DOI: http://dx.doi.org/10.7551/978-0-262-33027-5-ch034

Practical Fault Tolerant 2D Cellular Automata

Steven Janke and Matthew Whitehead

Colorado College
Mathematics and Computer Science
14 E. Cache La Poudre St.
Colorado Springs, CO 80903

{sjanke, matthew.whitehead}@coloradocollege.edu

Abstract

Cellular automata often suffer from a level of brittleness that makes them susceptible to even the smallest unexpected environmental changes. We propose a method of converting CAs into more robust structures called meta-CAs that utilize cell redundancy along with added rules to correct errors and reproduce the functionality of the original CA. We show that the use of these meta-CAs can greatly increase the probability of CAs being intact when executing in an environment where cells fail on each step with a small probability.

Introduction

Cellular automata (CA) are the basis for many examples of computational models in artificial life research. The Game of Life is a traditional CA example where some configurations are mobile, some repeatedly reproduce new subconfigurations, and some simulate logical computation. CAs typically operate by using the neighborhood of a cell to determine the next state of the cell. This means that the CA often suffers from *brittleness* where the loss of a single cell's state completely disrupts the resulting configuration.

For example, Langton's loop (Langton (1984); see Fig.10) was one of the original self-reproducing CAs that consisted of a starting configuration of 86 cells in a particular pattern. During the simulation of the CA, the loop would reproduce yielding an exact replica of itself. If even a single cell of the original pattern is in the wrong initial state, then the entire loop fails to reproduce properly.

This brittleness makes CAs less than optimal for artificial life research since some level of robust behavior is indicative of life-like behavior. In this work, we propose a general procedure for converting a regular CA into a fault-tolerant CA. The procedure involves replacing each original cell with a *metacell* which is a square array of cells with additional states and rules. The new CA is still a normal CA where each cell executes a common set of instructions. The result is a cellular system that mirrors biological cells and possibly could evolve from less robust systems. We focus on two-dimensional CAs and set-up a reasonable interpretation of faulty cells in order to test our procedure on a set of representatives.

Related Work

One of the first to explore fault tolerance in digital systems was von Neumann (1956) who utilized levels of redundancy to build reliable digital circuits from unreliable components. Later in 1975, two sets of researchers, (Harao and Noguchi (1975)) and (Nishio and Kobuchi (1975)), set a more formal stage for studying reliability in cellular automata. In these papers, failure of cells was spatially restricted to make the algebraic analysis feasible.

Theoretical work continued with Peter Gacs' extensive papers (1986, 1989) which include proofs of reliability in certain general settings. More recent work by McCann and Pippenger (2008, 2013) used a majority vote among neighbors to increase the reliability of CAs again in a theoretical setting.

Work in designing cellular automata led to CAs capable of self-reproduction (Byl (1989), Perrier et al. (1996)) and these example CAs are sufficiently complex to serve as test cases for fault tolerant research. One such design (Langton (1984)), referred to as Langton's loop, helped motivate this current research and serves as a key example in our original testing.

Work using self-reproducing loops took a more biological tact when Hiroki Sayama described loops that were able to dissolve when intruded upon (Sayama (1998)). His work continued with loops that evolved over time based on environmental constraints (typically lack of space) (Sayama (1999)). Sayama later described CAs that were able to employ defensive mechanisms to remain intact when facing intrusions (Sayama (2004)). Oros et al. more recently describe a system that uses loops that are able to reproduce and pass along copies of the parents' genetic material (Oros and Nehaniv (2007)).

Cellular Automata and Reliability

Design

Two-dimensional cellular automata which are the focus of this research are rectangular arrays of identical finite automata. An individual finite automaton is called a *cell* and the eight cells surrounding a given cell in the array comprise

Steven Janke, Matthew Whitehead (2015) Practical Fault Tolerant 2D Cellular Automata. Proceedings of the European Conference on Artificial Life 2015, pp. 158-165

the *neighborhood*. Traditionally, there are two interpretations of the neighborhood. In the von Neumann design, the neighborhood is just the four cells bordering to the north, east, south, and west; the other four are ignored. The Moore design includes all eight cells in the neighborhood.

The cells operate on discrete time steps and in one step, a cell uses its current state plus the state of the neighbors to determine its next state. The transition is described by a function that sends the neighborhood configuration along with the current state of the central cell to a new state for the central cell. This transition function can be presented as a set of rules. For a Moore neighborhood design, each rule is a 10-tuple of states, $(S_{cur}, S_N, S_{NE}, S_E, S_{SE}, S_S, S_{SW}, S_W, S_{NW}, S_{new})$. Here, the given cell is currently in state S_{cur} and the eight neighbors are in the states listed. On the next time step, the given cell transitions to state S_{new}. This means that for the Game of Life CA, since there are two cell states (usually designated "alive" and "dead"), there are 2^8 possible Moore neighborhoods. The given cell has two possible states S_{curr}, so there are $2^8 \times 2 = 2^9$ rules in the transition function. In practical implementations, many of these rules are inactive in the sense that they do not change the state of the central cell; utilizing this observation can speed up the actual simulation. Also, it is generally agreed that state 0 is designated as a quiescent state and therefore the tuple (0,0,0,0,0,0,0,0,0,0) is a rule in most CAs.

In general, with S states, there are S^9 rules for a Moore neighborhood CA and S^5 rules for a von Neumann neighborhood CA. (The total number of transition functions and hence CAs with S states using the Moore neighborhood is then S^{S^9}.) The number of activated rules along with the number of states gives a rough measure of complexity for the CA. One constraint in converting a CA to a more fault-tolerant form is the level of complexity introduced. Some redundancy has to be introduced to improve reliability, but it is also useful and important to give some consideration to the minimal necessary redundancy.

The rules determine a CA, but in order for the CA to do useful work or display useful behavior, we also need an *initial configuration*. This is simply a finite set of cells containing at least one non-quiescent state. The initial configuration and the set of rules together determine the fault-tolerance of a CA. For example, in the Game of Life the set of rules (using the Moore neighborhood) basically quantifies the two intuitive rules:

1. Any "dead" cell becomes alive if it has exactly 3 "live" neighbors.

2. Any "live" cell stays alive only if it has 2 or 3 "live" neighbors.

With these rules, an initial configuration forming a 2×2 block of live cells, is fault-tolerant in the sense that if any of the four become dead, the configuration repairs itself. On the other hand, a *glider* configuration, which will move diagonally across the cellular array in 4 steps, is not fault-tolerant. If any of the cells die, the diagonal movement fails. Cells which are particularly critical for the continued successful operation of the CA are referred to as *essential cells*.

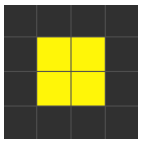

Figure 1: A simple block in the Game of Life that repairs the loss of any single one of its cells.

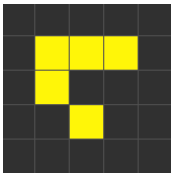

Figure 2: A glider in the Game of Life fails to continue its diagonal movement pattern when any of its cells die.

Cell Lifetime

In order to specify the source of faults in CAs, we use machine lifetime as an analogy. Each cell is a finite automaton with a finite lifespan. At the end of a cell's lifetime, it is immediately replaced and the new cell begins in the quiescent state. Probabilistically, if a cell has probability p of failing on any given time step and if we assume each time step is independent (more theoretical than practical), then the expected lifetime of the cell is $1/p$. We assume that all cells in an array have the same lifetime. On each step of the CA simulation, each cell has probability p (independently) of failing and hence needing replacement. We refer to this as a $p - death$ environment.

Note that there are many ways of introducing faults into a CA. A fault might mean that a cell spontaneously transitions to another state rather than to the quiescent state as in the lifetime model. (Notice that the 2×2 block in the Game of Life CA is not tolerant to spontaneous transitions to active states.) In some previous work, the fault model assumes that there is a maximum number of faults in each spatial region of the CA. Yet another model might allow periods of no faults giving the CA time to repair. However, in this study we settle on the lifetime model and allow each cell to independently experience a fault with some common probability.

Test Cases

To test a CA for fault tolerance, we first identify a specific task that the CA should complete. For example, we might

decide that the key function of the Langton loop CA is to produce one additional copy of itself which it may do in n steps. So we run the CA for n steps in a $p - death$ environment and then remove the threat of failures for an additional k steps. If the CA effectively repairs itself, it is counted as a success. The proportion of successes is then recorded as the reliability or fault tolerance.

Note that there are three sources of ambiguity here. First, it is not always obvious when a CA has completed a task; for a self-reproducing CA, it may be obvious, but for a digital gate simulation, perhaps only the correct signal needs to pass through the gate. Second, the number of extra steps k is not well-defined; in some test cases we took $k = 0$ and in others we used $k = n$. Finally, in practical uses of CAs, it is not necessary for the final state to exactly match the CA with no faults. In other words, it may only be a sub-configuration, like the digital signal through a gate, that needs to be correct.

To test our procedure for constructing fault-tolerant CAs, we focused on the following CAs with von Neumann neighborhoods (see `http://cs.coloradocollege.edu/~mwhitehead/metacells` for .RLE and .table files).

- Single Transition (Figure 3): This is the most primitive CA since it simply starts with a configuration of five cells and transitions (using the rules of any CA) to the new state for the central cell. Testing this configuration gives us information for predicting how larger CAs may perform. Experiments are considered successful for this CA if the center cell is correct after the single transition.

Figure 3: Single transition structure

- Glider (Figure 4): Unlike the glider configuration in the Game of Life which uses a Moore neighborhood, this glider functions with a von Neumann neighborhood and three rather than two states. Figure 4 shows the stages in moving from right to left. Experiments are considered successful for this CA if the glider is intact after flying to the left for four simulation steps.

Figure 4: Glider Stages: moves from right to left

- Coral (Figure 5): This is a slightly more complicated CA whose starting configuration is simple, but it produces a symmetric structure that continually grows in size (Figure 5 also shows the structure after eight steps). Experiments are considered successful for this CA if the coral structure is intact after eight simulation steps.

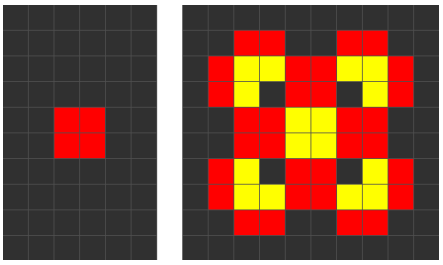

Figure 5: Coral: Start(left) and after 8 steps (right)

- XOR Gate (Figure 6): This CA simulates the digital XOR gate with two inputs and one output. The final configuration after 14 steps is shown in Figure 7. Experiments are considered successful if the correct signal is sent out to the end of the structure and the gate is intact.

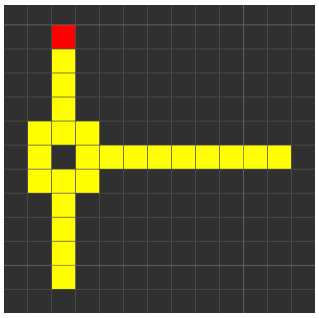

Figure 6: XOR Gate Starting Structure – Input 1 at top and input 0 at bottom

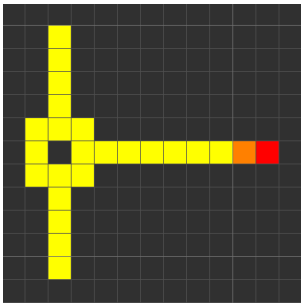

Figure 7: XOR Gate after 14 steps

- Byl's Loop (Figure 8): One of the key self-reproducing CAs. After 25 steps, the evolving configuration contains two copies of the initial configuration (Figure 9). Experiments are considered successful for this CA if the loop can correctly copy itself one time.

Steven Janke, Matthew Whitehead (2015) Practical Fault Tolerant 2D Cellular Automata. Proceedings of the European Conference on Artificial Life 2015, pp. 158-165

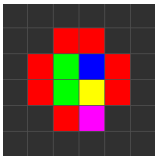

Figure 8: Byl's loop starting structure

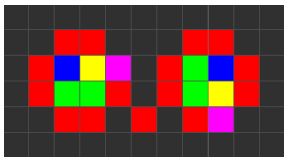

Figure 9: Byl's loop after one successful reproduction (25 steps)

- Langton's Loop (Figures10, 11): This is a larger configuration that reproduces after 151 steps. As with Byl's loop, experiments with Langton's loop are considered successful one additional copy of the loop is produced correctly.

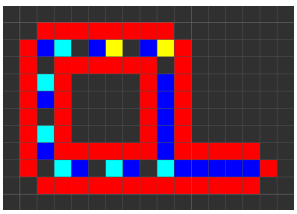

Figure 10: Langton's loop starting structure

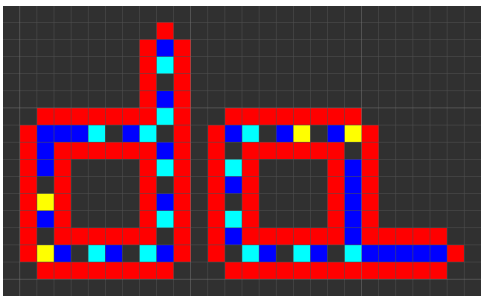

Figure 11: Langton's loop after one successful reproduction (151 steps)

Metacells

Our procedure for converting CAs into reliable systems relies heavily on the concept of a *metacell*. These are square arrays of cells that replace each of the single cells in the original CA. The point is to provide informational redundancy to improve fault tolerance, but in order to do so, the rules governing transitions in the original CA have to be altered to now simulate the original operations on square arrays of cells. One constraint is that the converted CA must still be a genuine CA; each cell in the conversion follows a common set of rules. The transformation to a converted CA (called a meta-CA), theoretically works on either Moore or von Neumann neighborhoods, but it is much more straightforward for original CAs using the von Neumann neighborhood.

There are several ways to design a metacell and our approach evolved as we tried to balance the achieved redundancy with the number of states in the CA. We set an arbitrary threshold of 256 states since that is the limit of the Golly simulator (Trevorrow and Rokicki (2013)). This practical limit did help focus our work on the efficiency of metacell design, but it is clear that using a larger simulator has its advantages. For more complex test cases, we had to approximate how successful the tolerant meta-CAs would be.

Figure 12 shows an example metacell of size 3. The 3x3 grid in the interior (yellow cells) of the metacell stores the redundant information that collectively represents a state from the original CA cell. A common flood-fill algorithm can keep the state information intact despite random faults. The *border cells* (red cells in the figure) serve as a boundary between metacells and can also be used as a built-in clock to synchronize the operation of the meta-CA. Meta-CAs operate at a slower speed than their normal, single-cell CA equivalents. The application of a single rule in an original CA is translated to several steps in a meta-CA. These additional steps are required in order to allow metacells to recover information disrupted by cell faults and to propagate state information from the neighboring metacells.

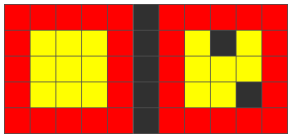

Figure 12: Bordered metacell of size 3 (left) and bordered metacell with two dead interior cells (right)

The weakness of this first metacell design is that there are several *essential* cells. Both the border cells and the signals carrying state information into the interior of a metacell have no redundancy. The success of this design relies on keeping the ratio of essential cells to total cells rather small.

In order to reduce the essential cells, we altered the flood-fill algorithm to simulate mixing paint. The states of neighboring cells are considered paint colors that flow into a metacell. Whenever two such colors are neighbors, the corresponding cells transition to a state representing the mix of the two colors. Similarly, whenever two mixed colors meet or a mixed color and a third plain color meet, there is a transition to a cell representing a mix of all colors involved. This ensured that there was not just a single cell carrying the message on any clock step. The number of essential cells was

decreased and the success of the meta-CA increased. However, the boundary cells remained and consequently there were still vulnerable cells (not necessarily essential) in the meta-CA. The mixed-paint principle unfortunately requires on the order of n^4 additional states where n is the number of states in the original CA.

Positional Metacells

Eliminating the border cells improves fault tolerance and our final approach, which we call *positional metacells*, was the most successful. Positional metacells encode a cell's position in the metacell along with the original state information. A cell would then have information showing that it was in the upper-left corner of the metacell, for example. Positional metacells use Moore neighborhoods to simulate an original CA that used von Neumann neighborhoods. Once again, more states are needed to cope with the positional information. Nevertheless, the positional metacell design does lead to improved fault tolerance since failed cells can be repaired based on their relative position in relation to other correct cells.

The simplest kind of positional metacell is a 2x2 grid as shown in Figure 13. Each of the four cells comprising the metacell contains information about its location within the metacell (signified by the numbers in the figure) and the state value from the original CA.

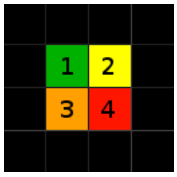

Figure 13: A 2x2 Positional Metacell. Each interior cell is encoded by its position within the metacell.

The metacell transitions take two steps. On the first step, each metacell interior cell changes to a state that represents the combination of its two neighbors belonging to other adjacent metacells. This is done using the color-mixing idea from above. Note that there is redundancy in each of the incoming state messages. For example, the upper-left cell changes to a state representing the combination of the metacell's top and left metacell neighbors and the upper-right cell encodes the metacell's top and right neighbors. This means that the top metacell neighbor's value is encoded in two places. This helps eliminate any essential cells and provides a more reliable structure.

On the second step, the original CA's transition is computed using the four mixed color values. Because of the redundancy of the encoded neighbor information, if any single cell dies, then the original transition can still be calculated. In fact, as long as two cells that are diagonally opposite one another are intact, then the original transition can be performed. The output of the original transition is then positionally encoded throughout the four interior cells and one metacell transition is complete.

These metacells do not need separate border states because of the positional encodings, but the positional encodings do require four states per original CA state. They also require states to represent all combinations of mixes of three original states since each positional cell must encode its own value along with the two neighbor metacell cells that it borders. The final number of required states is $4n^3 + 4n + 1$, where n is the number of original states.

Because of their lack of essential cells and short clock cycle of two steps, these 2x2 positional metacells were the most fault tolerant design we experimented with. The results for the various test cases reported below compare the reliability of these structures versus their original CA counterparts.

Converting Existing CAs to Positional Metacell CAs

One of the appeals of using positional metacells for fault tolerance is that existing CAs can be converted automatically into their meta-CA equivalents. We have developed a set of conversion scripts to perform this process. (See `http://cs.coloradocollege.edu/~mwhitehead/metacells` for scripts.)

The first part of the conversion process involves changing the original CA structure into a meta-CA structure using the appropriate new state values. For example in the 2x2 positional metacell case, a single cell with state value 3 from the original CA must be converted into four cells that are positionally encoded as *original state 3 in the upper-left corner, original state 3 in the upper-right corner, original state 3 in the lower-left corner, and original state 3 in the lower-right corner*. Figure 14 shows a simple 3-cell CA converted to the corresponding meta-CA.

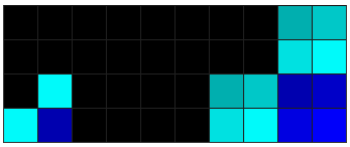

Figure 14: Original CA (left) and Positional Meta-CA (right)

Once the structure of the original CA has been converted, then the original ruleset needs to be converted to include the rules for meta-CA operation. Meta-CA operation requires several different types of rules. In particular, rules must be included to encode the combination of neighbor states, ignore and repair missing states (if possible), and perform the original CA's transitions.

First, rules must be added to convert each positionally-encoded cell to a new state representing the combination of its two neighbors from adjacent metacells. For example, the

Steven Janke, Matthew Whitehead (2015) Practical Fault Tolerant 2D Cellular Automata. Proceedings of the European Conference on Artificial Life 2015, pp. 158-165

A cell in Figure 15 needs a rule to transition to the state that represents the combination of starting state A, X above, and Y to the left. Similar transition rules are added for each interior cell position and all combinations of possible external neighbor states.

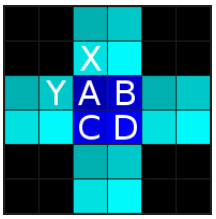

Figure 15: A, B, C, and D form one metacell. A's external neighbors are X and Y and its internal neighbors are B, C, and D.

Second, rules must be generated to ignore dead cells when possible while still performing the correct transitions. Again using Figure 15 as an example, if cell A were dead it can still properly transition to the state that encodes the combination of X and Y as long as one of its internal neighbors is intact. So if A is dead, but B is intact, then cell A can correctly transition to the combination of X and Y since B *is on its right*. This is the added fault tolerance of using the positional encoding scheme.

Finally, rules are added to perform the original CA's transitions. These transitions occur on the second step of the meta-CA's cycle. Suppose the original CA had the rule (1,2,3,2,4,5). That is, if a cell were in state 1 and had neighbors starting at the top and going clockwise of 2, 3, 2, and 4, then it should change to state 5. Using the labels in Figure 15, the cells would have the following mixes after the first step of the meta-CA's simulation. Cell A would have a mix of the original state 2 from above and state 4 from the left, cell B would have a mix of state 2 from above and state 3 from the right, cell C would have a mix of state 2 from below and state 4 from the left, and cell D would have a mix of state 2 from below and state 3 from the right.

These cells then need to transition to the positionally-encoded state that corresponds to the original CA's new state after the transition. For this example, A would need to transition to the state representing the *original state 5 in the upper-left corner*. In order to perform these kinds of transitions, rules must be included to decode the mixes of states in the context of the original rules. For example, cell A has information about the states from above and left and cell D has information about the states from below and right, so cell A gets a rule that transitions to its new state (*original state 5 in the upper-left corner*) when it has D's state as a lower-right neighbor. Similar rules are added for the other positions in the metacell and for all other possible internal neighbor states.

Notice again that external neighbor state information is redundantly encoded. For example, both cells A and B encode the original state 2 from above. If at least one of these two cells is intact, then the neighbor information from above is not lost and the original transition can be successfully performed.

Results

Using the test cases mentioned above, we ran a series of experiments to test the fault tolerance of our proposed meta-CAs against normal CAs. The results listed here are for the 2x2 positional meta-CAs, as they were the most successful model that we tested. We used a range of p-death environments and ran 1000 trials per test case. For larger structures that require too many states for the Golly simulator, we list predicted success rates calculated by using the experimental success rates for single cell transitions raised to the power of the number of required transitions for the given CA. To ensure accurate single cell transition success rates, we performed 200,000 single cell transition trials.

p	Original	Meta-CA
0.00001	99.9980	100.0000
0.00002	99.9960	100.0000
0.00005	99.9900	100.0000
0.00010	99.9800	100.0000
0.00020	99.9600	100.0000
0.00050	99.9000	99.9999
0.00100	99.8001	99.9996
0.00200	99.6004	99.9984
0.00500	99.0025	99.9901
0.01000	98.0100	99.9604
0.02000	96.0400	99.8432
0.05000	95.0250	99.0494
0.10000	81.0000	96.3900

Table 1: Success rates for a single cell/metacell transition.

In Table 1, the Meta-CA success rates were calculated using the formula $1-(p^4+4p^3(1-p)+4p^2(1-p)^2)$. This follows from the binomial distribution where p^4 is the probability of all four cells being dead, $4p^3(1-p)$ is the probability of exactly three cells being dead (there are four ways for this to occur) and $4p^2(1-p)^2$ is the probability of two non-diagonal cells being dead (there are also four ways for this to occur). A simulator experiment confirmed these results.

The tables of results list the percentage of trials that were successful, with success defined as above for each test case. The results show that meta-CAs provide an added level of fault tolerance across a variety of CA structures and levels of p-death environments. In general, the greater the number of cells and required transitions in the CA structure, the greater the benefit of using a meta-CA representation. This is to be expected since metacells outperform regular CA cells for each individual cell transition.

The type of p-death environment makes a difference in

the degree of added fault tolerance provided by meta-CAs. For very low values of p, both regular CAs and meta-CAs are likely to be successful. On the other hand, very high values of p usually cause both regular CAs and meta-CAs to fail. Intermediate p values show the greatest benefit of using meta-CAs. For example, using the coral-like structure with $p = 0.005$, the meta-CA was successful 97.7% of the time while the normal CA was only successful 50.3% of the time.

p	Original	Meta-CA
0.00001	100.0	100.0
0.00002	100.0	100.0
0.00005	100.0	100.0
0.00010	100.0	100.0
0.00020	100.0	100.0
0.00050	99.5	100.0
0.00100	99.4	100.0
0.00200	98.5	100.0
0.00500	97.1	99.8
0.01000	92.0	99.1
0.02000	85.6	97.5
0.05000	71.6	86.9
0.10000	41.0	52.0

Table 2: Success rates for a von Neumann neighborhood glider to complete one cycle in 4 steps.

p	Original	Meta-CA
0.00001	100.0	100.0
0.00002	99.5	100.0
0.00005	99.2	100.0
0.00010	98.3	100.0
0.00020	97.1	100.0
0.00050	93.2	100.0
0.00100	85.3	99.7
0.00200	76.8	99.4
0.00500	50.3	97.7
0.01000	25.4	89.2
0.02000	5.5	63.0
0.05000	0.1	5.8
0.10000	0.0	0.0

Table 3: Success rates for a von Neumann neighborhood coral to be intact after 8 steps.

Complexity of Meta-CAs

All of our constructions centered on duplicating information in the original CA. The most straightforward approach is to introduce more cells for redundancy as is done in a meta-cell. However, in order to benefit from the redundancy there must be a mechanism for combining information in order to recover from information lost in faulty cells. For example, border cells were used to keep information within the bounds of a particular metacell and signals were transmitted

p	Original	Meta-CA (predicted)
0.00001	99.6	100.0
0.00002	99.2	100.0
0.00005	98.0	100.0
0.00010	97.0	100.0
0.00020	93.6	100.0
0.00050	84.5	99.7
0.00100	72.9	99.7
0.00200	48.4	99.5
0.00500	18.8	97.0
0.01000	2.7	88.6
0.02000	0.1	59.4
0.05000	0.0	4.0
0.10000	0.0	0.0

Table 4: Success rates for a von Neumann neighborhood XOR gate to produce the correct output value.

p	Original	Meta-CA (predicted)
0.00001	99.1	100.0
0.00002	98.7	100.0
0.00005	97.8	100.0
0.00010	95.0	100.0
0.00020	91.1	100.0
0.00050	76.2	99.5
0.00100	57.0	99.5
0.00200	36.2	99.2
0.00500	6.6	95.3
0.01000	0.5	82.6
0.02000	0.0	43.8
0.05000	0.0	0.6
0.10000	0.0	0.0

Table 5: Success rates for a Byl loop to reproduce once.

p	Original	Meta-CA (predicted)
0.00001	84.3	100.0
0.00002	70.5	100.0
0.00005	43.9	100.0
0.00010	17.3	100.0
0.00020	3.4	100.0
0.00050	0.0	83.1
0.00100	0.0	83.1
0.00200	0.0	75.7
0.00500	0.0	18.9
0.01000	0.0	0.1
0.02000	0.0	0.0
0.05000	0.0	0.0
0.10000	0.0	0.0

Table 6: Success rates for a Langton loop to reproduce once.

Steven Janke, Matthew Whitehead (2015) Practical Fault Tolerant 2D Cellular Automata. Proceedings of the European Conference on Artificial Life 2015, pp. 158-165

between metacells. Since cells in a CA are homogeneous, there cannot be border cells or signals without additional states. This suggests that some measure of CA complexity based on the number of states might determine the minimal number of additional states necessary to achieve a given level of fault tolerance.

A recent paper (Lui et al. (2015)) describes work that uses both the Shannon measure of complexity (based on entropy) and the Kolmogorov measure (based on the length of code description) to construct a hybrid measure that shows some promise in distinguishing one dimensional cellular automata. This measure takes into account both the CA rules and the starting configuration. In our work, it is not yet clear that there is a direct relationship between the starting configuration and the level of achieved redundancy. Clearly, some configurations are more brittle than others, but our procedure for introducing redundancy is independent of the starting configurations and it may be that more simple measures of the CAs structural complexity could shed light on the number of additional states necessary to add appropriate redundancy. Currently, we plan to investigate the effect of active rules (those that change the central cell state) on the fault tolerance of our meta-CAs.

Conclusions

This work offers evidence for the following CA design principles:

1. A practical general purpose algorithm can convert an original CA into a more fault-tolerant meta-CA. The definition of fault-tolerance in this context refers to cells with a lifetime that is geometrically distributed.

2. The general design of meta-CAs involves the inclusion of metacells, and this research identified one promising metacell type: positional metacells. First attempts to design metacells required too many essential cells in mechanisms to manage the redundancy. The fault tolerance did not scale well to higher amounts of redundancy. Our positional metacells proved to be the most efficient (number of states).

3. Complexity of the derived metacell is an issue. The amount of redundancy and the way it is managed introduces many new states to the original CA. Positional metacells kept the number of additional states within reason, but we hypothesize that there may be hopefully simple measures of complexity that might lead to bounds on the number of additional states necessary to reach conclusions about the minimum number of additional states.

Acknowledgements: The authors wish to thank the reviewers for their useful suggestions.

References

Byl, J. (1989). Self-reproduction in small cellular automata. *Physica D*, 34:295–299.

Gacs, P. (1986). Reliable computation with cellular automata. *Journal of Computer and System Science*, 32:15–78.

Gacs, P. (1989). Self-correcting two-dimensional arrays. *Advances in Computing Research*, 5:223–326.

Harao, M. and Noguchi, S. (1975). Fault tolerant cellular automata. *Journal of Computer and System Sciences*, 11:171–185.

Langton, C. G. (1984). Self-reproduction in cellular automata. *Physica D*, 10:135–144.

Lui, L., Terrazas, G., Zenil, H., Alexander, C., and Krasnogor, N. (2015). Complexity measurment based on information theory and kolmogorov complexity. *Artificial Life*, 21. (to appear).

McCann, M. and Pippenger, N. (2008). Fault tolerance in cellular automata at high fault rates. *Journal of Computer and System Sciences*, 74:910–918.

McCann, M. and Pippenger, N. (2013). Fault tolerance in cellular automata at low fault rates. *Journal of Computer and System Sciences*, 79:1136–1143.

Nishio, H. and Kobuchi, Y. (1975). Fault tolerant cellular spaces. *Journal of Computer and System Sciences*, 11:150–170.

Oros, N. and Nehaniv, C. L. (2007). Sexyloop: Self-reproduction, evolution and sex in cellular automata. In *The First IEEE Symposium on Artificial Life*, pages 130–138.

Perrier, J.-Y., Sipper, M., and Zahnd, J. (1996). Toward a viable, self-reproducing universal computer. *Phys. D*, 97(4):335–352.

Sayama, H. (1998). Introduction of structural dissolution into Langton's self-reproducing loop. In *Proceedings of the 6th International Conference on Artificial Life (ALIFE-98)*, pages 114–122, Cambridge, MA, USA. MIT Press.

Sayama, H. (1999). Toward the realization of an evolving ecosystem on cellular automata. In *In M. Sugisaka and H. Tanaka (Eds.), Proceedings of the Fourth International Symposium on Artificial Life and Robotics (AROB 4th 99)*, pages 254–257.

Sayama, H. (2004). Self-protection and diversity in self-replicating cellular automata. *Artificial Life*, 10(1):83–98.

Trevorrow, A. and Rokicki, T. (2013). Golly cellular automata software. `http://golly.sourceforge.net/`. Version 2.6, 2013.

von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies (ed. C.E.Shannon, J.McCarthy)*, pages 43–98. Princeton University Press.

Steven Janke, Matthew Whitehead (2015) Practical Fault Tolerant 2D Cellular Automata. Proceedings of the European Conference on Artificial Life 2015, pp. 158-165