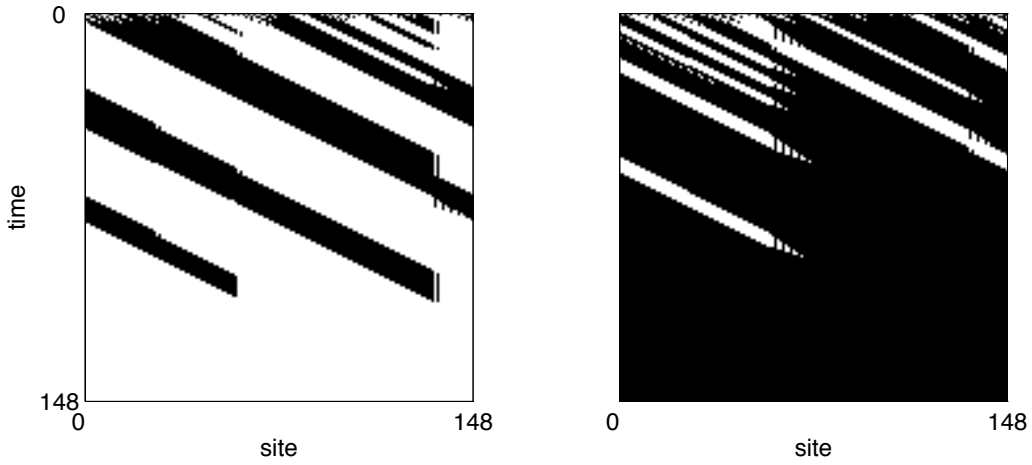


Figure 6.8: Messaging Strategy for a Small World Synchronous CA with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.58$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.7205$.

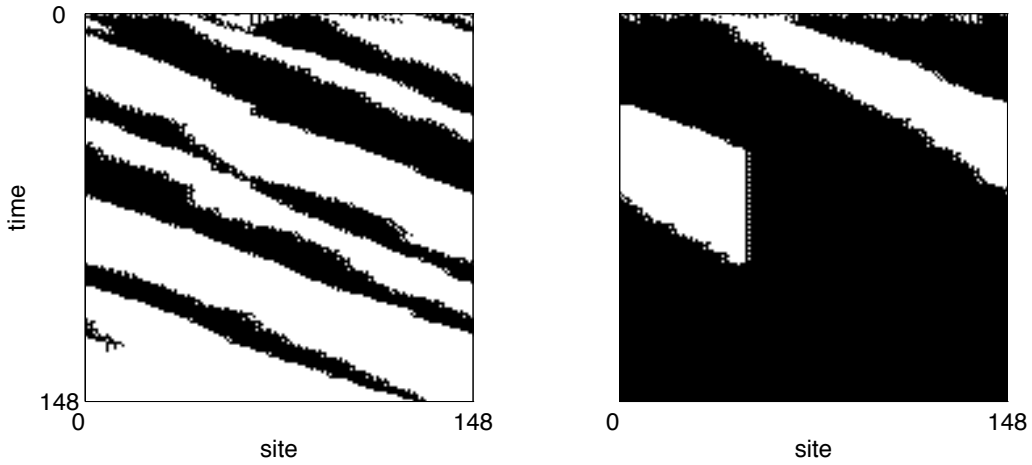


Figure 6.9: Messaging Strategy for a Small World Asynchronous CA with a $\rho_0 \approx 0.48$ (left) and $\rho_0 \approx 0.58$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.63$.

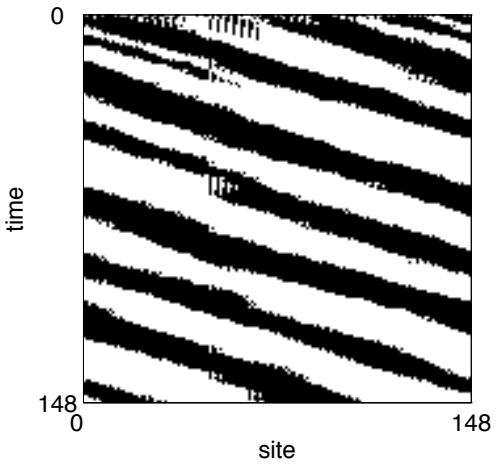


Figure 6.10: Failed Instance of a Messaging Strategy for a Small World Asynchronous CA with a $\rho_0 \approx 0.58$. This rule has $\mathcal{P}_{149}^{10^4} = 0.6369$.

### 6.1.5 Unclassified Atypical Strategies

Both the small world synchronous and small world asynchronous cases also evolved strategies showing various interesting behavior. Due to time limitations and the lack of a large sample size[1] an in depth analysis of these strategies was not done. Therefore these strategies were lumped together under the term unclassified atypical strategies. Further studies of these strategies might be a good extension of this research. Space-time diagrams of a few of these strategies are shown in Figures 6.11–6.13.

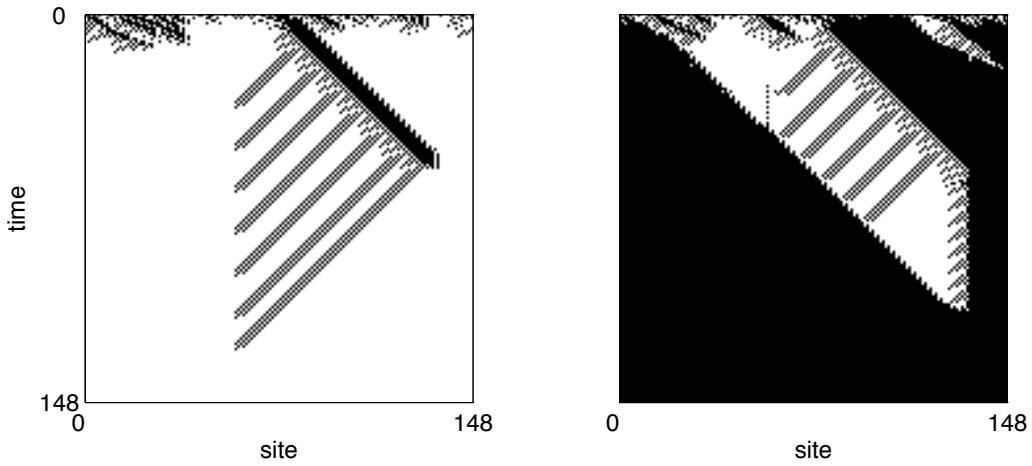


Figure 6.11: Unclassified Atypical Strategy for a Small World Synchronous CA with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.58$ (right). Both are correctly classified by a rule with $\mathcal{P}_{149}^{10^4} = 0.6933$.

[1] In some cases only one or two rules were found that exhibited a certain new behavior type.

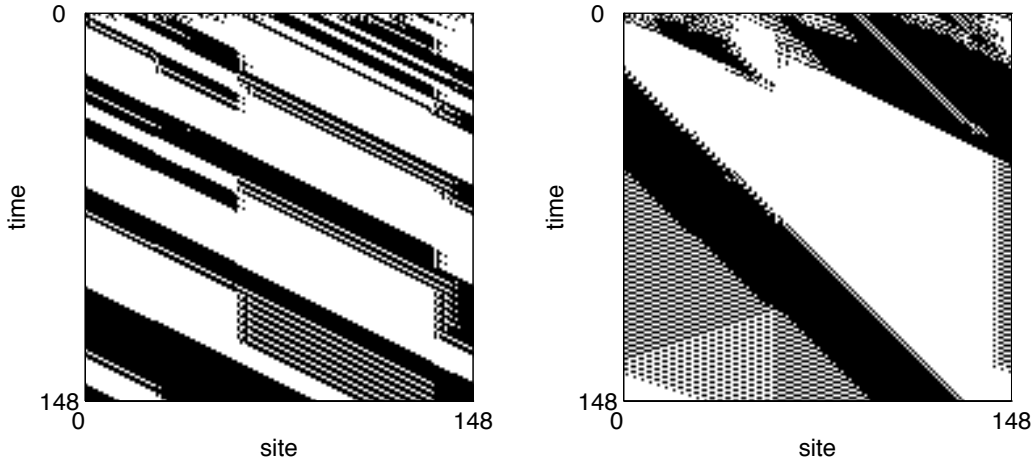


Figure 6.12: Two Unclassified Atypical Strategies for Small World Synchronous CAs with a $\rho_0 \approx 0.46$ (left) and $\rho_0 \approx 0.58$ (right). Neither is correctly classified. The rule on the left has $\mathcal{P}_{149}^{10^4} = 0.6866$ and the rule on the right has $\mathcal{P}_{149}^{10^4} = 0.6741$.

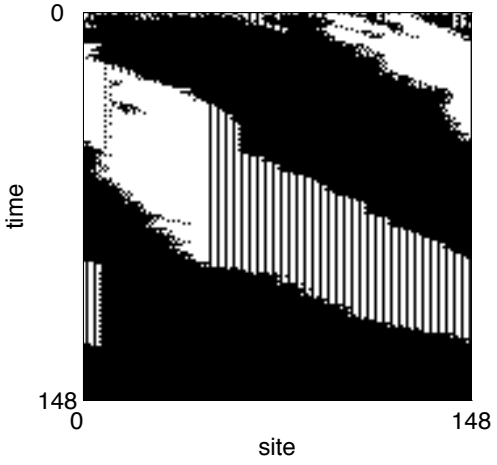


Figure 6.13: Unclassified Atypical Strategy for a Small World Asynchronous CA with a $\rho_0 \approx 0.48$. This figure is incorrectly classified and the rule has $\mathcal{P}_{149}^{10^4} = 0.5948$.

## 6.2 Performance Histograms

In order to compare the results of the various experiments, a histogram of rule performance $\mathcal{P}_{149}^{10^4}$ was made for each case. These can be seen in Figures 6.14–6.17. The histogram population for each case consists of all 100 rules in the final generation of each of the 300 runs of the genetic algorithm, for a total count of 30,000 rules per case.

The four histograms share a few commonalities. Each exhibits a small peak at $\mathcal{P}_{149}^{10^4} \approx 0$, corresponding to whichever recombinations from the previous generation were failures. There are three other, more interesting, peaks in all of the histograms. The smaller peaks at $\mathcal{P}_{149}^{10^4} \approx 0.45$ and $\mathcal{P}_{149}^{10^4} \approx 0.5$ correspond to default strategies or poor block expanding strategies or messaging strategies. The large peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ corresponds to block expanding strategies or messaging strategies, depending on the case. It is this largest peak that is of most interest. In the following paragraphs the structure of the histograms for each case will be briefly discussed.

In the locally connected synchronous histogram (Figure 6.14) there exists a small number of rules with $\mathcal{P}_{149}^{10^4} > 0.7$, which are the rules that contain embedded particle strategies. The peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ corresponds to the large number of block expanding strategies found in that case.

| | Maximum $\mathcal{P}_{149}^{10^4}$ | $\mathcal{P}_{149}^{10^4}$ Mode |
|---|---|---|
| Local Sync | 0.7368 | 0.63 |
| Local Async | 0.6727 | 0.57 |
| Small World Sync | 0.7205 | 0.64 |
| Small World Async | 0.6403 | 0.58 |

Table 6.3: Table of maximum and mode rule performance for each case. The performance mode is the performance at which the histogram peaks in number of counts near $\mathcal{P}_{149}^{10^4} \approx 0.6$ in each of the cases. This corresponds to the performance at which the largest number of high performing strategies (block expanding + particle strategies for the locally connected cases, and messaging strategies for the small world cases) can be found.

The locally connected asynchronous histogram (Figure 6.15) exhibits an overall shape and distribution of rules that is slightly different from the local, synchronous histogram. The peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ (which again corresponds to the block expanding strategies) is much larger while the two other peaks are much smaller. It can also be seen that rules with $\mathcal{P}_{149}^{10^4} > 0.7$ are non-existent. The small number of rules with highest performance correspond to the embedded particle strategies.

The small world synchronous histogram (Figure 6.16) exhibits the shape most unlike those of the other three histograms. The main peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ is shifted closer to 0.65 and is much wider and encompasses a larger percentage of total rules. The strategies encompassed by this peak are the messaging strategies. There are a few messaging strategy rules with $\mathcal{P}_{149}^{10^4} > 0.7$ but not nearly as many as compared to the local synchronous case and the maximum performance is only $\mathcal{P}_{149}^{10^4} \approx 0.72$.

The small world asynchronous histogram (Figure 6.17) seems to share common-

alities with both the small world synchronous and locally connected asynchronous histograms. Just like the locally connected asynchronous histogram, the peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ is much larger and the other two peaks are much smaller. The peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ is a wider peak, just like the small world synchronous histogram. Just like the locally connected asynchronous histogram, the small world asynchronous histogram exhibits no rules with $\mathcal{P}_{149}^{10^4} > 0.7$. The main peak in this case consists mostly of messaging strategies, with a few block expanding strategies with lower performance.

In summary, the histograms provided insight into how the different cases stacked up against each other. The degradation of the density classification performance can be seen in both asynchronous cases by the fact that the peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$ in the histograms is shifted lower as compared to the same peak in the synchronous histograms. The histograms of both small world cases show the increased ability of the genetic algorithm to find high performance rules by the larger, broader peak at $\mathcal{P}_{149}^{10^4} \approx 0.6$, as this indicates that more of the final rules were high performance rules.

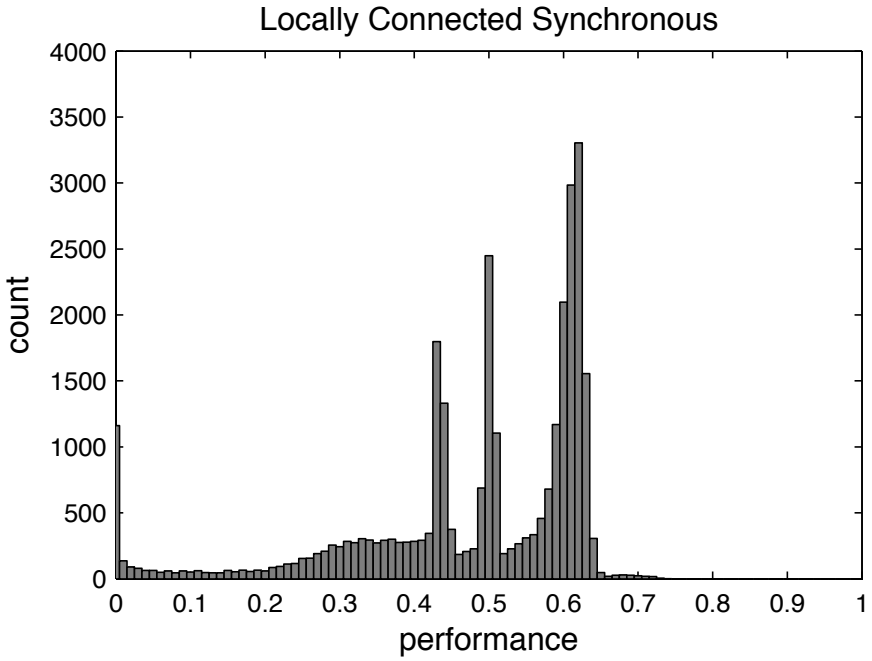


Figure 6.14: Performance Histogram for the Local Synchronous Case

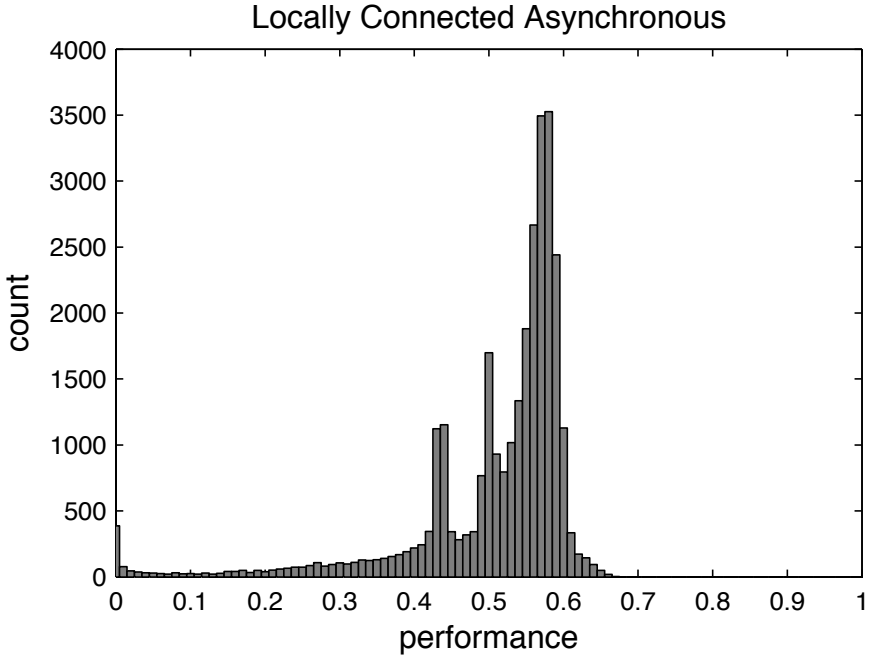


Figure 6.15: Performance Histogram for the Local Asynchronous Case

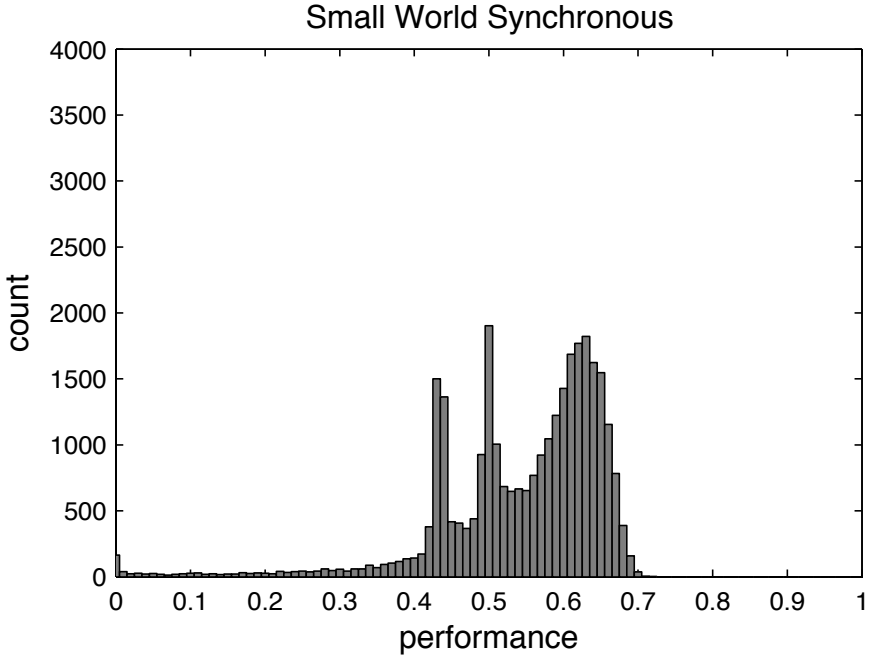


Figure 6.16: Performance Histogram for the Small World Synchronous Case

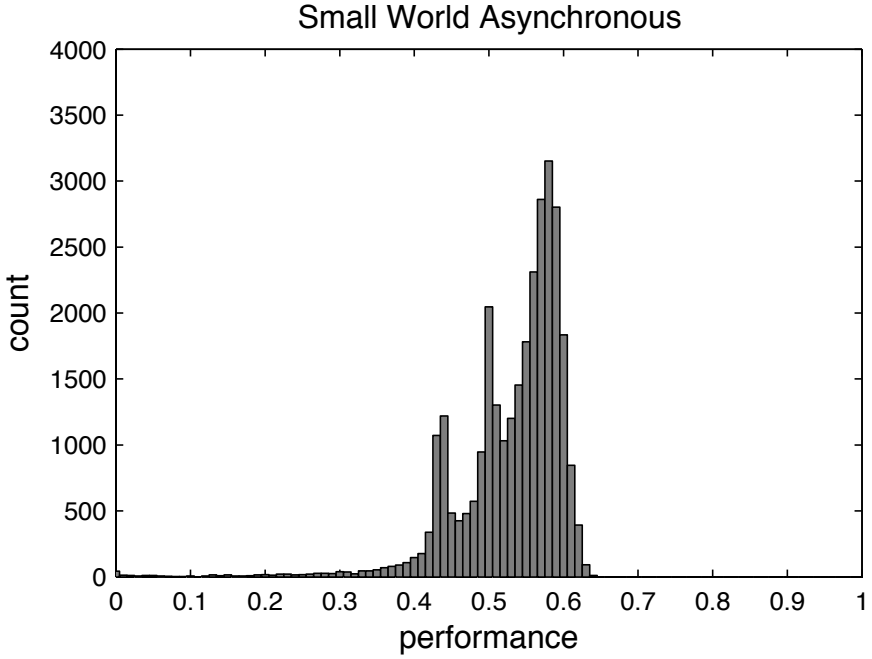


Figure 6.17: Performance Histogram for the Small World Asynchronous Case

### 6.2.1 Synchronous vs Asynchronous

One important question is whether the rules evolved in the locally connected synchronous case would also perform well in the locally connected asynchronous case. Additionally, it was wondered if the rules from the asynchronous case were actually substantially different than the rules evolved in the locally connected synchronous case. Therefore, 300 rules — the top performing rule from each run of the genetic algorithm in the locally connected synchronous case — also had their performance $\mathcal{P}_{149}^{10^4}$ on locally connected asynchronous CAs tested. Similarly, 300 rules from the locally connected asynchronous case had their performance $\mathcal{P}_{149}^{10^4}$ tested on locally connected synchronous CAs. The performance results for these two groups of rules on both synchronous and asynchronous CAs are shown in the form of histograms in Figures 6.18 and 6.19.

It can be seen in Figure 6.18 that the locally connected synchronous rules evolved in the locally connected synchronous case perform worse when run on locally connected asynchronous CAs. The locally connected synchronous rules have $\mathcal{P}_{149}^{10^4} \approx 0.5$ when run on locally connected asynchronous CAs. This is the same level of fitness acquired by the default strategies. Therefore rules constructed for the synchronous case do not work any better than chance in the asynchronous case.

Next, we consider the locally connected asynchronous rules. It can be seen from the histograms in Figure 6.19 that while the bulk of the rules had $\mathcal{P}_{149}^{10^4} > 0.6$ when run on locally connected asynchronous CAs, the performance of those same rules on locally connected synchronous CAs was much worse, with a small peak at $\mathcal{P}_{149}^{10^4} \approx 0.45$ and a much larger peak at $\mathcal{P}_{149}^{10^4} = 0$. This indicates that the rules evolved in the locally connected asynchronous case are very dissimilar to the rules evolved in the locally connected synchronous case and do not work at all for locally connected synchronous CAs. In other words, the addition of the asynchronous update requires rules tuned specifically to handle this type of update. Synchronous and asynchronous rules are not interchangeable, and thus studies on rules for synchronous CAs cannot be used to indicate anything about rules for asynchronous CAs.

An in depth analysis of the asynchronous rules and investigation of the specific methods used by the asynchronous block expanding and embedded particle strategies in order to take advantage of the asynchronous update method is outside of the scope of this thesis. The above results showing a substantial difference between mechanics used by the rules in the synchronous and asynchronous cases indicates that this would be an interesting area of study.

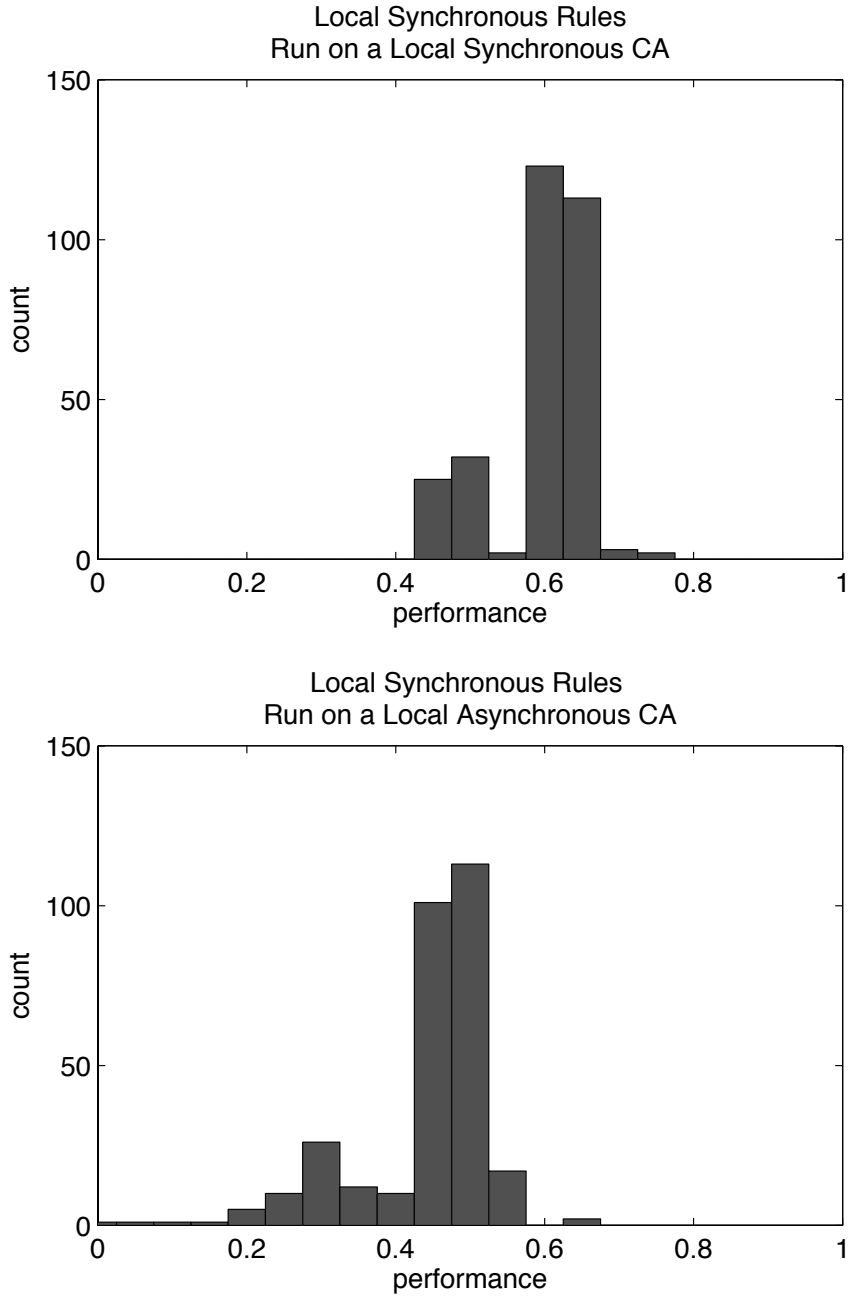


Figure 6.18: Histograms comparing performance of rules evolved for the Local Synchronous case on both Local Synchronous and Local Asynchronous CAs.

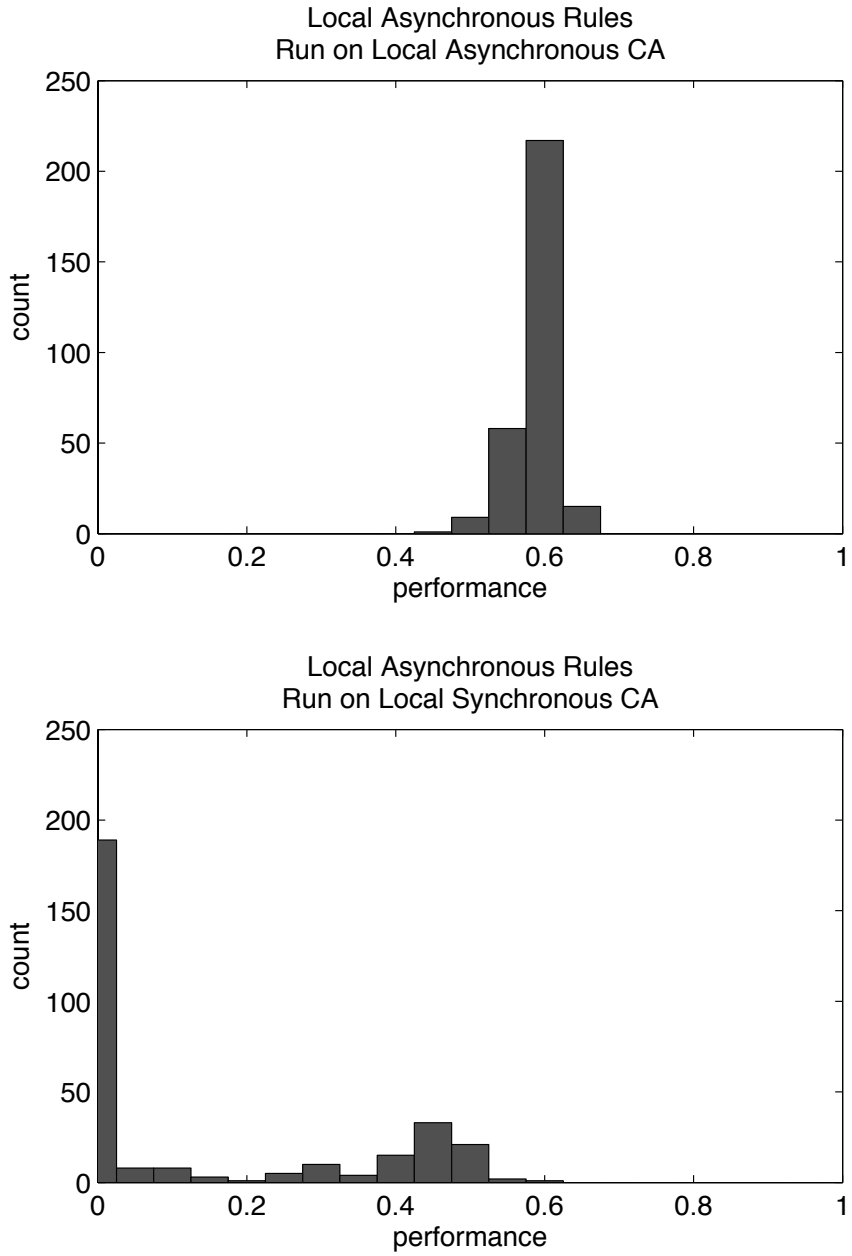


Figure 6.19: Histograms comparing performance of rules evolved for the Local Asynchronous case on both Local Asynchronous and Local Synchronous CAs.

## 6.3 Fitness vs Generation

Example plots of fitness ($\mathcal{F}_{149}^{100}$) vs generation for a typical run of the genetic algorithm for the four different cases are shown in Figures 6.20–6.23. The plot of fitness vs generation for a single run of the genetic algorithm plots the highest fitness of any rule found in that generation against the generation number. Each of the following plots shows an overlay of eight different fitness vs generation plots for different runs of the genetic algorithm for a particular case. This is to more accurately show what happens during an average run for that case. Recall that the fitness calculation $\mathcal{F}_{149}^{100}$ uses a density-uniform distribution for selecting initial configurations, compared with the unbiased distribution used to calculate the performance $\mathcal{P}_{149}^{10^4}$. This explains why the plots of fitness vs generation show the fitness $\mathcal{F}_{149}^{100}$ nearing 1.0 while the performance would be below $\mathcal{P}_{149}^{10^4} \approx 0.75$. For more explanation see Section .

The plots for the locally connected synchronous case (Figure 6.20) match well with previous results [5]. For most runs a high fitness rule is found within the first 20 generations, with the occasional run failing to find a high fitness rule at all.

The plot for the locally connected asynchronous case (Figure 6.21) shows that the number of generations needed to develop the block expanding strategies is on average increased as compared to the local synchronous case. This is another

indication that asynchronous updates provide an additional obstacle for performing global tasks such as the density classification task.

The plot for the small world synchronous case (Figure 6.22) shows a much faster increase in fitness both in numbers of generations until an increase in fitness is first exhibited and in the speed at which fitness is increased as compared to the locally connected synchronous case. The best rules found for the small world synchronous case also exhibit a higher fitness than the locally connected cases. This is more evidence that the small world property allows for faster communication within the cellular automata, which increases its ability to perform the density classification task.

Finally, the plot for the small world asynchronous case (Figure 6.23) indicates that the presence of asynchronous updates again causes the genetic algorithm some difficulty in finding high fitness rules. The plots also show some of the same fast increases that were exhibited by the small world synchronous case.

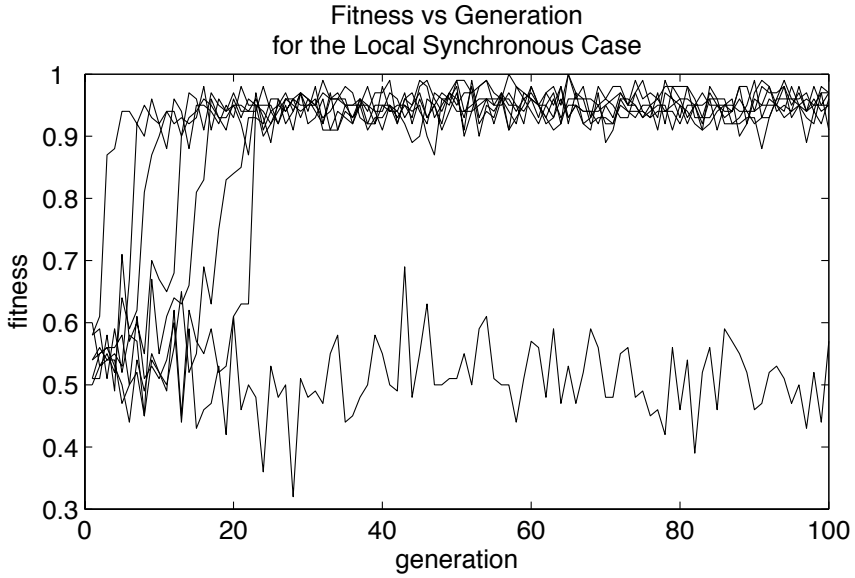


Figure 6.20: Plot of Fitness ($\mathcal{F}_{149}^{100}$) vs Generation for eight runs of the genetic algorithm for the Local Synchronous Case

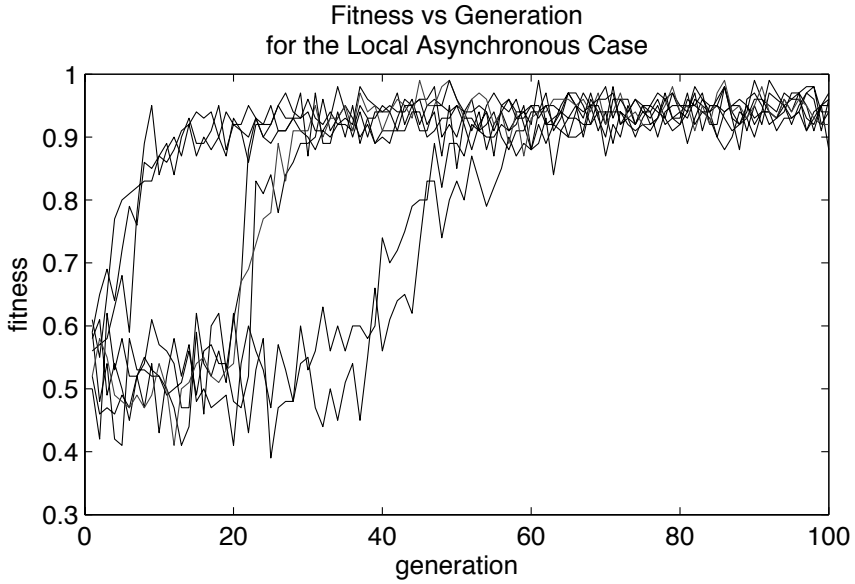


Figure 6.21: Plot of Fitness ($\mathcal{F}_{149}^{100}$) vs Generation for eight runs of the genetic algorithm for the Local Asynchronous Case

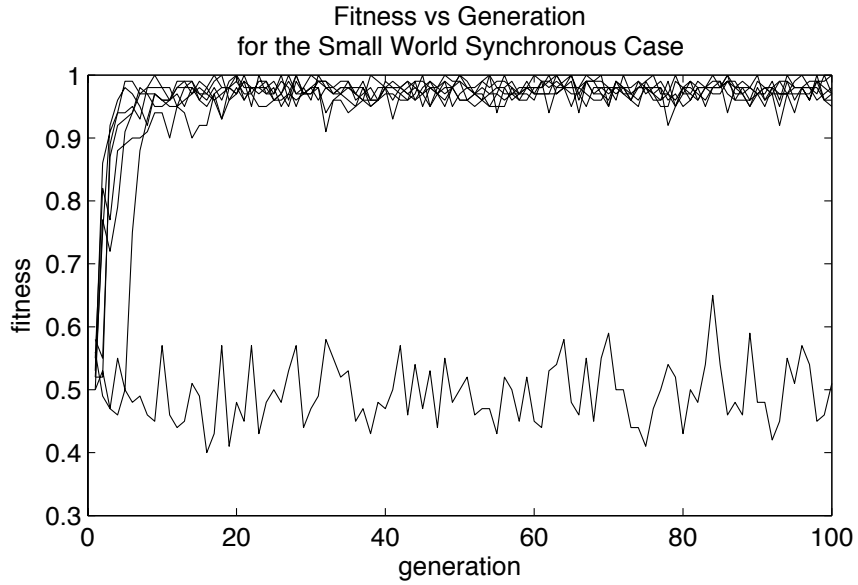


Figure 6.22: Plot of Fitness ($\mathcal{F}_{149}^{100}$) vs Generation for eight runs of the genetic algorithm for the Small World Synchronous Case

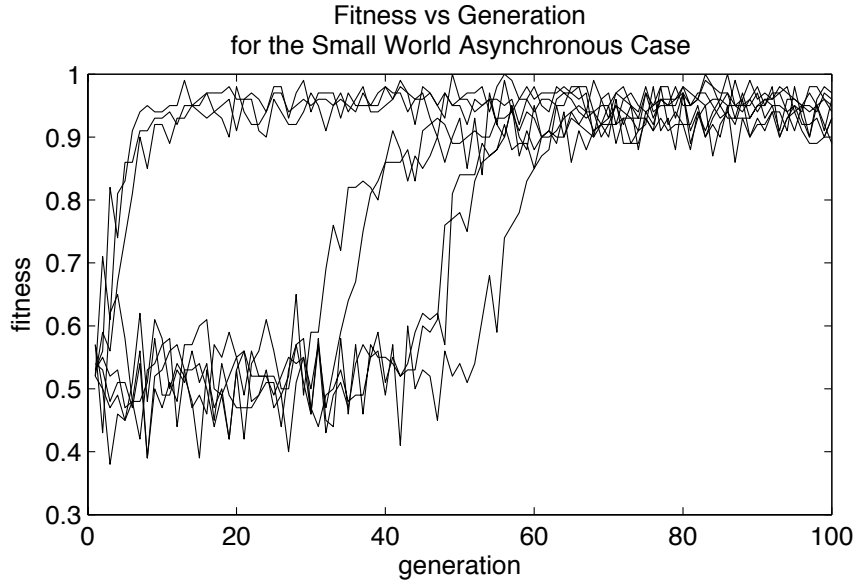


Figure 6.23: Plot of Fitness ($\mathcal{F}_{149}^{100}$) vs Generation for eight runs of the genetic algorithm for the Small World Asynchronous Case

| | Average # of Generations | Standard Deviation |
|---|---|---|
| Local Synchronous | 16.99 | 19.47 |
| Local Asynchronous | 30.65 | 21.60 |
| Small World Synchronous | 7.03 | 10.46 |
| Small World Asynchronous | 26.98 | 22.29 |

Table 6.4: Table of average number of generations before reaching $\mathcal{F}_{149}^{100} > 0.8$ and the corresponding standard deviation for each of the four cases.

The fitness vs generation statistics for all 300 runs of the genetic algorithm for each of the four cases were analyzed in order to calculate the average generation and standard deviation at which the genetic algorithm discovers a high fitness rule. For these calculations, the generation at which the fitness reaches $\mathcal{F}_{149}^{100} = 0.8$ was used as a definition of the generation at which the genetic algorithm discovered a high fitness strategy such as a block expanding strategy, embedded particle strategy or messaging strategy. This calculation will provide more insight into how easy or difficult it was for the genetic algorithm to locate the high fitness rules for each of the four cases.

In comparing the results of Table 6.4, it can be seen that the addition of asynchronous updates increases the average number of generations before the genetic algorithm reaches $\mathcal{F}_{149}^{100} = 0.8$. The average generation for the local asynchronous case was almost double that of the local synchronous case, and the small world asynchronous case was almost four times greater than the small world synchronous case. This is further indication that the asynchronous update step hinders the abil-

ity of the CA to perform the density classification task. Comparing the average number of generations for the small world synchronous case as compared to the local synchronous case shows that the addition of the small world property to the CA substantially increases the genetic algorithm's ability to locate high fitness rules, as the average number of generations is less than half that of the locally connected case. This is more evidence showing that the addition of the small world property increases the CAs ability to transfer information and thus perform the density classification task.

Average fitness at each generation was calculated by taking all 300 fitness values for a particular generation (one fitness value per run of the genetic algorithm) and averaging them together. These average fitness values are then plotted against generation for all four cases in Figure 6.24. In this plot both of the asynchronous cases exhibit a much shallower slope, which is indicative of the higher standard deviation in the number of generations taken by the genetic algorithm in order to locate high fitness rules. Both synchronous cases exhibit an increase in fitness that more closely matches the individual fitness vs generation plots shown in Section . At first this might seem a little paradoxical, as the locally connected synchronous case also shows a high standard deviation and thus a shallow slope such as seen for the asynchronous cases would be expected. This is explained by the much higher number of runs that fail to find a high fitness rule at all, which increases the

standard deviation of the locally connected synchronous case. The runs that do find high fitness rules continue to do so within a tighter generation range, causing the well-defined increase in the plot of average fitness.

The locally connected synchronous case shows the lowest maximum average fitness of only $\mathcal{F}_{149}^{100} \approx 0.85$, while the three other cases have an maximum average fitness of $\mathcal{F}_{149}^{100} \approx 0.95$. This is again explained by the fact that the locally connected synchronous case has a much greater number of runs finding only default rules, which brings down the maximum average fitness. Both the asynchronous cases and the small world synchronous case have much lower numbers of default rules, and thus a higher maximum average fitness.

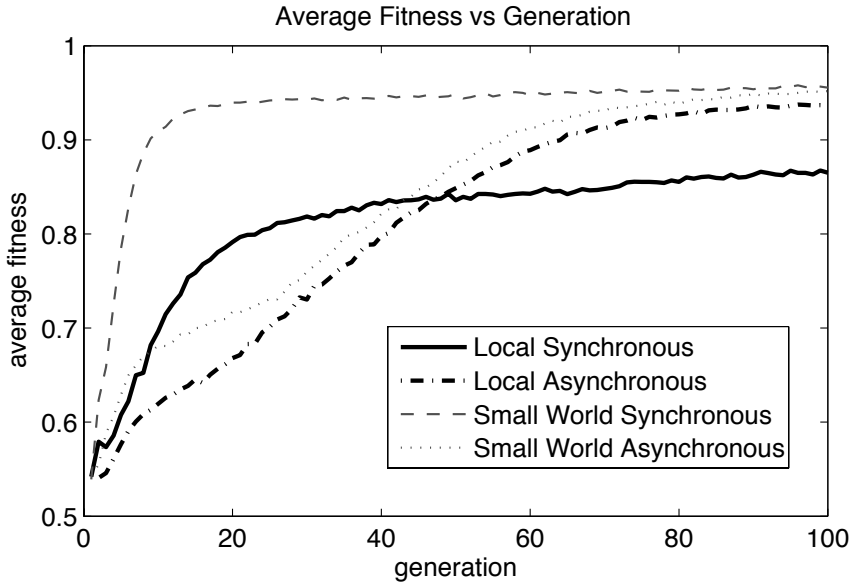


Figure 6.24: Plot of Fitness ($\mathcal{F}_{149}^{100}$) vs Generation averaged over all 300 runs of the genetic algorithm for each case.

## 6.4 Analysis of the Messaging Strategies

The messaging strategies discovered in the analysis of the small world cases exhibits interesting behavior allowing the transmission of information through the CA by taking advantage of the small world links. This section provides a theory as to how these strategies accomplish this information transfer. This analysis only describes some of the more obvious behavior as seen from the space-time diagrams of rules exhibiting the messaging strategy. A more thorough analysis can and should take place in order to more thoroughly understand how these rules work, which would possibly help with the creation of other, higher performing rules. Space-time diagrams of these strategies can be found in Subsection .

In order for a rule strategy to be able to take advantage of the small world links between random cells, the strategy itself must not assume anything about where these links are located. The only thing the strategy must rely on is the fact that these links do in fact exist. In order for these links to be taken advantage of, the CA must transmit the information on entire state of the initial configuration to each of the links. This is accomplished by first creating a simplified version of the initial configuration. A local section containing mostly 1s (black cells) becomes a block of black. A section containing mostly 0s (white cells) becomes a block of white. Next, this information must be transmitted to the link locations. This is

accomplished by simply moving the entire state to the right or left. In this way bands of black and white cells sweep down the space-time diagram in a diagonal manner.

When a band hits a link location, the cell containing the link is now connected to two different bands. The cell can thus judge the relative frequency of black or white bands, and adjust one of the bands accordingly. It can thus be seen that the bands reduce in width or disappear altogether at particular locations — the link locations. It is in this fashion that an initial collection of stripes is reduced to a fixed point of 1s or 0s.

To more fully explain how this works, let us think about what must happen at a band edge in order to keep a band moving to the right. Let's imagine that a black band is moving to the right. The leading edge of the band is about to enter a cell. At this point, the cell itself and all neighbor cells to the right are white, while all the neighbor cells to the left are black. In the locally connected case the center cell wants to adjust itself to a black state, meaning that in the next time step the black band will have moved one cell to the right. Such a configuration is shown in Figure 6.25. Now let's imagine what might happen in the case that this cell is not locally connected, but contains a small world link to some other area of the CA. Let's imagine that this link is attached to one of the cells on the left, replacing what would have been part of the black band. Now the center cell

has a single data point sample of the density of another section of the automata. If this sample is black, then nothing has changed and the band moves as before, as seen in Figure 6.26. If this sample is white, however, then the center cell has the opportunity to cut off one cell of the black band — instead of switching to black, it may remain white, as seen in Figure 6.27. This would reduce the width of the black band by one cell. It is in this way that the cells containing small world links can sample the density of a different area of the automata and apply this knowledge to the density of its area of the automata. This is but a simple example of the how such a strategy might work. There are undoubtedly other aspects of messaging strategies that have not been discussed here and would be open for further investigation.

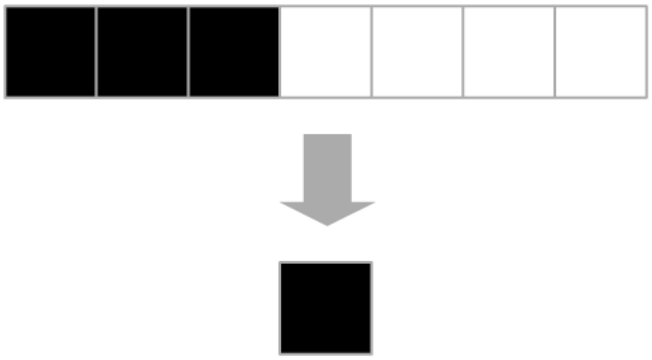

Figure 6.25: Block of black moving right in a locally connected neighborhood.

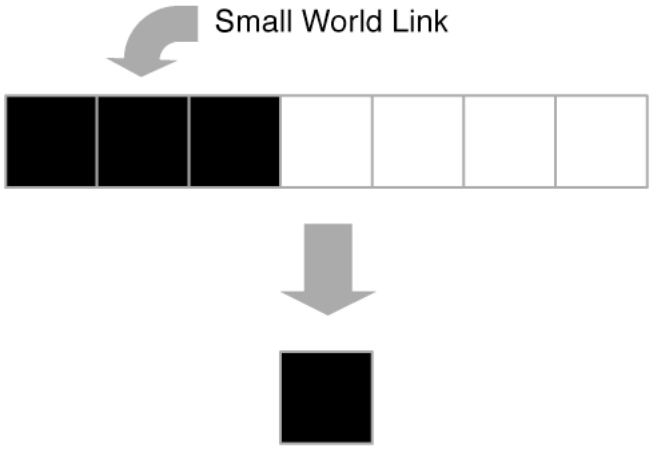


Figure 6.26: Block of black moving right in a neighborhood with a small world link. The link points to a black cell, so the behavior is not changed.

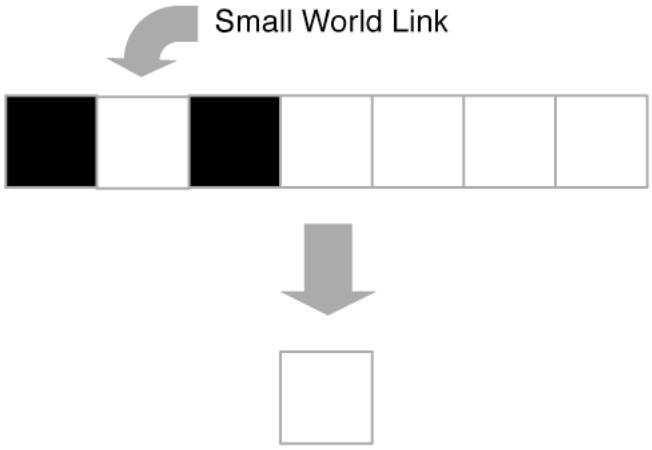


Figure 6.27: Block of black moving right in a neighborhood with a small world link. The link points to a white cell, so the movement of the black block is halted.

# 7 Related Work

Genetic algorithms have been shown to successfully generate CA rules that perform various computations such as the density classification task and the synchronization task [22, 5]. A few related areas of study include coevolution [25, 15], genetic programming [2], and extending the connectivity of the CA to exhibit the small world property [26]. Coevolution was used to successfully increase the percentage of runs that developed embedded particle strategies and discovered the most successful rule for small ($N \approx 149$) CAs. Andre, Bennett and Koza [2] have discovered a very successful density classification rule using genetic programming as opposed to a genetic algorithm. This rule shows improvement over rules discovered by genetic algorithms and the most successful human-written rules. Extending the connectivity of a CA to exhibit the small world property has proven to increase the ability of rules to perform well on the density classification task [2]. Random Boolean networks have also been studied with the aim of solving global tasks [19, 18].

Biologically inspired models such as CAs have been successfully used to simu-

late sensor networks. Jones et al. [14] showed that a simple 2-D CA could be used to determine the exact average value of a field of motes within a short number of timesteps. Wokoma, Sacks and Marshall [30] have also expressed the capabilities of CAs for simulating sensor networks.

Genetic algorithms and other biologically inspired techniques have also been used to optimize the structure of a sensor network in order to minimize power consumption while maximizing the detection accuracy of the network [3, 7, 13]. While this is not directly relevant to the research covered here, it shows how genetic algorithms have been successfully applied to other problem areas in the sensor network field and supports the importance and feasibility of sensor networks in the future of computing.

The synchronization of asynchronous sensor networks has also been an area of study, though the use of genetic algorithms in this area appears limited. Current research has focused on techniques such as post-facto synchronization [8] and other methods described in [9]. In a similar vein, studies have also proven that a synchronous CA can be simulated with a sufficiently large asynchronous CA [1].

# 8 Future Work

There are a variety of extensions possible for this research. One is simply to repeat the experiments given the synchronization task (Section ) instead of the density classification task.

Another area to study is a more thorough concentration on the small world cellular automata, including more experiments investigating how different numbers of reconnections affect the CA's performance. The research in this thesis only concentrated on a single reconnection value.

Another idea that is aimed towards increasing the ability of the genetic algorithm to find embedded particle strategies or other strategies implementing high-level communication is to slowly adjust the initial configurations chosen in each generation. Currently the initial configurations are chosen from a density-uniform distribution. This spreads the densities of the initial configurations from 0 to 1, rather than concentrating on the most difficult 0.5 density case. However, when comparing the performance of block expanding strategies to that of embedded

particle strategies on a population of such initial configurations, not much difference is found. Both types of strategies score around $\mathcal{F}_{149}^{100} = 0.9$ on a collection of density-uniform initial configurations. This makes it more difficult for the genetic algorithm to distinguish which type of strategy is better at categorizing a collection of unbiased initial configurations, like those used to calculate the performance $\mathcal{P}_N^I$ of a rule. Therefore, it would be interesting to study a genetic algorithm that slowly switches from using a density-uniform distribution of initial configurations at the start to an unbiased distribution of initial configurations by the end. The thought is that such a genetic algorithm would be able to find high-performance rules with much higher probability. This is somewhat similar to a coevolution approach.

Evaluation of the newly discovered messaging strategies can and should be further studied. A more comprehensive analysis on the methods used within the messaging strategies could be employed to create small world networks more carefully fashioned in order to fully take advantage of this strategy. In addition, further study of some of the other interesting unclassified strategies found in the small world runs could be interesting.

# 9 Conclusions

This thesis explores the relationships between asynchronous and synchronous updates as well as locally connected and small world networks on the ability of rules to perform the density classification task for $k = 2$, $r = 3$ cellular automata. The performance of locally connected synchronous cellular automata was compared to previous research [5, 20, 22] in order to validate the testing procedure. Next, the performance of locally connected asynchronous, small world synchronous and small world asynchronous cellular automata were compared.

It was discovered that cellular automata exhibiting asynchronous updates continued to develop embedded particle strategies in the locally connected case and messaging strategies in the small world case. The asynchronous updates did degrade cellular automaton performance on the density classification task. Showing the evolution of strategies incorporating advanced information transfer is an important step towards applying the knowledge gained from these strategies to the field of sensor networks. Requiring sensor networks to update synchronously will

most likely remain a burdensome requirement for the near future, thus methods exhibiting asynchronous updates are desirable. It was also discovered that the rules found in the synchronous case and rules found in the asynchronous case were not interchangeable, which further promotes the importance of studying asynchronous cellular automata.

In studying the addition of a small world network topology to the cellular automata, an entirely new type of rule strategy, dubbed the messaging strategy, was discovered. These strategies take advantage of the small world links within a cellular automaton in order to transmit information through the automaton more quickly. A theory on how the rules take advantage of the small world links was proposed and analyzed. Evidence indicating the small world topology increased a cellular automaton's ability to transfer information was discovered. While the type of small world topology studied in this thesis was constructed to be broad, more research comparing the results of more specific types of small world topologies would be beneficial in discovering more about the performance bounds of the messaging strategy.

It is hoped that our results will provide helpful in the areas of both cellular automata and sensor networks. Comparisons between the performances of cellular automata exhibiting both asynchronous updates and the small world property have been discussed in the specific instance of performing the density classification task,

which provides an idea of the capabilities and limitations of such cellular automata.

## References


[1] S. Adachi. Computation by asynchronously updating cellular automata. *Journal of Statistical Physics*, pages 261–289, 2004.

[2] D. Andre, F. Bennett, and J. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. *Proceedings of the First Annual Conference on Genetic Programming*, pages 3–11, 1996.

[3] A. Buczak, Y. Jin, H. Darabi, and M. Jafari. Genetic algorithm based sensor network optimization for target tracking. *Intelligent Engineering Systems through Artificial Neural Networks*, 9:349–354, 1999.

[4] C. Chong and S. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91:1247–1256, 2003.

[5] J. Crutchfield, M. Mitchell, and R. Das. The evolutionary design of collective computation in cellular automata. In J. P. Crutchfield and P. K. Schuster, editors, *Evolutionary Dynamics – Exploring the Interplay of Selection, Neutrality, Accident, and Function*, pages 361–411. New York: Oxford University Press, 2003.

[6] J. Dreicer, A. Jorgensen, and E. Dors. Distributed sensor network with collective computation for situational awareness. *AIP Conference Proceedings*, 632:235–243, 2002.

[7] F. Dressler, B. Krüger, G. Fuchs, and R. German. Self-organization in sensor networks using bio-inspired mechanisms. *Proceedings of the 18th ACM/GI/ITG International Conference on Architecture of Computing Systms - System Aspects in Organic and Pervasive Computing (ARCS'05): Workshop Self-Organization and Emergence*, 2005.

[8] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. *Proceedings of the 2001 International Parallel and Distributed Processing Sym-*

*posium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, 2001.

[9] J. Elson and K. Römer. Wireless sensor networks: A new regime for time synchronization. *ACM SIGCOMM Computer Communication Review*, 33:149–154, 2003.

[10] H. Fuks. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55:2081R–2084R, 1997.

[11] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. *Technical Report CSD-TR 02-0013, UCLA*, 2002.

[12] Y. Ishida. The immune system as a self-identification process: a survey and a proposal. *International Workshop on Immunity-Based Systems*, 1996.

[13] S. Jin, M. Zhou, and A. Wu. Sensor network optimization using a genetic algorithm. *Proceedings of the 7th World Multiconference on Systemics, Cybernetics, and Informatics*, 2003.

[14] K. H. Jones, K. N. Lodding, L. Wilson, and C. Xin. Biology inpired approach for communal behavior in sensor networks. *Proceedings of the 39th Hawaii International Conference on System Science*, 2006.

[15] H. Juillé and J. B. Pollack. Coevolutionary learning: a case study. *In Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[16] M. Land and R. K. Belew. No Perfect Two-State Cellular Automata for Density Classification Exists. *Physical Review Letters*, 74:5148–5150, 1995.

[17] C. Langton. *Artificial Life, Santa Fe Institute Studies in the Sciences of Complexity Proceedings*, volume VI. Redwood City, CA: Addison-Wesley, 1989.

[18] B. Mesot and C. Teuscher. Critical values in asynchronous random boolean networks. *7th European Conference on Artificial Life*, 2003.

[19] B. Mesot and C. Teuscher. Deducing local rules for solving global tasks with random boolean networks. *Physica D*, pages 88–106, 2005.

[20] M. Mitchell. Coevolutionary learning with spatially distributed populations. In G. Y. Yen and D. B. Fogel, editors, *Computational Intelligence: Principles and Practice*. New York: IEEE Computational Intelligence Society, 2006.

[21] M. Mitchell. Complex systems: Network thinking. *Artificial Intelligence*, pages 1194–1212, 2006.

[22] M. Mitchell, J. 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)*, 1996.

[23] M. Mitchell, J. Crutchfield, and R. Das. Evolving cellular automata to perform computations. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford: Oxford University Press, 1998.

[24] M. Newman. Models of the small world. *J. Stat. Phys.*, 101:819–841, 2000.

[25] J. Paredis. Coevolving cellular automata: Be aware of the red queen! *Proceedings of the 7th Int. Conference on Genetic Algorithms*, 1997.

[26] C. Teuscher. On irregular interconnect fabrics for self-assembled nanoscale electronics. *2nd IEEE Int. Workshop on Default and Fault Tolerant Nanoscale Architecture*, 2006.

[27] C. Teuscher. Small-world power-law interconnects for nanoscale computing architectures. *Proceedings of the 6th IEEE Conference on Nanotechnology, IEEE Nano 2006*, 2006.

[28] J. von Neumann. *The Theory of Self-reproducing Automata*. Univ. of Illinois Press, Urbana, IL, 1966.

[29] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[30] I. Wokoma, L. Sacks, and I. Marshall. Biologically inspired models for sensor network design. *Proceedings of the London Communications Symposium*, 2002.

[31] S. Wolfram. *A New Kind of Science*. Wolfram Media, Inc., 2002.