Summer 8-6-2020

Modeling Braids with Space-Varying and Time-Varying Stranded Cellular Automata

Brian Chan
chanb@rose-hulman.edu

Modeling Braids with Space-Varying and Time-Varying Stranded Cellular Automata

Brian Chan

Faculty Mentor: Joshua Holden

Date: August 7th, 2020

Abstract

Braids in a traditional sense and braids in a mathematical sense are wildly different outlooks on the same concept. Using cellular automata to represent and analyze braids is a way to bridge the gap between them. Joshua and Lana Holden and Hao Yang have previously worked on developing and expanding upon a Stranded Cellular Automata (SCA) model capable of representing many different braids and weaves. Continuing their work, we were able to devise a more user-friendly method for interacting with the model such that even those without a mathematical background can construct and analyze braids of their own. This paper will also discuss the addition of space-varying and time-varying rulesets to expand upon the types of braids and weaves the SCA model is able to represent.

1. Introduction

Braids can be found in many places, from braided cables to commonplace hairstyles. The appeal of braids can be attributed to their repeating patterns and intuitive construction. However, when braiding one does not focus on directly creating patterns but rather on the instructions of which strand goes over which. The mathematical side of braiding does the opposite [1] ; looking solely at patterns that emerge from braids instead of focusing on the steps to recreate a braid. This paper aims to provide a middle ground between the two extremes by using cellular automata to represent braids in a way that makes them simple to recreate but also convenient to analyze.

Cellular automata are mathematical models that consist of a grid of cells evolving through discrete steps in time. As the name implies, they consist of cells with states that are “neighbors” to each other and change their states based on the states of their neighbors. The set of all neighbors that influence a cell’s state is defined as the cell’s “landscape”. [2] All the cells present at during any discrete time $t > 0$ belong to the same “generation”, and the states of the cells that belong to the generation at time t are determined by the landscapes of the generation at time $t - 1$. The generation at $t = 0$ is defined as the initial condition and has no predecessor generations. [3]

2. The Stranded Cellular Automata Model

In the case of the Stranded Cellular Automata (SCA) created by Joshua and Lana Holden [4], each cell has 8 possible states and a landscape of 2 neighbor cells that determine its state. Each cell is generated based on a set of rules applied to its landscape. Figure 1 shows the landscape cells in highlighted in red and the resulting new cell in highlighted in blue.

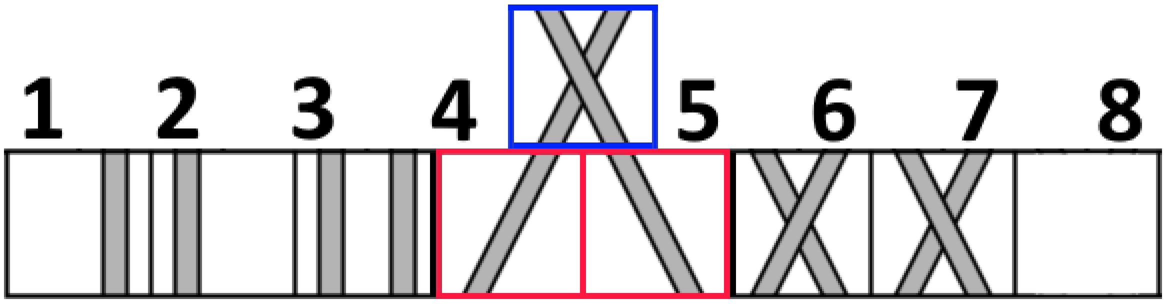


Figure 1: All 8 cell states, with an example neighbor pair generating a new cell.

In order to distinguish between the two types of crossings, we will refer to the crossing with the strand on top resembling the slant in the letter Z as a "z-cross" and the opposite crossing with the strand on top resembling the slant in the letter S as a "s-cross". Figure 2 shows a visualization of this concept.

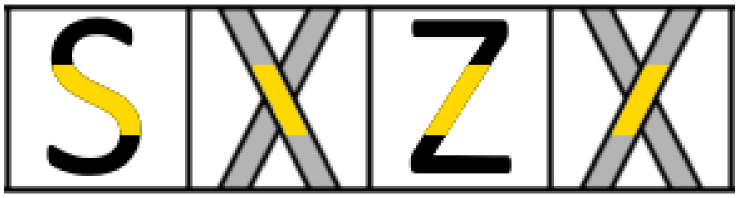


Figure 2: The letter S next to a s-cross, and the letter Z next to a z-cross. The relevant sections of each are highlighted.

The calculation of each cell's state based on its landscape is split into two different rules: the "turning rule", which dictates whether or not strands will slant, and the "crossing rule", which dictates which strand goes over the other in the case of a cross. Instead of covering every single case, each rule deals with a more general set of cases, where multiple cell states are equivalent to each other if they exhibit the same features. The turning rule cases include straight, slanted, and absent cells, and the crossing rule cases include s-cross, z-cross , and no cross cells. See Figures 3 and 4 for details.

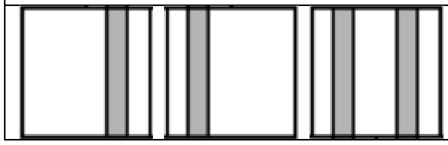
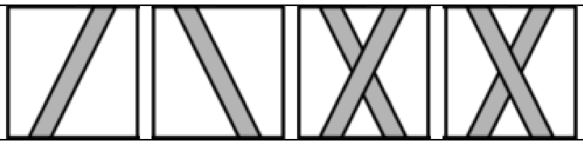
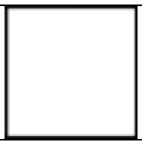
Turning Rule		
Straight Cells	Slanted Cells	Absent Cells
		

Figure 3: Possible cells for the 3 turning rule cases.

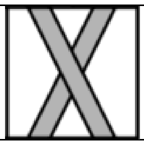
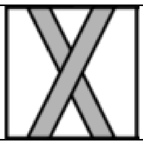
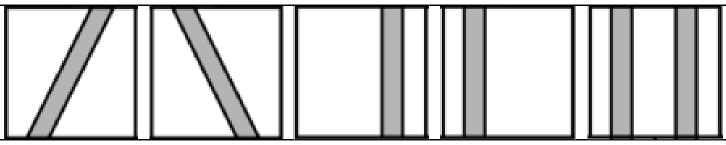
Crossing Rule		
S-Cross	Z-Cross	No Cross
		

Figure 4: Possible cells for the 3 crossing rule cases.

Because each landscape consists of two cells, the number of possible landscapes comprised of the 3 different cases would be 3*3 = 9 different landscapes. Each one of these 9 landscapes controls the status of the new cell. For the turning rule, every landscape determines whether the new cell has straight strands or slanted strands. For the crossing rule, every landscape determines whether the new cell has a s-cross or z-cross. We can label each of these landscapes with a number ranging from 0 to 8 and based on the status it determines we can assign it a 0 or a 1. The number 0 corresponds to straight strands for the turning rule and s-crosses for the crossing rule, while the number 1 corresponds to slanted strands for the turning rule and z-crosses for the crossing rule. A visual example can be seen in Figure 5 for the turning rule and Figure 6 for the crossing rule.

Since each of these bits is labeled 0-8, it is possible to write out each rule in decimal notation. For example, instead of writing turning rule 101000100, it is more concise to write turning rule 324 (the equivalent base 10 number).

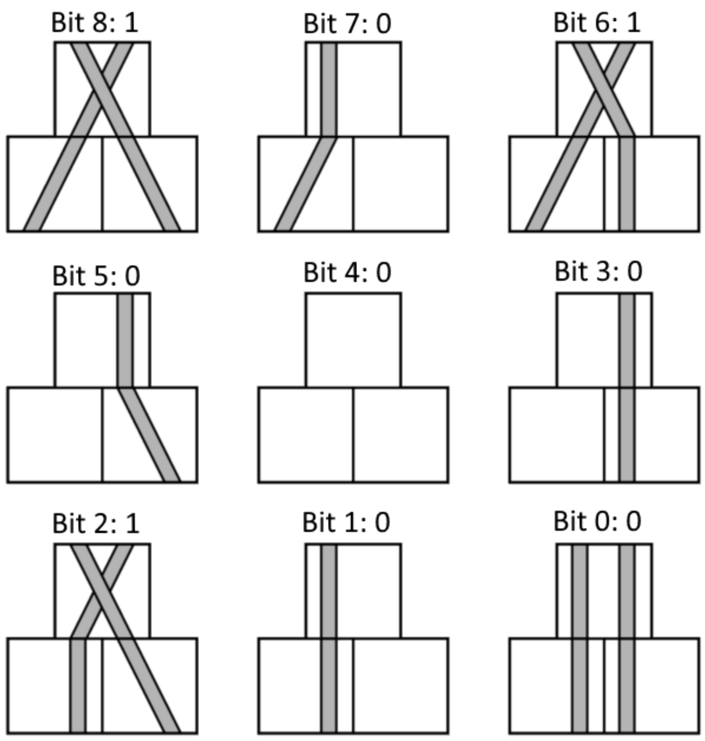


Figure 5: Turning Rule 324 (Binary 101000100)

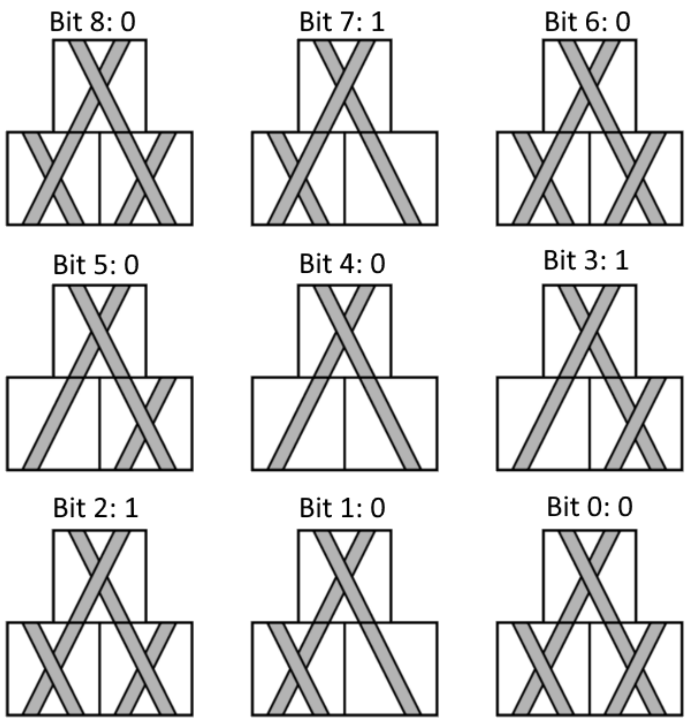


Figure 6: Crossing Rule 140 (Binary 010001100)

3. Representing Braids with Stranded Cellular Automata

We can use Stranded Cellular Automata to model various types of braids with different numbers of strands. According to Wolfram Mathworld, a braid is an intertwining of some number of strings attached to top and bottom “bars” such that each string never “turns back up”. [1] Braids, unlike weaves, have finite width because they reuse the same strands. This means that there is no need to let the border cells “wrap around” as Hao Yang defined the border cells in his work with weaves. [5] Instead, our border cells will act as absent cells that are not drawn in the figures below.

We started off by constructing physical models of the braids to analyze. We then transcribed the crossings and strands as their corresponding cell states in a Stranded Cellular Automata. Upon checking the output of each neighbor pairing, we were able to derive an initial condition, turning rule, and crossing rule that generated a braid identical to the model.

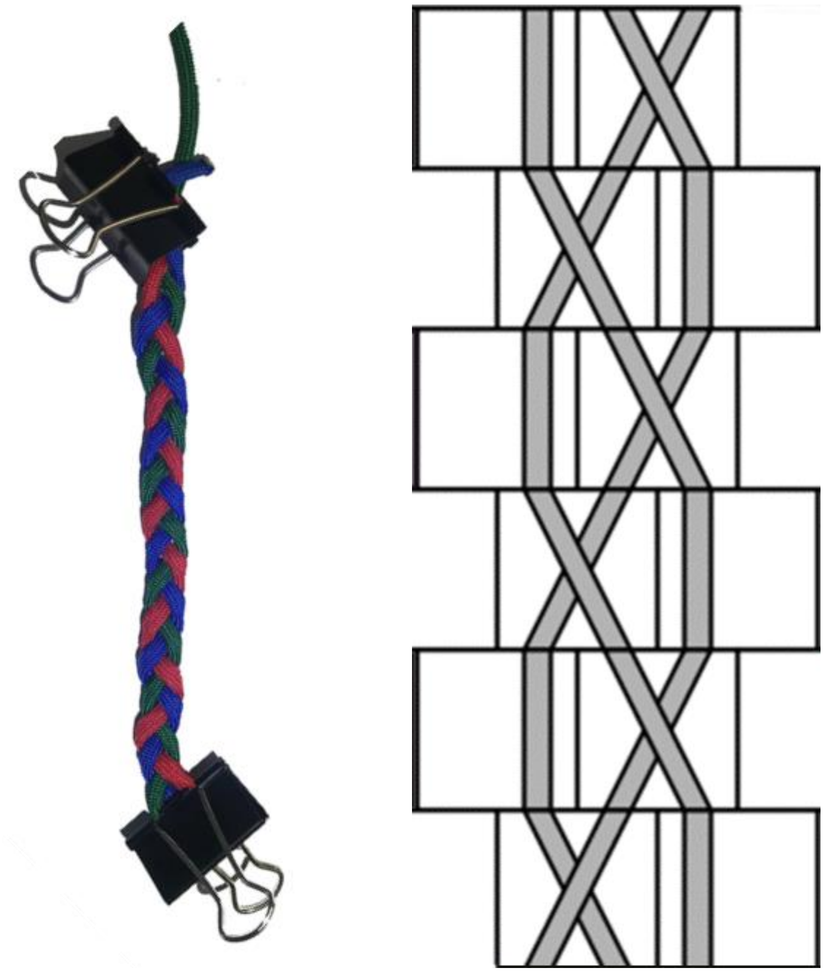


Figure 7: 3-Strand Braid and its SCA counterpart, Turning Rule 68, Crossing Rule 32 (68, 32)

To start, we analyzed the simple 3-strand braid commonly used for braiding hair [6] and found no issues with converting it into an SCA with Turning Rule 68 and Crossing Rule 32. Because a ruleset is comprised of a turning rule and a crossing rule, a shorthand method of writing these rulesets would be in an ordered pair format. For example, the ruleset for the simple 3-strand braid in Figure 7 would be written as (68,32). After analyzing the 3-strand braid, we decided to add another strand to add to the complexity. We found two 4-strand braids that were representable by SCA, a “flat” [4] and “square” [7] pair of braids that both used the same turning rule but different crossing rules. See Figures 8 and 9 for details.

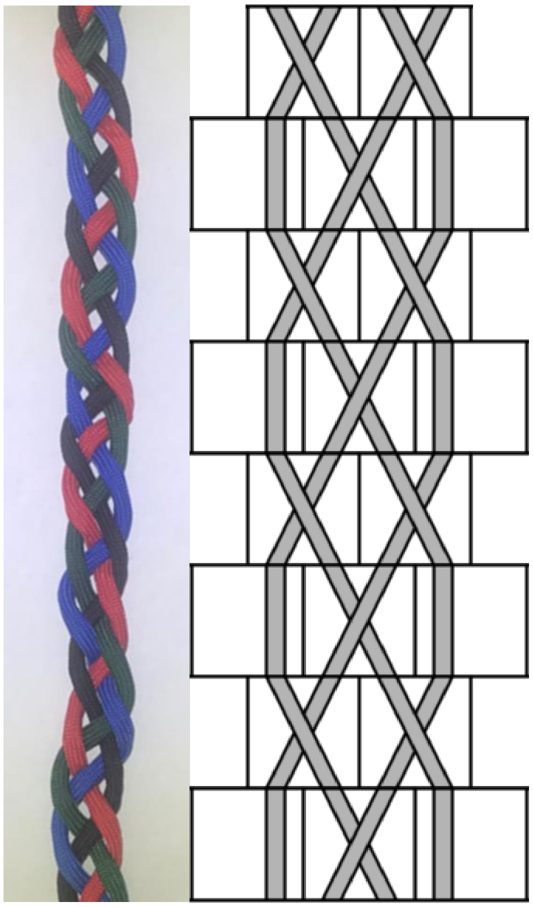


Figure 8: Flat 4-Strand Braid with SCA counterpart, Ruleset (324, 4)

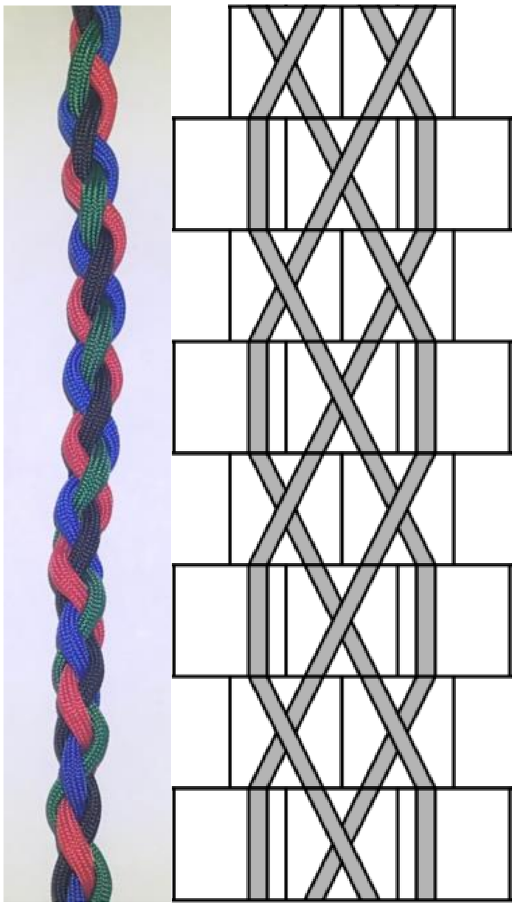


Figure 9: Square 4-Strand Braid with SCA counterpart, Ruleset (324, 140)

An interesting observation made when comparing 3-strand braids to 4-strand braids was the “backwards compatibility” of the turning rule shared by the two 4-strand braids we analyzed.

Bit Number	8	7	6	5	4	3	2	1	0	Decimal
3-Strand Turning Rule	0	**<u>0</u>**	**<u>1</u>**	**<u>0</u>**	0	0	**<u>1</u>**	0	0	68
4-Strand Turning Rule	**<u>1</u>**	**<u>0</u>**	**<u>1</u>**	**<u>0</u>**	0	0	**<u>1</u>**	0	0	324

Figure 10: Turning Rule Comparison, the underlined/bolded bits are the bits relevant to generating the braid's behavior.

Since the case that bit 8 governs in the turning rule does not appear in the 3-strand braid, the value of bit

8 is irrelevant in choosing a turning rule to represent the 3-strand braid. Therefore, it is possible to reuse the turning rule from the 4-strand braids to generate a 3-strand braid identical to the original. See Figure 10 for details.

For the case of braids with 5 strands, there was a lot more room for experimentation as different combinations of cells that previously could not be represented with only 3 or 4 strands emerged. To start, we applied the ruleset of the flat 4-strand braid (324, 4) to 5 strands. The result was that the braid became no longer flat as the number of s-crosses outnumbered the number of z-cross and made the braid start to twist. We observed that each generation of this braid had two crossings, so we altered the crossings of the braid to have equal numbers of s-crosses and z-crosses. We accomplished this in two different ways. First, we had the crossings alternate between 2 z-crosses and 2 s-crosses. Because each generation contained 2 slanting strands that alternated every generation, we referred to it as the "double slant" braid. A photo of the double slant braid and corresponding SCA are pictured in Figure 11.

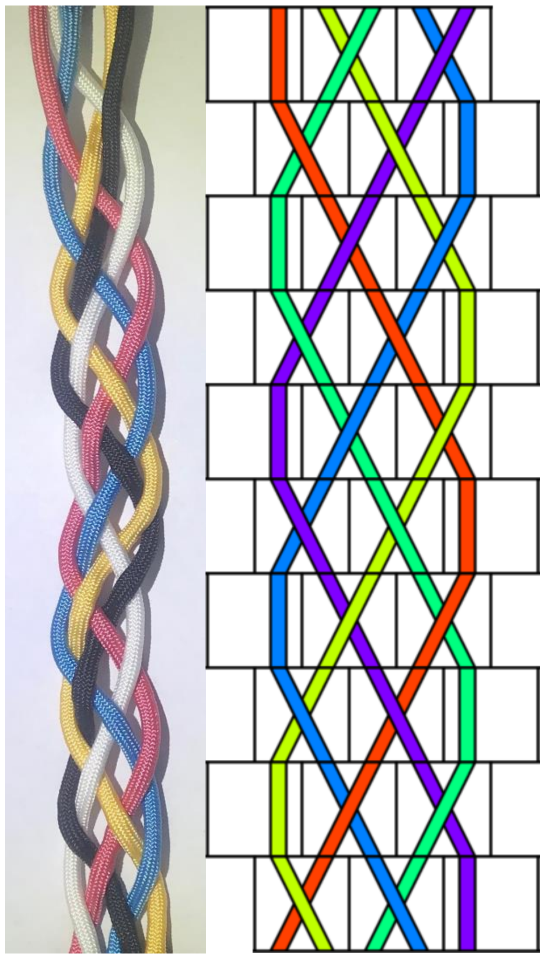


Figure 11: Double slant 5-strand braid with SCA counterpart, Ruleset (324, 6)

4. Space-Varying and Time-Varying Rulesets

Building off the previous braid that had generations that alternated between 2 z-crosses and 2 s-crosses, we attempted to construct a braid that had the same crossings for each generation without twisting. We decided upon having each generation contain a single s-cross adjacent to a single z-cross, resulting in a braid with the top strands exhibiting a “v-shaped” pattern as shown in Figure 12.

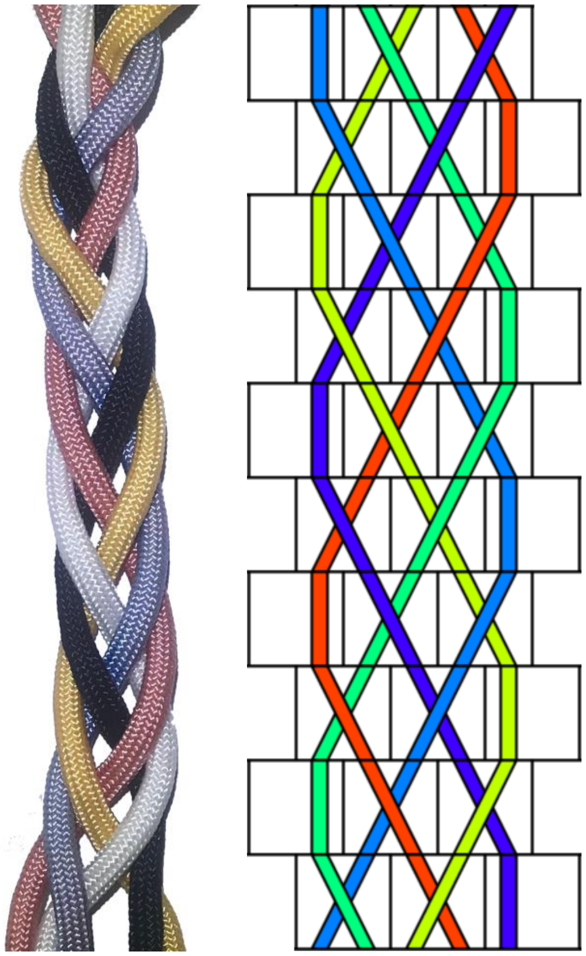


Figure 12: V-shaped 5-strand braid with SCA counterpart

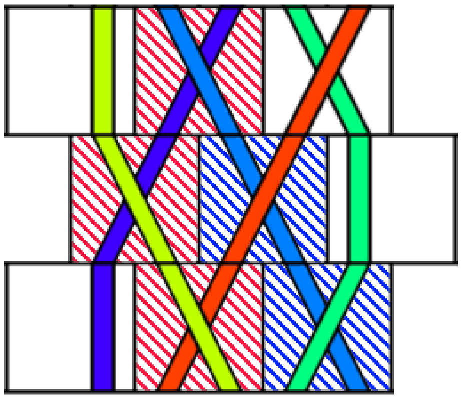


Figure 13: Zoomed-in view of the first 3 generations. S-crosses are highlighted red and z-crosses are highlighted blue. Note how the red-blue pairs generates different crossing types.

When analyzing the v-shaped braid, we encountered an issue with finding a crossing rule to represent the crossings. As shown in Figure 13, identical landscapes were generating different output crossings in different generations. To avoid the problem of landscapes generating conflicting crossings, we sought to make each landscape more distinct. We tried giving each strand in the braid its own unique color to make the currently conflicting landscapes differ from each other. If the landscapes that currently generate conflicting crossings get split into different landscapes based on the colors of their strands, each new landscape generating a different crossing type will no longer be a problem.

However, distinctly coloring each strand would require adding a lot of complexity to the rulesets that govern them. For example, Figure 14 shows the one colorless cell in the original model becoming 5 distinctly colored cells. The amount of complexity will be quantified in the following sections. It is important to note that since there were no conflicts with the turning rule representing the braid, we will only look at changes to the model that fix the crossing rule conflicts.

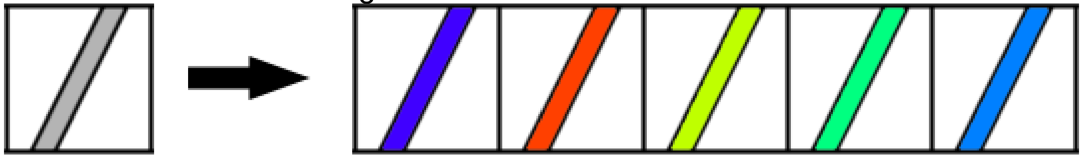


Figure 14: Conversion of a colorless no cross cell into 5 color variations

In Figure 6, the crossing rule is composed of 4 landscapes with 4 strands, 4 landscapes 3 strands, and 1 landscape with 2 strands. The formula for calculating the number of bits needed to represent a crossing rule for a *n*-strand braid with *n* colors is:

$$4 * \left(\frac{n!}{(n-4)!}\right) + 4 * \left(\frac{n!}{(n-3)!}\right) + 1 * \left(\frac{n!}{(n-2)!}\right)$$

Plugging in *n* = 5 for our 5 strands gets us 740 as the number of bits needed to represent a distinctly colored turning rule. Since each of the bits can be either on or off, there are 2^{740} possible crossing rules which is several orders of magnitude larger than the original 2^9 possible crossing rules for the non-color model. Using this colored model would result in rule numbers too unwieldy to reference. Additionally, the model created by this new ruleset would only work for braids with 5 or fewer strands. To model 6 or more strands a new model would need to be created.

To decrease the number of bits needed to represent the rules and make the model expandable past 5 strands, we decided to try coloring the strands with only two different colors. First, we numbered the strands 1-5 from left to right, and then we colored the odd strands red and even strands blue. Note that in Figure 15, the order appears different since the section pictured is not taken from the starting generation. Because repeated color strands were now possible, the formula for the number of bits needed to represent a crossing rule with number of colors *n*, where *n* is less than the number of strands in the braid it represents is:

$$4 * n^4 + 4 * n^3 + 1 * n^2$$

Plugging in *n* = 2 for our odd-even coloring scheme gets us 100 as the number of bits needed to represent an odd-even colored turning rule. The number of possible crossing rules for this method, 2^{100}, is still not feasible for analysis. The even-odd coloring method also failed to resolve all the landscape conflicts which further invalidates its usefulness.

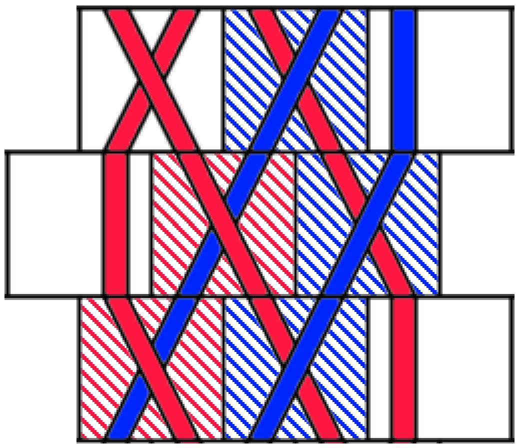
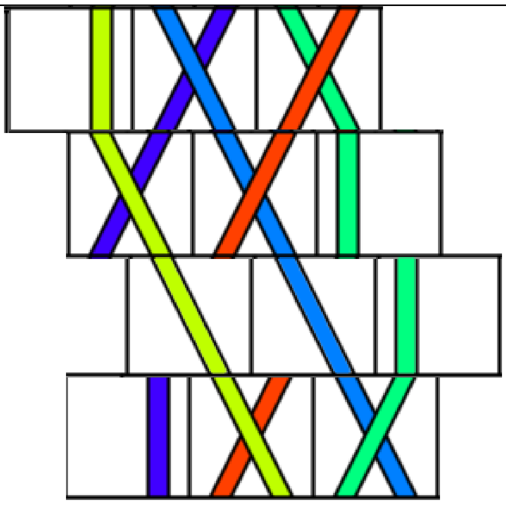


Figure 15: Section of the v-shaped 5-strand braid that contains the conflict; the middle generation here is the initial generation, the conflict occurs when the braid repeats

Pinpointing that the crossing rule conflict occurred because generation 1 generated generation 2 and vice-versa (Figure 15), we sought to add a “hold state” generation consisting of straight, non-crossing strands sandwiched between the two generations. Because the strands in the hold state generation do not cross, adding the hold state creates a braid with the same crossings as the original. This would make generation 1 generate the hold state instead of generation 2, and the hold state generation would generate generation 2.

Generation 1	 Figure 16: The staggering of the grid causing some strands to disconnect and generations to shift in the opposite direction due to the extra offset hold state generation
^ Generation 2	
^ Hold State	
^ Generation 1	

However, as shown in Figure 16, the staggering of the grid that the cells are generated in prevents us from connecting the two braid generations with a single hold state. When we added more hold states we encountered the same issues with crossing rule conflicts since the hold state could not generate both the additional hold state and generation 2.

Taking a step back, we observed that all the s-cross/z-cross neighbor pairs that produced s-crosses were located on the left side of the braid, and the s-cross/z-cross neighbor pairs that produced z-crosses were located on the right side of the braid. If we were to draw a zipper-like line through the middle of the braid, it would be possible to assign a different ruleset to each side of the line. See Figure 17 for details.

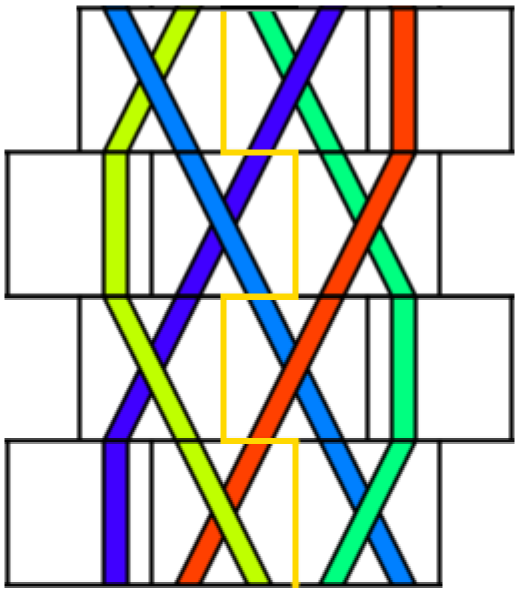


Figure 17: Zipper-shaped line dividing braid into two parts each with different rulesets

The ruleset used to generate a cell is based on the side of the zipper line that the new cell is on. In Figure 17, the bottom generation's middle and rightmost cell generate a cell that is to the right of the zipper line, so the righthand ruleset is used to calculate the crossing of the new cell. To represent the v-shaped braid, we used the ruleset (324, 128) for the left side and (324,129) for the right side. We called this a set of space-varying rulesets. This is because the cells that were in the same generation timewise but in different locations in space had different rulesets applied to them.

Armed with this new workaround, we decided to search for other braids that could not be represented by a single ruleset but were able to be represented by using space-varying rulesets. For a braid to fulfill these conditions, both sides of the braid needed to differ in the turnings and crossings exhibited. We revisited the simple 3-strand braid by braiding it loosely at first (Figure 18) and intertwining 2 new strands in the gaps of the 3-strand braid to make a new 5-strand braid.

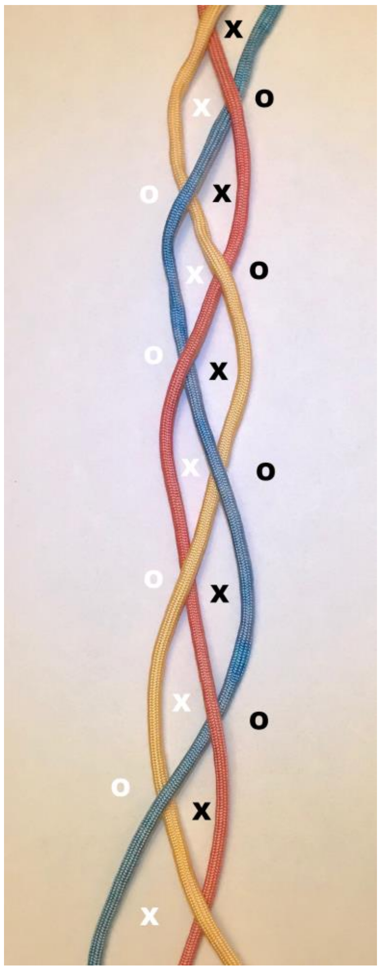


Figure 18: Loosened-up 3-strand braid. Note the alternating gaps between the strands. X's mark when an outer strand points away from the camera and O's mark when an outer strand points towards the camera.

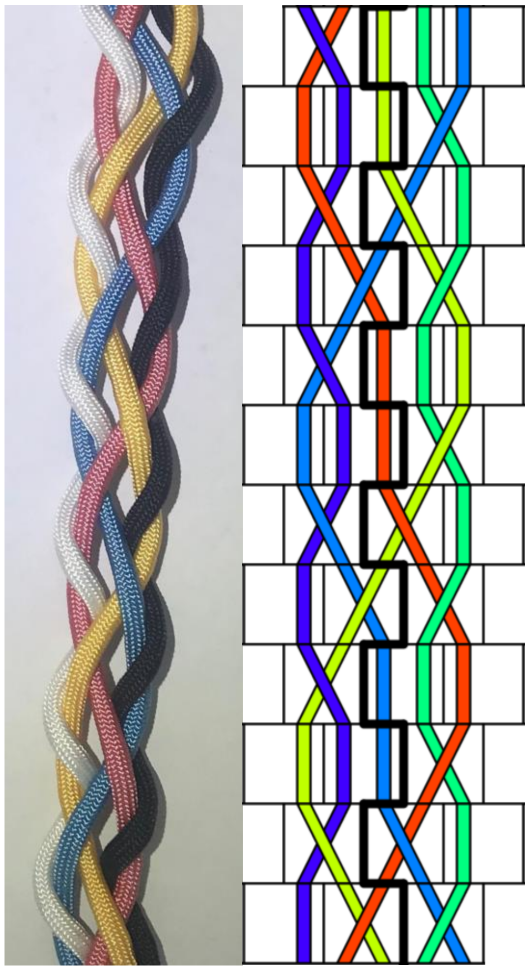


Figure 19: 3+2 over-only braid with SCA counterpart, left ruleset (69,2) and right ruleset (321,18).

In Figure 19, the original 3-strand braid is colored red, yellow, and blue; the new 2 strands are black and white. The weaving pattern for the outer strands for this braid was going through the gap into the page, moving towards the outside of the braid, and going through the next gap into the page again. This is illustrated in Figure 18, where x's mark when a strand points away from the camera and o's mark when a strand points towards the camera. The x and o markings are also color coded based on the colors of the outside strands in Figure 19. The rulesets that represented this braid were left ruleset (69, 2) and right ruleset (321, 18).

We then changed up the weaving pattern for the new strands to create a new 5-strand braid. Instead of only going through the gap into the page, the new strands alternate between going through the gap into and out of the page, as in Figure 20.

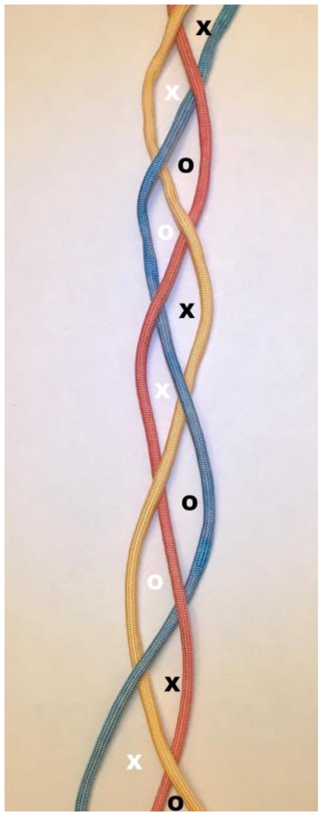
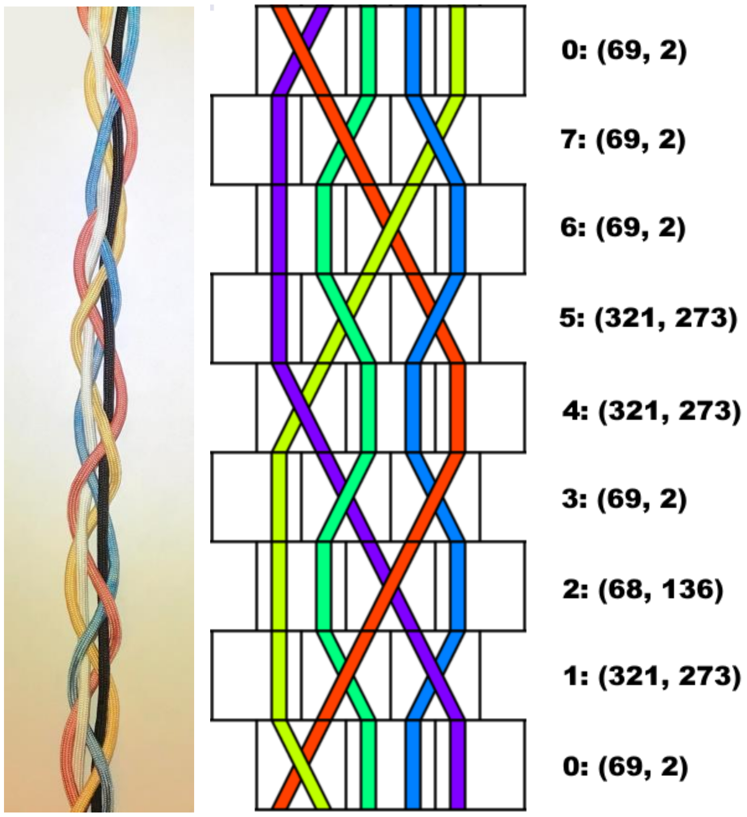


Figure 20: Loosened up 3-strand braid with weaving pattern markings.



A B

Figure 21: 3+2 over-under braid with SCA counterpart, time-varying rulesets are labeled to the right of the SCA.

The 5-strand braid created by changing the weaving pattern for the new strands ended up unrepresentable even when using the space-varying rulesets. Refer to the left side of the SCA in Figure 21B. From index 1 to 2 there is a landscape made of a straight strand and a slanting strand that generates a straight cell. From index 3 to index 4 the same landscape generates a slanting cell. This conflict involves the turning rule, but there is also a similar conflict on the right side of the braid that affects the crossing rule. From index 0 to 1 there is a landscape made of two no cross strands that generates a s-cross. From index 4 to index 5 the same landscape generates a z-cross.

Again, we observed that although the locations of the conflicts may have occurred on the same sides of the zipper line, they occur during different generations. Instead of splitting the braid's rulesets space-wise with a vertical line, we decided to split them time-wise with multiple horizonal lines running between the generations. To give every unique generation its own ruleset would take 8 different rulesets, so we applied the idea of backwards compatibility to minimize the number of rulesets used.

Index Number of Generation	Bit no.									
		8	7	6	5	4	3	2	1	0	Dec.
	7	0	0					1			4
	6			1			0	1			68
	5	1	0					0			256
	4			1	0					1	65
	3	0	0					1			4
	2			1				1	0		68
	1	1	0					0			256
	0			1	0					1	65

Figure 22: Turning Rule Comparisons

Index Number of Generation	Bit no.									
		8	7	6	5	4	3	2	1	0	Dec.
	7						0				0
	6		0				0				0
	5									1	1
	4					1			0		16
	3						0				0
	2		1				1				136
	1	1									256
	0					0			1		2

Figure 23: Crossing Rule Comparisons

In Figure 22 and Figure 23, we compare the rulesets of each generation to see which ones we can combine without running into conflicts. We can distinguish what rulesets are compatible with each other by looking at the columns. If you stack two rows on top of each other and no columns contain both a 0 and 1, those two generations have combinable rulesets. We then compiled all the relations between each generation's ruleset into an undirected graph for easier viewing. See Figure 24 for the graph in question.

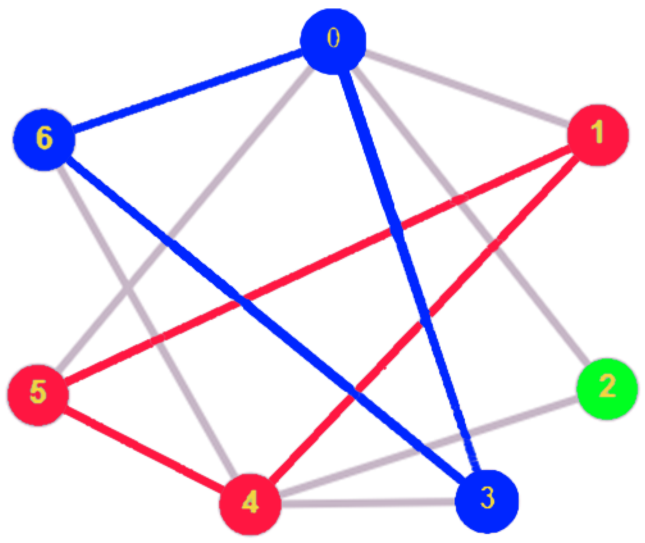


Figure 24: Undirected graph containing all compatible rulesets. Note that the ruleset for the generation at index 7 is not pictured because it is identical to the ruleset for the generation at index 3.

To find the smallest number of rulesets that could represent all the generations, we needed to search for the largest complete subgraphs of the graph, or cliques. [8] We first found the clique {0, 3&7, 6} by searching for the largest cliques we could find. After finding that clique, we looked for the next largest clique that did not contain any elements we had already sorted into the first clique we found. {1, 4, 5} was the next clique we found, and the last clique was a one element clique of {2}. Figure 24 shows each clique we found.

In Figure 23, we defined two rulesets to be compatible if no bit was 0 for one ruleset and 1 for the other. Consider that rules may contain irrelevant bits (-) where the value of the bit does not affect the rule because the landscape described by the bit does not appear in the braid that the rule governs. This leaves us with the possible combinations (- , -), (- , 0), (- , 1), (0, 0), (1, 1). Because it does not matter what value the irrelevant bits hold, they do not influence the combination of rulesets. For example, when combining rules where the first rule has an irrelevant bit and the second rule has either 0 or 1 in the same bit, the resulting bit will take the value of the second rule.

Bit no. Pairs	(- , -)	(-, 0)	(-, 1)	(0, 0)	(1,1)
Resulting Combined Bit no.	0	0	1	0	1

Figure 25: Each possible case of the bits of combinable rulesets.

Looking at the table in Figure 25, we decided to treat the irrelevant bits as 0's and apply a bitwise OR operation to get the resulting combined rulesets out of our 3 ruleset cliques. The results from the bitwise OR put the {0, 3&7, 6} clique under ruleset (69, 2), the {1, 4, 5} clique under ruleset (321, 273), and the {2} clique under ruleset (68, 136). See the labels to the right side of Figure 21B to see how the ruleset grouping affects the SCA.

5. Processing 3 Implementation of the SCA model

Building off Hao Yang's Java implementation of the SCA model that used text-based command line input [5], our goal was to create a GUI-based implementation that had support for space-varying and time-varying rulesets. We chose to use Processing 3 as it supports many kinds of graphical displays and would be easier to work with than Java graphics. The program is available at https://github.com/Nirb8/SCA-Processing.

To input rulesets into the model, we adapted the turning rule and crossing rule tables shown in Figure 5 and 6 to be interactive. Clicking the "tabs" at the top of the tables specified which rule is being edited and clicking the output cells toggled the value of bit controlled by that cell. Textbox-based input of rule numbers was also supported by clicking on an already active tab and entering in the decimal notation of a rule. Figure 26 and Figure 27 show the rule table input GUI in action. The currently displayed ruleset in the tables may be loaded into the SCA at any time.

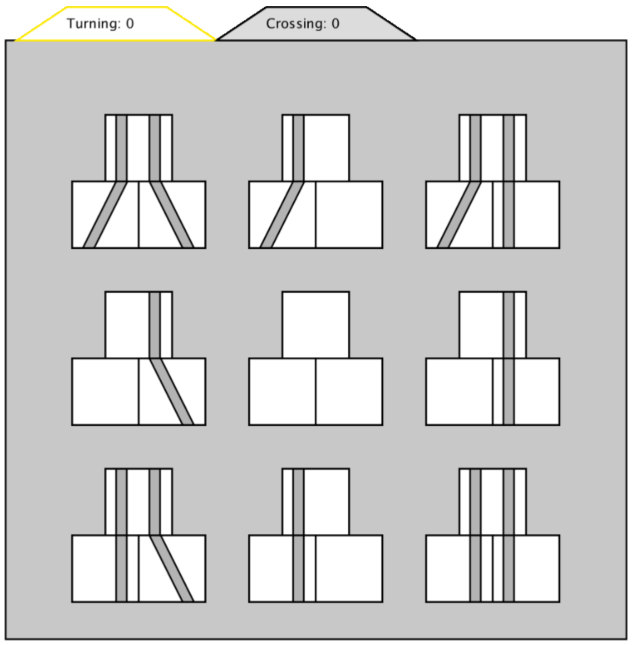


Figure 26: Active turning rule tab, the turning rule landscapes are being displayed in the grid.

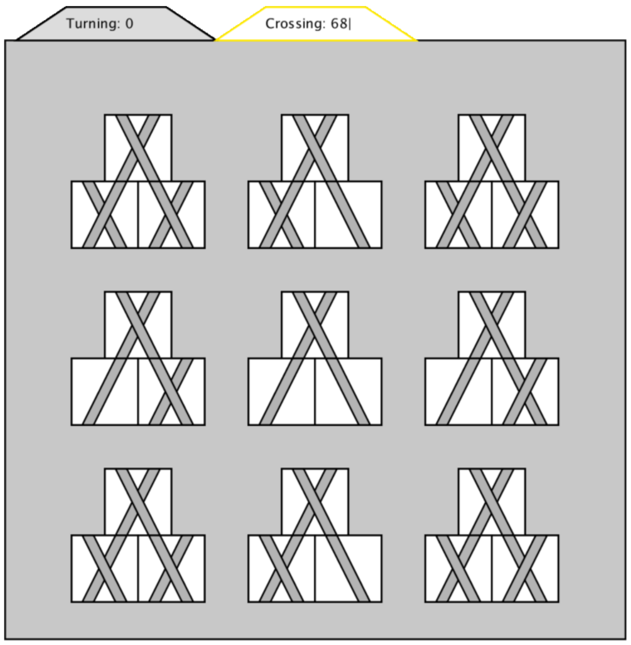


Figure 27: Editing the crossing rule by directly inputting a rule number.

Switching ruleset modes changed the information displayed in the labels of the model. The labels for single rulesets (Figure 28) show the generation number and the ruleset used by that generation to calculate the next generation.

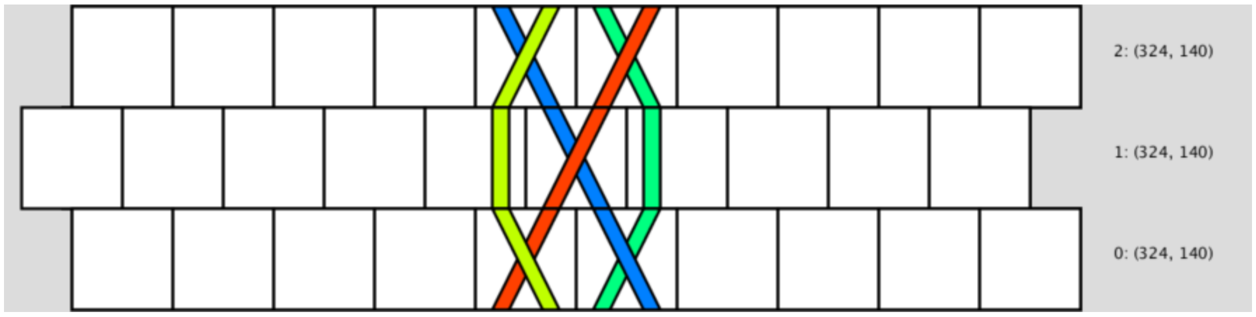


Figure 28: Single Ruleset mode, screenshot of the first 3 generations.

The labels for time-varying rulesets (Figure 29) show the index of the current generation in the list of rulesets stored in the SCA, along with the ruleset stored at that index. Once the SCA reaches the end of the list of rulesets it loops back around to the start of the list.

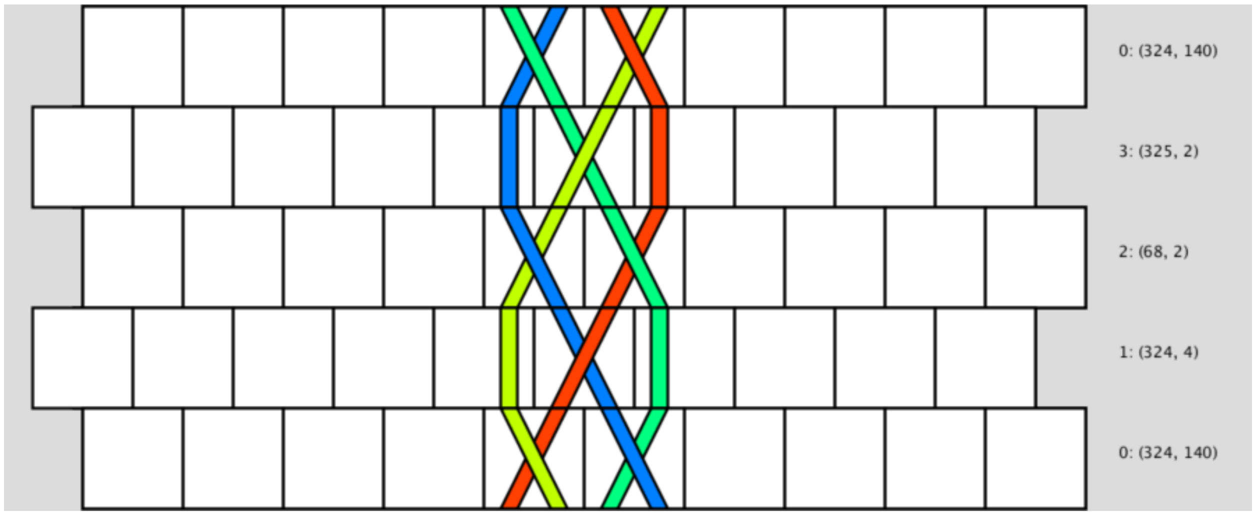


Figure 29: Time-varying ruleset mode, screenshot of the first 5 generations of a length 4 list of rulesets.

The labels for space-varying rulesets (Figure 30) differ from the previous two modes as there are only two labels, one on each side of the cell grid. There is also a bolded zipper line distinguishing where the grid is split on rulesets; the left side uses the ruleset displayed in the left label and the right side uses the ruleset displayed in the right label. The buttons below the grid can be toggled and the currently active side is where the rulesets will get loaded to.

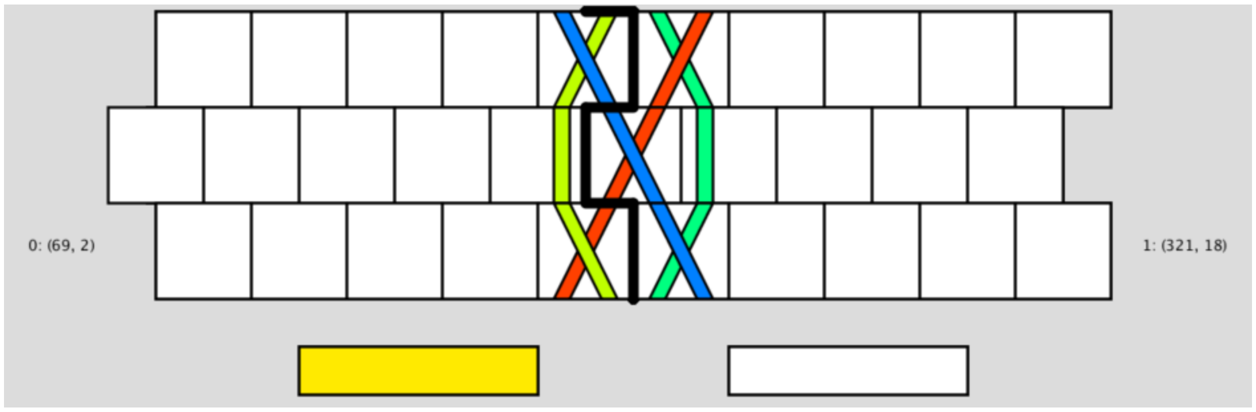


Figure 30: Space-varying mode, screenshot of the first 3 generations with an active left side selection button.

6. Conclusion

As a result of our research, we were able to expand upon the SCA model originally proposed by Joshua and Lana Holden and further researched by Hao Yang. We used time-varying rulesets to represent braids that behave differently at different points in time, and we used space-varying rulesets to represent braids that behave differently on either side of the braid. The concept of dividing the grid time-wise and space-wise to have a braid behave differently in different locations in time and space can also be applied to weaves. For example, the basket weave is constructed by making the same pattern twice, switching to a different pattern, doing the new pattern twice, and returning to the first pattern. [10] This pattern is not representable by a single ruleset SCA because the same patterns generate different patterns based on their location in time. Using a set of four time-varying rulesets would allow such a weave to be represented by the SCA model.

Although we created the Processing 3 SCA implementation specifically to represent braids no wider than 10 cells (20 strands), it can be easily extended to analyze weaves or braids with a width greater than 10 cells. Currently, there is support for vertical scrolling if a braid is too long to fit on screen completely. A similar function can be implemented to allow for horizontal scrolling when analyzing weaves. Additionally, much of the code is written in such a way that the whole model can be scaled based on the default size of the cells. This feature may be useful for those who desire to make a “zoom” feature to see the bigger picture of a braid or weave with a particularly large repeat length. Zooming in on certain parts of a braid or weave to view in greater detail might also be of interest. The Processing SVG library available at https://processing.org/reference/libraries/svg/index.html also may be of use when dealing with braids and weaves too large to fit on a single screen. Exporting braids and weaves as .svg files allows for easy scaling with external editors, and the patterns exhibited by the automata can be applied to things such as laser etching designs and 3D printing.

Perhaps the most interesting development future researchers can work on would be using time-varying and space-varying rulesets simultaneously to represent more complex braids that have conflicts in both space and time. Our work with braids was also only with braids with 5 or fewer strands, so it may be the case that there exists a braid with 6 or greater number of strands that needs both time-varying and space-varying rulesets to be represented.

References

[1] Weisstein, Eric W. "Braid." (2020). From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Braid.html

[2] Toffoli, T., Margolus, N. H. “Invertible Cellular Automata: A Review.” (1990). Physica D: Nonlinear Phenomena. 45(1-3), 229-253.

[3] Weisstein, Eric W. "Cellular Automaton." (2020). From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/CellularAutomaton.html

[4] Holden, J. & Holden, L. “Modeling Braids, Cables, and Weaves with Stranded Cellular Automata.” (2016). Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Culture, 127-134. Tessellations Publishing.

[5] Yang, Hao, "Stranded Cellular Automaton and Weaving Products" (2018). Mathematical Sciences Technical Reports (MSTR). 168. https://scholar.rose-hulman.edu/math_mstr/168

[6] Three Strand Braid. (2019). Retrieved July 30, 2020, from https://www.animatedknots.com/three-strand-braid-knot

[7] Four Strand Square Sinnet. (2019). Retrieved August 01, 2020, from https://www.animatedknots.com/four-strand-square-sinnet-knot

[8] Weisstein, Eric W. "Clique." (2020). From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Clique.html

[9] Textile Resources - Textile Encyclopedia. (2019). Retrieved August 04, 2020, from https://www.cottoninc.com/quality-products/textile-resources/textile-encyclopedia/