I. Introduction

Long have scientists searched for models that simplify complex problems with many individual components. Oftentimes the work of physicists and chemists involves approximating many-bodied multi-agent systems with a macro-scale model that represents the overall behavior (e.g. fluid dynamics, which makes sense of macro-scale results about the interactions of countless particles). However, sometimes interest lies in the complexity itself. The subject of natural computing, a field of research dedicated to creating computational models that replicate natural systems, is usually studied through models that contain many simple objects called agents. One such computational model is the Cellular Automaton, among some of the oldest models of natural computing [1].

In this review I will examine the cellular automaton model in detail, discussing how it is implemented and constructed. With this model I can explore the evolutionary methods used to create cellular automata capable of self-organization, a special property of many real-world phenomena that will serve as the underlying subject of this research. Additionally, I will explore some of the optimization methods used to make the model more efficient and where those optimizations could be improved to create evolutionary cellular automata.

II. Self-Organizing Systems & Emergence

Before too much detail about cellular automata, it is important to establish some context around the systems in nature they tend to best represent. Nearly all phenomena in the natural world are described by physical interactions of numerous small objects (I will call ‘agents’) where the macroscopic behavior exhibited by the system may not correlate precisely with expectations generated purely by microscopic rules. However, many times

there are large-scale interactions that underpin the overall interaction (e.g. a magnetic field around a wire, an external guiding force for the numerous agents in the molecular structure). There is a special class of interactions and behaviors, though, that are marked by some large-scale behavior that is generated purely by agent-based interaction with local information without external direction. These behaviors are called *self-organizing*, and it is a major theme of most natural, living and non-living systems around the world [10]. Self-organizing behavior is present in molecular formations, social structures, and more. This kind of phenomena is useful for modeling because it is resistant to scaling (making the model larger usually involves increasing the agent count) and often contains robust properties that make it resistant to sudden structural change.

According to Jin [11], self-organizing systems, particularly those that are multi-agent systems, "have been highlighted in many engineering fields, such as computer science, industrial engineering, and material science." Most of the research done has focused on designing self-organizing systems that fulfill a certain set of goals. However, some researchers, like Jin, are concerned with the way that the local interactions correlate with the requirements of the overall system. For many reasons discussed in *Section V: Evolving Cellular Automata*, it is critical to have the ability to recognize self-organizing systems from an automated standpoint and being able to quantify the effect is important both in experimental validation as well as when developing these systems using algorithms.

Researchers have searched for self-organizing behavior by quantifying 'pattern-ness' since many self-organizing systems produce patterns which can be analyzed using conventional means such as image and texture analysis. Suzudo [7] automates the

identification of pattern-forming systems by finding a way to measure the local entropy of the agents and then plotting its distribution as a contour map. Silva [12] showed how image processing can be applied to multi-agent systems to classify their behavior, some of which possess important qualities desired by artificial life and other simulated self-organizing systems. Critically, this pattern of research demonstrates that it is possible to quantify this effect and sort it into categories that allow for proper experimental reporting and identification by computer algorithms.

A particularly interesting and difficult-to-quantify phenomenon in self-organizing systems is *emergence*. Emergence, according to Parunak [6], is a "system-level behavior that is not explicitly specified in the individual components." For example, consider an ant colony, where the overall structure comes into spontaneous order from the interactions between numerous agents; no individual ant is explicitly programmed with the goal to create the final structure, though nonetheless, the structure occurs. We would say that an ant colony is an *emergent phenomenon* of the multi-agent system. Emergence is an interdisciplinary research subject that has a long history, "forming the central focus of statistical mechanics", and it has applications to other important concepts such as entropy, differential equations, traffic analysis, and more. In addition, the way that we understand micro-behavior from macro-behavior is significantly different from the way we understand it the other way around:

To discover macroscopic behavior from microscopic interactions, we largely perform the same techniques that physicists use to model the world. Here, mathematical equations, assumptions, and simplifications rule the design process. On the other hand, to discover microscopic behavior from macroscopic interactions we tend to use some form of

process which involves trial and error and large sets of data. One quite well-known method which uses simulated evolution is genetic programming (or more broadly, evolutionary computation), which is described in more detail in *Section IV: Genetic Algorithms*.

III.Cellular Automata

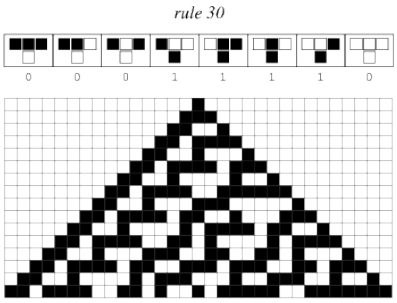

Figure 1: (credit: WolframAlpha) A time-space graph (time on the vertical axis) of a simple cellular automaton with transition rules displayed on the top (three cells on top indicate the current state, the resulting cell is below). Notice that a single cell transitions based only on its two immediate neighbors and its own state.

Starting with von Neumann, researchers are guided towards CA (cellular automata), one of the most popular multi-agent models, to study natural computing through a biological motivation, desiring to understand how biological systems self-organize. Cellular automata are represented by a grid, sometimes called a *world*, and there are agents in that world, called *cells*, which evolve and change state based on mathematical rules. Why, though, do CA simulate natural systems so well? Kari explains that "[they] possess several fundamental properties of the physical world":

1. They are massively parallel: calculations across the model are done uniformly and synchronously, as opposed to models which are like lists of instructions.
2. They are homogenous: the laws governing the agents' behaviors are uniform throughout the model.
3. They are local: interactions are not affected by large distances – cells are only affected by immediate neighbors and their own states.

Although traditional mathematical tools can model many behaviors in the sciences, in principle Ermentrout [4] explained that there is a "price paid" for this detail: the "basic physical principles" that are underlying the phenomena are obscured by the complex

systems of equations and laws. Additionally, simulating physical phenomena starting with the modeling equations can prove to be a "formidable" challenge for calculation. One way to simplify the complex tools often used for traditional modeling is to represent the system in discrete and simplified parallel instructions, provided by a CA.

Not only do cellular automata provide modeling capacity, but they present their own set of interesting mathematical problems that offer insight into other complex systems, and the solutions developed in these contexts are interdisciplinary, even on very simplified CA problems [3]. There are two main types of abstract problems from CA:

1. *Identifying rules:* starting with a known sequence of states achieved by the computation of a cellular automaton, identify the set of transition rules that produce these sequences of states.
2. *Synthesizing rules:* starting with an initial condition and a desired set of properties or end states, identify the set of rules that fit these constraints.

The large difference between these two types of problems is that in Type 2 we do not have detailed steps of the CA's evolution, only a desired goal, while in Type 1 we are attempted to reverse-engineer a set of rules from detailed accounts of each time step. Solving problems of the second kind transfers to a host of tasks in artificial intelligence, since many optimization tasks come from a desire to generate a set of actions an agent can take to achieve some final state. In *Section V: Evolving Cellular Automata*, I will present a specific case of this category of problem and describe a generalized and interesting method to performing this search for solutions that has implications in a variety of fields. It is important to note that solving this kind of problem precisely is generally very difficult and

has applications in nanoscale design and engineering, programmable gates, molecular computation, and transport networks.

To increase reliability for physical modeling, CA have also been endowed with additional rules like conservation laws. These have desirable properties when searching for CA that evolve self-organizing behaviors because they closely represent the subject's behavior [2], and they exclude CA which evolve homogeneity (completely empty or filled grids).

There are two main ways to assign an update instruction to a cellular automaton:

1. *Synchronous*: the results of the transitions are the same regardless of the order of the cell updates because the transition of the cells is a function of the previous cell states to the next.
2. *Asynchronous*: the results of the transitions are dependent on the order of the updates since a cell transition will be a function of the *current* neighboring cells and the next state.

Asynchronous CA (ACA) have some advantages over synchronous CA. Suzudo [7] explained that they chose an ACA for pattern-forming research because one can induce conservation laws due to assurance that cells are never duplicated across synchronous updates. Cenek [2] wrote that "evolving CA with asynchronous cell updates is an important topic for future research" since asynchronous updates make CA "more realistic as models of physical systems". However, while ACA have advantages in modeling, what does it mean to 'evolve' a CA and how can this be done in a computer?

IV. Genetic Algorithms

A Genetic Algorithm (GA) is a type of computer program that is designed in part by simulated evolution. Modeled after Darwinian evolution, as Cenek [2] explains, GAs have proved successful for solving a variety of difficult problems. In problems that may be extremely difficult for a programmer to understand analytically, GAs can be evolved instead. A genetic algorithm is designed as follows:

1. *Encoding*: represent your program and the calculations it performs as a chromosome, a packet of virtual 'genetic material' that can be used to reconstruct an instance of your program that, now, may be very poor at performing the target task.
2. *Selection:* assess the objective and create a way to test the performance of an instance of the program against this objective. This performance should be scored as a number that is called *fitness*.
3. *Mutation:* design a randomized way to mutate an instance of your program, performing slight modifications to its chromosome.
4. *Evolution:* start with a random initial population of chromosomes, mutate them and assess their fitness. Apply selective pressure by deleting the chromosomes that are outperformed on the *fitness* test, replicate these and apply this method iteratively.

At the end of the evolutionary process, you should obtain a collection of chromosomes that reconstruct versions of your program that perform the target task very well, sometimes better than you would have been able to write by hand. This method works well for a variety of tasks because it takes advantage of the power modern processors possess, essentially brute-forcing a solution to a variety of problems. If the model is designed well (*encoding, selection, mutation*, etc.), many optimization problems are solved quickly by GAs.

Genetic algorithms especially have promise in studying *emergence*. This should come as no surprise, as it is well known from Darwinian theory that natural selection has been the driving force behind the development of emergence in life. Genetic algorithms have been applied to search for the microscopic behavior of individual agents in the system that will generate specific large-scale behaviors [5].

However, exploring such a problem with genetic algorithms is not a trivial task. Performing the steps listed above, especially for studying self-organizing systems, is often complicated and difficult to tune. Additionally, Humann [8] describes "the nonlinear" and "unclear [...] link between the input parameters and the global measured behavior." The behavior itself may be hard to identify as well, since many times it is challenging to characterize and describe self-organizing behavior. Despite this, there are "many successful demonstrations of evolutionary optimization of complex systems in the literature". Methods and studies dedicated to characterizing, identifying, and quantifying self-organizing behavior is discussed in *Section II: Self-Organizing Systems & Emergence*.

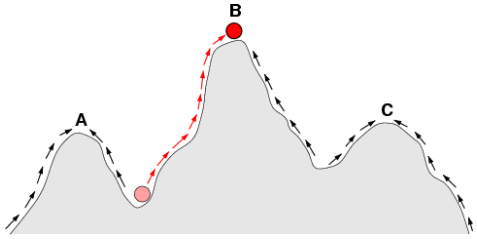

Figure 2: (Wilke, public domain): sketch of a fitness landscape, where the fitness of a chromosome variety is represented by the height of the graph, and the similarities between population individuals is correlated with closeness on the x-axis. A genetic algorithm proceeds towards optima, following the arrows on the graph. A multitude of individuals is spawned to avoid converging only on local maxima (A, C).

To deal with the complex interaction between parameters and results, it is useful to construct what is called the *fitness landscape*. During the evolutionary search, several variations of specific parameters are produced by mutation, and we can measure the fitness of each of these states and produce a graph that, in some cases, can resemble a natural landscape.

Producing such a landscape helps researchers make important decisions about the design of the genetic algorithm and often reveals characteristics of the problem being studied. For example, a problem with a relatively smooth fitness landscape often contains a set of small, incremental changes that the genetic algorithm can perform to optimize the task at hand. However, other problems may contain a corrugated, volatile fitness landscape that will require experimentation and analysis to develop and perfect the algorithm which processes it. More about fitness landscapes will be explored when applied to generating cellular automata with self-organizing behavior, and how we may overcome some of the inherent limitations of the evolutionary methods introduced here.

V. Evolving Cellular Automata

Not only have CA been investigated to study self-organizing systems, but CA have also demonstrated interesting and life-like behaviors. One primitive example self-replicating CA was investigated by Bidlo [3] to understand how information can be encoded in self-replicating structures called loops. A self-replicating structure called an envelope loop (right) was discovered by evolutionary methods performed by Bidlo. While it is possible to create self-replicating structures by hand using cellular automata, as Langton did to investigate artificial life, it is particularly interesting to use the wealth of computational power available now to generate these structures using artificial evolution. There are two major methods to achieve computational evolution of CA:

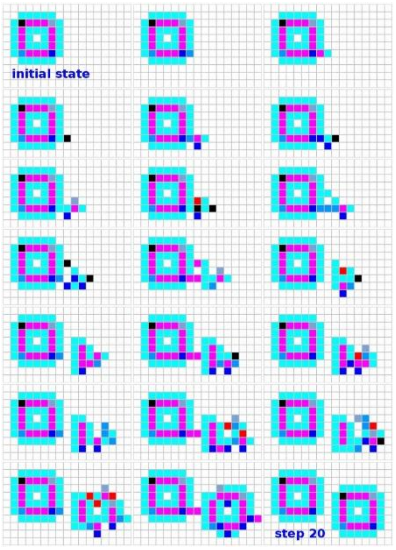

Figure 3: (Bidlo [3]) A primitive example of a self-replicating cellular automata inspired by Langton's Loops. The evolution of the produced individual is read by following the images left-to-right, then top-to-bottom. The individual is capable of self-replication, and this behavior was produced semi-spontaneously due to selective pressures from a genetic algorithm designed to search for this behavior.

1. Evolve the *transitions*/*rules*, as Sughimura [5] performed to generate cellular automata that develop a specific and interesting kind of chaotic behavior that is characteristic of self-organizing systems. In this method, we encode the transition rules of the CA as the genetic material of the program, and then we mutate these rules to study the results, applying some fitness measure to keep rules that give a desired behavior.
2. Evolve the *initial states*. In this method, we fix the rules and evolve the way the CA is initialized.

As discussed in *Section I: Self-Organizing Systems & Emergence*, there are a variety of methods used to find and search for the behaviors we may desire from an evolved cellular automaton. Ashlock [12] developed a robust and large set of species from a biological motive, selecting for cellular automata that possess apoptosis, a property of living cells where a population or an individual grows quickly, lives for some steady amount of time, and then suddenly undergoes a period of planned death. The time-space graphs of these cellular automata have a fixed height and often demonstrate chaotic behavior that can be compared across generations and species. The work of Suzudo [7] demonstrated that cellular automata can 'learn' pattern forming behavior by evolutionary synthesis by the application of a genetic algorithm, again by evolving the set of rules from a fixed initial state and by tuning the parameters of the genetic algorithm to search for entropy-balancing behavior. Some, like Sughimura [5] even discovered cellular automata with, what some researchers refer to as "order on the edge of chaos" (a term coined by mathematician Doyne Farmer), behavior that is found immediately between rules which produce seemingly random disordered patterns, and rules which produce overly ordered patterns such as oscillations.

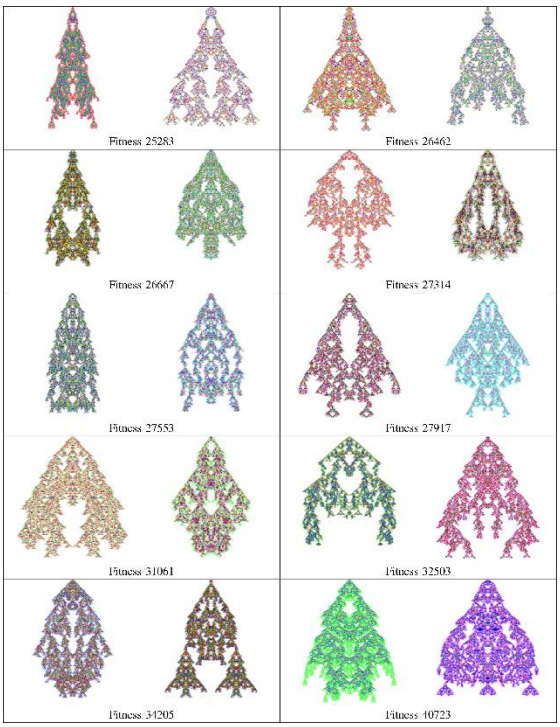

Figure 4: (Ashlock [12]) Time-space graphs of cellular automata produced by a genetic algorithm designed to search for apoptotic behavior. Viewing from top-to-bottom, observe the way the cellular automata grow for a period and then quickly disappear before the end of the graph.

VI. Conclusions & Research Question

Given this wealth of work on self-organizing systems, cellular automata, and the application of evolutionary methods to investigate the former using the latter, it is important to summarize the place to look going forward:

First: *how does searching the initial-state space compare to searching the rule space?* Much of the research presented performed comprehensive analysis of evolving rules in cellular automata to evolve complex behavior, but what happens when we start with a set of rules that we suspect support self-organizing behavior and then evolve the initial states? Is there a clear reason one may be more robust than the other? It should be investigated, on a comparative basis, how these methods compare. Additionally, how closely intertwined with the initial state is the evolution of the cellular automata? From the current research it is unclear what effects the choice of initial states might have on the generation of pattern forming cellular automata. Answering these questions could have important impacts on the way this research is conducted going forward.

References

[1] Kari, Jarkko. "Theory of Cellular Automata: A Survey." Theoretical Computer Science 334, no. 1 (2005): 3-33.

[2] M. Cenek and M. Mitchell, "Evolving cellular automata," in New York, NY: Springer New York, 2012, pp. 1043-1052.

[3] Bidlo, Michal. "On Routine Evolution of Complex Cellular Automata." IEEE Transactions on Evolutionary Computation 20, no. 5 (2016): 742-754.

[4] Ermentrout, G. Bard, and Leah Edelstein-Keshet. "Cellular automata approaches to biological modeling." Journal of theoretical Biology 160, no. 1 (1993): 97-133.

[5] N. Sughimura, R. Suzuki and T. Arita, "Non-uniform Cellular Automata based on Open-ended Rule Evolution," Artificial Life and Robotics, vol. 19, no. 2 (2014): 120-126.

[6] H. V. D. Parunak and S. A. Brueckner, "Software engineering for self-organizing systems," The Knowledge Engineering Review, vol. 30, no. 4 (2015): 419-434.

[7] T. Suzudo, "Spatial pattern formation in asynchronous cellular automata with mass conservation," Physica A: Statistical Mechanics and its Applications, vol. 343 (2004): 185-200.

[8] J. Humann, N. Khani and Y. Jin, "Evolutionary computational synthesis of self-organizing systems," Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 28, no. 3 (2014): 259-275.

[9] B. Abolhasanzadeh and S. Jalili, "Towards modeling and runtime verification of self-organizing systems," Expert Systems with Applications, vol. 44 (2016), 230-244.

[10] D. M. Curry and C. H. Dagli, "Establishing Rules for Self-Organizing Systems-of-Systems," Procedia Computer Science, vol. 114 (2017): 14-18.

[11] Y. Jin and C. Chen, "Cellular self-organizing systems: A field-based behavior regulation approach," Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 28, no. 2 (2014): 115-128.

[12] Ashlock and S. McNicholas, "Fitness Landscapes of Evolved Apoptotic Cellular Automata," IEEE Transactions on Evolutionary Computation, vol. 17, no. 2 (2013): 198-212.