Emergence in Network Automata

Richard Foard

28 June 2019

Contact:
Richard Foard
email:
richard.foard@gtri.gatech.edu
phone: 404-281-3487

A simple, rule-based graph automaton was defined and simulated. Each of its 722 trillion possible rules specifies a set of local topology and node state changes to be applied iteratively, starting with a random initial graph. Simulation runs were performed using many different rules, each run terminating when its graph collapsed or cycled. Evolving and terminal graph states were analyzed macroscopically, using aggregate statistics, and microscopically, by inspecting graph structures. Results were compared with those from simulations of a machine that iterated by applying random, rather than rule-based, changes. Simple neural networks were unable to predict terminal graph characteristics given the rule that yielded the graph. Using a genetic search procedure to search the rule-space for rules producing target terminal characteristics yielded results dramatically better than those from random searches.

Introduction

Since Alan Turing conceived his universal machine in 1936, simple abstract automata have drawn research attention. Interest broadened beyond the academic realm in 1970, when Conway published his *Game of Life* simulations[1] that highlighted the ability of uncomplicated cellular automata to behave in complex ways. Researchers in the nascent field of complexity theory began studying similar phenomena, such as the sandpile avalanches first explored by Per Bak[2].

[1] Games, M., 1970. The fantastic combinations of John Conway's new solitaire game "life" by Martin Gardner. Scientific American, 223, pp.120-123.

[2] Bak, P., 2013. How nature works: the science of self-organized criticality. Springer Science & Business Media.

In *A New Kind of Science*[3], Stephen Wolfram systematically analyzed a variety of cellular automata types. He found particular inspiration in the behavior of a one-dimensional machine running "Rule 110."

[3] Wolfram, S., 2002. A new kind of science (Vol. 5, p. 130). Champaign, IL: Wolfram media.

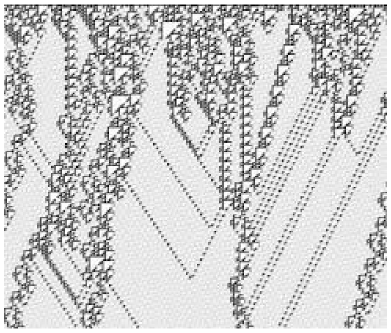
Figure 1: Excerpt from the state sequence produced by Wolfram's *Rule 110* one-dimensional cellular automaton

Wolfram and others have suggested that some natural processes that were previously thought to evolve by natural selection are instead manifestations of things that nature found "easy" to accomplish using the same fundamental principles that underlie simple automata.

In this work, we analyze simple, rule-based graph automata using simulations of abstract machines. Our machines operate on the same principles as cellular automata but use a graph, rather than a "tape" or grid, as a substrate. Where cellular machines define cell adjacency spatially, our machines use graph topology. We discuss the machines' behavior under varied rules and explore a genetic search algorithm for finding rules that yield terminal graphs with specific characteristics.

Methods

Two similar rule-based automata, **Machine C** and **Machine CM**, were designed and implemented in simulation. The simulators were run repeatedly, using rules selected at random from a large universe of possibilities. Measurements were recorded as the graphs evolved through each run. The resulting database accumulated data on many thousands of runs. The body of data was analyzed macroscopically, using aggregate statistics, and microscopically, by inspecting statistical and graphic snapshots from individual runs.

Each simulation run begins with a selected rule and a randomly generated graph. It proceeds by iteratively modifying the graph's edges and node states according to the rule. Each rule is effectively a simple program that specifies, based on the state of each node and its neighbors, how local changes should be applied during an iteration.

Graph Composition

Nodes may be in one of two states: *black* or *white*. Graph topology is restricted in order to simplify rule design. For the **CM** machine:

- Edges are directed.
- Nodes have out-degree of exactly two.
- "Self-linking" edges are permitted.

The **C** machine's graphs are more narrowly defined by adding the restriction:

- Multiple, like-directed edges between two nodes are prohibited.

C and **CM** machines operate using the same rule structures. A single rule is chosen for each run. Each execution begins on a graph that has been randomly generated to conform to its machine's topological restrictions. Both machines maintain conformance throughout the run. In applying some rules, **C** creates prohibited structures. When this occurs, a "pruning" process selectively removes nodes and edges to restore conformance before proceeding with the next iteration. The **CM** machine does not need this process because it is structurally incapable of altering its graph in a way thati violates its more relaxed structure restrictions. **CM** is otherwise identical to **C**.

An initial, random graph of N nodes is constructed by assigning each node a *black* or *white* state with equal likelihood. Two outbound edges are attached to each node, with the destination of each selected at random from all nodes. For **C** runs, the generator observes the constraint that the edges cannot share the same destination node. In all initial graphs, nodes have out-degree 2, but in-degree can vary from 0 to N. Initial graphs are a subset of the class of Erdos-Renyi random graphs[4] *G(n, p)* where *n* is the number of nodes and *p* is the probabil-

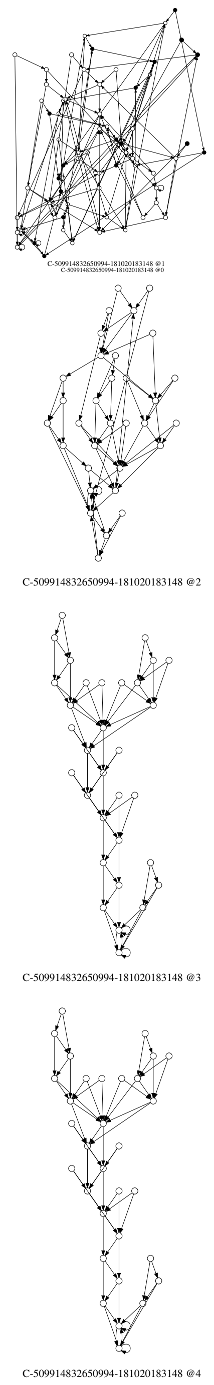
C-509914832650994-181020183148 @1
C-509914832650994-181020183148 @0
C-509914832650994-181020183148 @2
C-509914832650994-181020183148 @3
C-509914832650994-181020183148 @4

Figure 2: Four iterations of machine **C**, rule 509914832650994, beginning with a randomly generated 32-node graph and terminating in a static configuration.

[4] Erdos, P. and Renyi, A., 1959. On random graphs I. Publ. Math.

ity that two nodes are connected by an edge. Because all graphs have fixed out-degree 2, the generation process creates random variation only in the nodes' in-degrees. As a result, each initial graph is a member of *G(n, p)* where p is a function of graph size n (approximately $4/(n-1)$).

Rule Structure and Application

At the start of each simulation run, a rule is selected and an initial graph is generated. The simulation proceeds in a series of iterations. During each, the rule is applied at each node, yielding a draft version of a new graph. For machine **CM**, the draft immediately becomes the input for the next iteration. In **C**, the draft may require pruning before becoming input to the next iteration.

During an iteration, each node in the current graph is examined in turn. The combined *black*/*white* states of the node and its two neighbors[5] are used to select one of the rule's eight constituent parts; the selected part, in turn, controls the changes that are made to the node's state and its out-edges in the developing draft copy. Rules encode change instructions as follows:

[5] *Black* is interpreted as zero, *white* as one. In all cases, "neighbor" is used to indicate a node at the destination end of one of a node's out-edges.

A rule comprises eight parts, each corresponding to one of the possible compound, or "triad" states of a node and its neighbors. Each rule-part specifies replacement values for the node's state and the destinations of its out-edges (it is convenient to refer to a node's two out-edges as "left" and "right"). The replacement node state is either *black* (B) or *white* (W). The replacement edge destinations (Figure 3) are each one of:

- L - the node's current left-edge destination
- R - the node's current right-edge destination
- LL - the current destination of its left neighbor's left-edge
- LR - the current destination of its left neighbor's right-edge
- RL - the current destination of its right neighbor's left-edge
- RR - the current destination of its right neighbor's right-edge

A rule might be represented symbolically as a set of triples (*<new left-edge destination>*, *<new right-edge destination>*, *<new node state>*), for example:

$$\underbrace{(L, L, B)}_{0}\underbrace{(R, RR, W)}_{1}\underbrace{(L, R, W)}_{2}\underbrace{(LL, R, B)}_{3}\underbrace{(L, LL, B)}_{4}\underbrace{(RL, RR, W)}_{5}\underbrace{(RR, RR, B)}_{6}\underbrace{(R, L, B)}_{7}$$

in which each triple is a rule-part to be applied at nodes having a triad state equal to the part's index position in the rule string. For coding purposes, rule numbers are carried as mixed-radix integers between 0 and $(6 \times 6 \times 2)^8 = 722,204,136,308,736$.

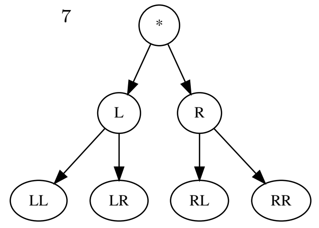

Figure 3: The six possible new destinations for node *****'s out-edges

After all nodes in the original graph have been processed, the re-

Table 1: Machine Characteristics

Machine	New Edge Selection	Multi-edges	Pruning Performed
C	*Rule-based*	*No*	*Yes*
R	*Locally Random*	*No*	*Yes*
CM	*Rule-based*	*Yes*	*No*
RM	*Locally Random*	*Yes*	*No*

sulting draft copy is pruned if necessary. Applicable only for machine **C**, the pruning step removes any like-directed edges (multi-edges) that have been created between pairs of nodes in the draft copy; the process, described in more detail below, selectively removes nodes and redirects edges, cascading as required, until no prohibited structures remain. If it is not empty, the pruned graph becomes the starting point for the next iteration; if it is empty, it is said to have collapsed. A simulation run ends when (1) the graph collapses, (2) a state cycle is detected, or (3) a maximum number of iterations is reached.

*The **C*** Machines' Random Counterparts, **R** and **RM***

To help gauge the extent to which machines **C** and **CM** introduce order as they transform random starting graphs, simulators for counterpart machines **R** and **RM** were constructed. Like the rule-based **C** and **CM** (**C***) machines, these change the graph iteratively, but make their changes *randomly.* As with the **C*** machines, though, the scope of edge destination changes for each node is limited to nodes reachable in its ”two-hop” neighborhood. (The random machines need not alter nodes' states because node states have no effect on the **R*** machines' actions). The **R** machine applies the same pruning process that **C** uses. As with **CM**, no pruning step is required in **RM** simulations because no prohibited structures can be generated by the ***M** machines.

Graph Pruning

Changes to out-edge destinations, whether made randomly by machine **R**, or based on **C**'s rules, can introduce multi-edges. These prohibited structures arise as the machine's iteration logic creates a first-pass, ”draft” copy of the transformed graph. the draft copy is pruned to restore conformance with structure restrictions before the following iteration begins. The diagram in Figure 4 shows the two kinds of prohibited structures that can occur and illustrates the restorative changes made in the pruning step.

Both types of changes eliminate the node at which the offending

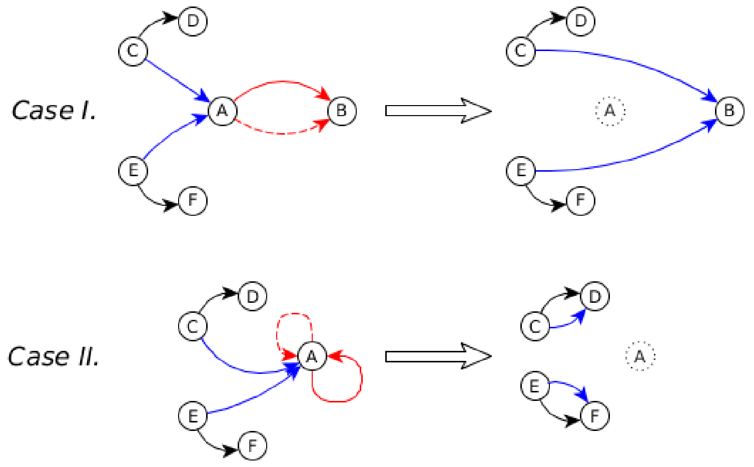

Figure 4: **Pruning.** Resolution methods for two variations of a (red) prohibited structure. In both cases, edges originating at node A are eliminated; edges in blue are redirected. Case II creates two case I-type structures that require further resolution.

edge pair originates (labeled ”A” in the illustrations), along with the edge pair. In case I, all edges inbound to the eliminated node are redirected to the destination (shown as node ”B”) of its to-be-eliminated edge pair. Redirected edges are shown in blue.

Lacking such a natural new destination for the inbound edges in case II, the pruning process instead moves the problem ”upstream”—any edge inbound to the eliminated node is redirected to the same destination node as that of its origin node’s other out-edge. This, of course, immediately creates case I structure violations for all such origin nodes; these violations must also be resolved before the next iteration can begin. Case II resolutions *always* create additional structure violations; case I resolutions may or may not. In either case, cascades can result. These cascades, and the resulting reductions in graph size, are common in both **C** and **R** simulations.

The Automaton Simulator(Explorer) and Data Flow

The automaton simulator was constructed in *C++*. It can be used to simulate the rule-based automaton with or without permitting multi-edges in the evolving graphs. It can also simulate operation in a random mode, in which node state and topology changes are still applied within each node’s two-hop neighborhood but are determined at random rather than by applying a governing rule. For each execution, command line arguments select the type of automaton, governing rule number, number of graph nodes, depth of cycle-checking, random number seed, and maximum number of iterations.

Each simulator run produces JSON-encoded files containing run parameters and intermediate and final outcome statistics such as node count, graph diameter, average clustering coefficient, etc. These files are imported into a *Postgres* database that accumulates data as simulation batches are completed. *SQL* and *R* scripts are then used to query and analyze the assembled results.

Outcome Prediction with Simple Machine Learning

A simple learning experiment was conducted to gauge whether, given a specified rule, a trained neural network could predict the values of the machine's terminal graph. Training data consisted of (*encoded_rule*, *statistic_value*) pairs in which *statistic_value* was one of:

- residual number of nodes
- average clustering coefficient
- number of iterations to reach terminal state
- diameter
- outcome (cycling vs. collapse)

Two types of encoding were used for the *encoded_rule* value:

- Rule-parts: A vector of eight integers in the range [0, 72 ($6 \times 6 \times 2$)], each representing the local transformation to be applied depending on the compound state of a node and its two neighbors, and,
- Rule-map: A vector of 112 *0/1* values encoding the rule in a modified one-hot scheme. Where an edge destination could take values *L, R, LL, LR, RL,* or *RR*, for example, the destination *RL* would be redundantly encoded in the bit sequence *0,0,0,0,1,0*, indicating *L=0, R=0, LL=0, L=0, RL=1, RR=0.*

The Rule-space Searcher (Searcher)

The rule-space for machine **C** is vast and incoherent. It comprises 7.2×10^{14} rules; numerically similar rules produce very dissimilar behaviors and yield widely varying terminal graph states. We developed a genetic search algorithm[6] for finding rules that produce terminal graph states with specific characteristics such as high clustering coefficients, or a large number of (separate) connected components. A "fitness" function quantifies the relative extent to which a rule produces a terminal graph with the desired characteristics.

[6] The algorithm design benefitted from the experience of Adonis Bovell, GTRI CIPHER laboratory.

The search proceeds by first creating a pool of randomly selected rules, then improving aggregate fitness by iteratively replacing lower-fitness rules with new ones. The replacement rules are synthesized by choosing two existing rules with relatively high fitness and "crossing" them, possibly mutating one of them before crossing. The process iterates, terminating either when the pool reaches a target fitness or when an iteration limit is reached. Successful searches terminate with a pool having significantly higher fitness than that of randomly generated pools.

A rule is mutated by replacing one or more of its components with a new, randomly chosen value. Two rules are crossed by randomly choosing one or more of their components and exchanging them. The likelihood of a mutation occurring is controlled by the *probMutation* parameter.

The search proceeds in generations. Each successive generation creates a new pool of rules from the previous generation's by repeatedly choosing pairs of rules from the previous pool, mutating and crossing them, then adding the chosen rules and their new offspring to the new pool until it is full. The process repeats until *maxGenerations* have been generated.

Rules with relatively high fitness are selected from a pool using a list of rules in order of decreasing fitness. Each entry is accompanied by cumulative fitness, summed from the beginning of the list. A choice is made by choosing a target fitness value randomly

Results

Our first task was to understand the characteristic behaviors of each machine, comparing the machines and their random counterparts. In this section, a survey of qualitative and quantitative simulation results is presented, followed by an examination of more specific aspects of behavior.

Summary of End States—Cycles and Collapses

Each simulation run begins with a finite graph, and each iteration either maintains the number of nodes or, in the cases of **C** and **R**, reduces it by pruning. It is consequently certain that the machines always terminate, either in a state cycle or in a graph collapse; this holds true for all types: **C**, **CM**, and their random counterparts **R** and **RM**. The possibility that a graph will traverse an astronomical number of states[7] before revisiting a previous one, however, limits the simulator's ability to detect cycles. If the simulator reaches a maximum number of iterations before its machine terminates, the simulation outcome is recorded as "undetermined." A summary of outcomes for all runs is given in Table 2.

[7] The number of possible states for an N-node graph with out-degree restricted to 2 is:

$$2^N \cdot \binom{N}{2}^N$$

The reason that nearly all **CM** runs end in state cycles is intuitive—because **CM** allows multi-edges, it tends to generate structures like

Table 2: Simulation Outcomes, All Runs

Machine	Cycling	Collapsed	Undet.
C	40.77%	59.22%	0.01%
R	0%	100%	0%
CM	99.07%	—	0.03%
RM	100.00%	—	0%

those in Figure 5 regardless of which rule is governing its execution. As these structures enter the graph, they shrink the number of possible new out-edge destinations for their origin nodes, progressively decreasing the number of possible new states in the graph overall.

Less inutuitive is the finding that all **RM** runs end in cycles—it would seem that this machine's continuously random edge reassignments would make state cycles impossible. In fact, **RM**'s graphs invariably devolve to configurations in which every node's six possible "new" out-edge destinations are, in fact, all the same node—the previous destination node of both its out-edges—resulting in no change to the edges and causing a tight, one-state cycle. An example of such a graph appears in Figure 6.

The prohibition of multi-edges in machines **C** and **R** necessitates the removal of prohibited structures when they arise. The number of nodes in the graph can consequently decrease during execution, often to the point of complete collapse. Machine **R** is far more likely to produce graph collapses than **C** is (99.9% vs. 59.2% of runs). The reason is similar to the reason that ***M** machine runs cycle: **R** and **RM** share the same random edge rerouting logic, and produce multi-edges with equal likelihood. These are allowed to remain in **RM**'s graph, but must be pruned out of **R**'s graph before it can continue with execution, reducing its node count and making collapse more likely. The number of multi-edges **C** produces, on the other hand, is determined by the rule under which it runs, and consequently varies widely as the machine is run with different rules.

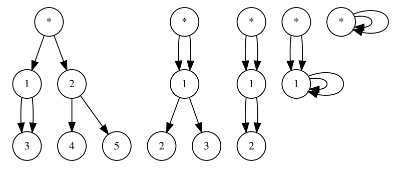

Figure 5: Structures with multi-edges reduce the number of possible new edge destinations for the origin ("*") node during each machine iteration.

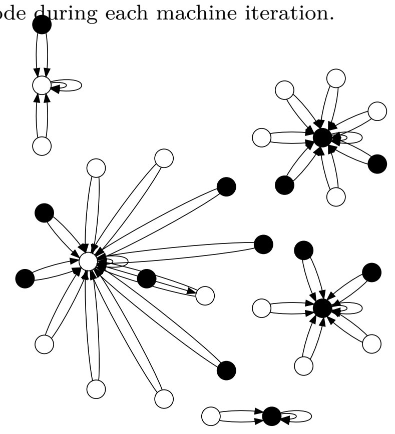

Figure 6: A representative terminal graph from an **RM** simulation. The configuration is a one-state cycle even under locally random edge reassignment.

Table 3: Comparison of Graph Measures, **C** vs. **R**

Statistic	Comparison
Average Run Length & Terminal Graph Size	Between **C** and **R** runs ending in collapse, **C** runs average nearly twice the number of iterations (Figure 8) and end with much higher pre-collapse node counts. They run longer and their node counts ”fall farther” when they collapse. **C** runs that end in state cycles terminate with higher average node counts than those for collapsing runs, despite averaging more than twice the run length. Machine **C** runs lengths correlate with terminal graphs sizes except in the case of the largest sizes, which tend to be produced in somewhat shorter runs. run lengths decrease somewhat
Average Clustering Coefficient	Terminal clustering coefficients for machine **C** are smaller than those for **R** in the large majority of cases (Figure 7).

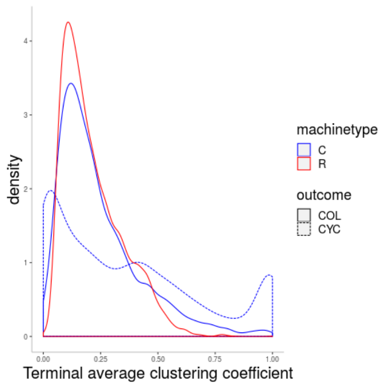

Figure 7: Distribution of terminal average clustering coefficients for machine **C** (cycling), **C** (collapsing), and **R** (collapsing).

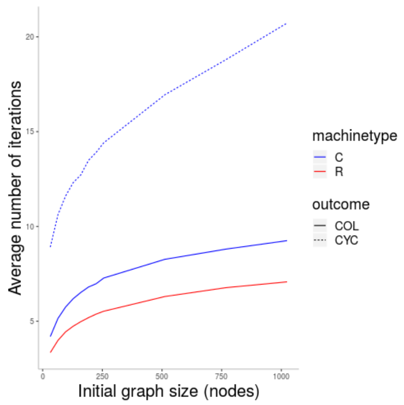

Figure 8: Average number of iterations for machine **C** (cycling), **C** (collapsing), and **R** (collapsing).

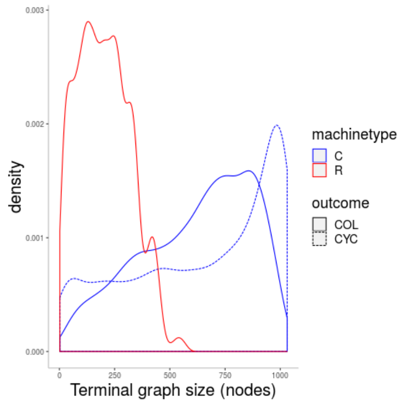

Figure 9: Distribution of terminal graph sizes for machine **C** (cycling), **C** (collapsing), and **R** (collapsing) with initial graph size 1031.

In-Degree Entropy

Machine **C**, on average, generates graphs with a larger maximum in-degree than in the random case of machine **R**, and also produces a larger number of distinct in-degrees.

Shannon's entropy[8] computed on summary in-degree statistics and normalized,[9] can be regarded a measure of a graph's "randomness." As would be expected, entropy is consistently large in the randomly generated initial graphs. Between the **R** and **C** machines, entropy in the terminal graphs for **C** is lower than that for the randomly operating R (Figure 10).

The drop in average entropy between initial graphs and the **R** machine's terminal graphs seems surprising on its face, but is accounted for by the restriction that **R**'s topological changes, like **C**'s, may only redirect a node's out-edge within its two-hop neighborhood. The effect is to "localize" the randomness, abruptly increasing apparent order in the random graph as soon as the first iteration is finished.

[8] Shannon, C.E., 1948. A mathematical theory of communication. Bell system technical journal, 27(3), pp.379-423.

[9] Normalized entropy calculation:

p_k = fraction of nodes with degree k
K = number of distinct in-degrees

$$entropy = \frac{\sum_{k=1}^{K} p_k log_2 p_k}{log_2 K}$$

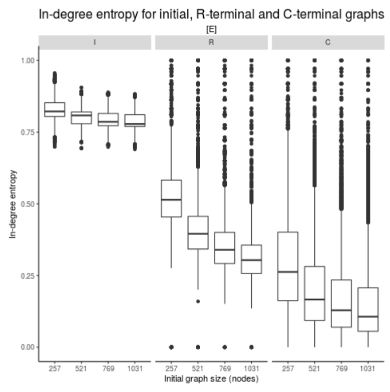

Figure 10: In-degree entropy is largest in initial random graphs, smaller for **R**'s terminal graphs, and smallest for **C**'s terminal graphs.

Degree Distribution

Because of the structure constraint that all nodes have out-degree of two, the distribution of out-degrees is always flat at that value. In-degrees, on the other hand, may vary unconstrained as successive iterations transform the graph. The terminal graphs produced by both the **C** and **R** machines almost invariably had in-degree distributions characterized by a large plurality of nodes with degree 0 or 1 and a handful of nodes with varied, very large degree. The blue line in Figure 11 shows the typical case; for comparison, the green line plots the distribution for a very rare case exhibiting the power-law distribution that appears for most real-world networks.

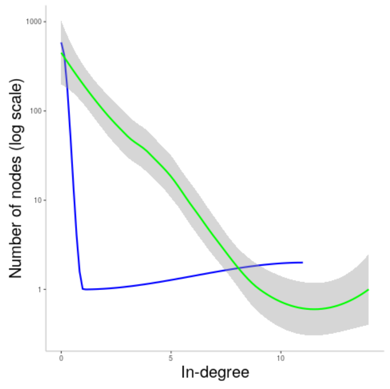

Figure 11: In-degree distribution excerpts for the typical case (blue) and a rare outcome (green) in which a power-law distribution was generated

Predicting Outcomes

Using the *KERAS* framework, a simple neural network with an input layer, a single hidden layer, and an output layer was constructed; weights were updated using stochastic gradient descent. For both *rule-part* and *rule-map* input encodings, with networks trained over half the rules for which data was available, no outcome statistic was predicted with accuracy better than that of random choice. No further attempt was made at predicting outcomes.

Searching the Rule-space

The genetic search algorithm described in *Methods* was applied to the task of finding rules that generated terminal graphs with characteristic statistics falling within specified ranges. For most characteristics, the

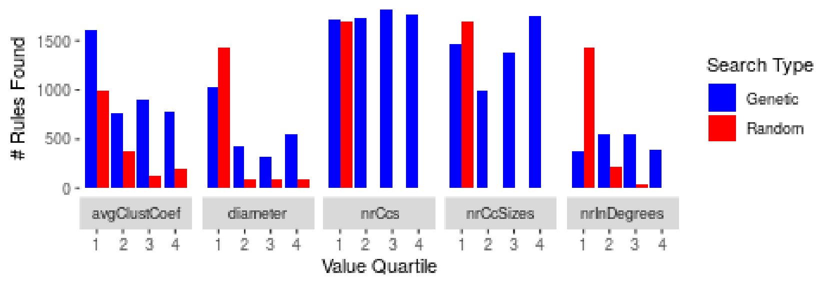

Figure 12: **Search performance.** Number of rules found with terminal graph statistics in the target quartile of the range after examining 1700 rules chosen using a random (red) or genetic (blue) search strategy.

genetic search algorithm perfomed dramatically better than random selection. Results are summarized in Figure 12.

Discussion and Conclusions

40% of terminal graphs produced by **Machine C** using randomly chosen rules did not collapse to empty. As a group, they had widely varying configurations and degree distributions. Only a handful, though, showed the power-law, ”scale-free” distribution that is a signature of most real-world networks. It is possible that, under the correct transformation, scale-free structures would be found to be encoded at a coarser level of detail than that of the final graphs – this is a possible area for further investigation.

A pruning process was applied to **Machine C**’s evolving graphs in order to maintain structure constraints. Pruning typically had the effect of reducing node counts in an irregularly cascading fashion reminiscent of Per Bak’s sandpile avalanches. Further research specifically focused on this phenomenon could illuminate interesting dynamics.

The failure to predict outcomes using simple neural nets suggests that **Machine C** behavior is not reducible to a simpler means of generation.

The effectiveness of genetic search in finding rules producing specific values in terminal graph statistics, on the other hand, may indicate the presence of regularities in the seemingly incoherent rule-space. It may be useful to explore the possibility more thoroughly, to see if it is feasible to discover rules that give rise to useful functional characteristics in evolving or terminal graph topologies.

The correspondence of graph structures and their dynamics with mathematical constructs was not investigated in this work, and could conceivably point out parallels that open new lines of inquiry.

In this study, practical considerations dictated the selection of a single type of automaton and accompanying graph topology restrictions. Examining other, similar automata types could give insight into strategies for seeking other abstract machines with behaviors of theoretical or practical interest.