Automated Discovery of Self-Replicating Structures in Cellular Space Automata Models

Jason D. Lohn[1]

Technical Report UMIACS-TR-96-60 (CS-TR-3677)
August 1996

Abstract This thesis demonstrates for the first time that it is possible to *automatically* discover self-replicating structures in cellular space automata models rather than, as has been done in the past, to design them manually. Self-replication is defined as the process an entity undergoes in constructing a copy of itself. Von Neumann was the first to investigate artificial self-replicating structures and did so in the context of cellular automata, a cellular space model consisting of numerous finite-state machines embedded in a regular tessellation. Interest in artificial self-replicating systems has increased in recent years due to potential applications in molecular-scale manufacturing, programming parallel computing systems, and digital hardware design, and also as part of the field of artificial life.

In this dissertation, genetic algorithms are used with a cellular automata framework for the first time to automatically discover self-replicating structures. The discovered self-replicating structures compare favorably in terms of simplicity with those generated manually in the past but differ in unexpected ways. This dissertation presents representative samples of the self-replicating structures and analyzes them both quantitatively and qualitatively. In order to effectively search the underlying rule space of such automata models, a fitness function consisting of three independent criteria is designed and successfully applied. Also, a new cellular space automata model called effector automata is introduced. It is shown to be more computationally feasible and to promote the discovery of more self-replicating structures as compared to the cellular automata models used in previous studies. In addition, a new paradigm for cellular space models with weak rotational symmetry called component-sensitive input is introduced and shown to facilitate discovery of self-replicating structures. The results presented suggest that genetic algorithms can be powerful tools for exploring the space of possible self-replicating structures. Furthermore, this research sheds light on the nature of creating self-replicating structures and opens the door to further studies that could eventually lead to the discovery of new self-replicating molecular structures.

[1]Departments of Computer Science and Electrical Engineering, University of Maryland, College Park, MD 20742

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Self-replicating systems are systems that have the ability to produce copies of themselves. Biological organisms are the most familiar examples of such systems, and until the late 1940s, the only instances formally researched. Mathematicians and scientists began studying *artificial* self-replicating systems when it became desirable to gain a deeper understanding of complex systems and the fundamental information–processing principles involved in self–replication [von Neumann51, von Neumann66]. The initial models consisted of abstract logical machines, or *automata*, embedded in cellular spaces [Arbib66, Codd68, Holland76, Langton84, Reggia93]. In addition to automata, other computational models such as those based on traditional programming languages continue to be the main subject of research [Ray92, Koza94]. Physical models exhibiting self-replication such as mechanical and biochemical models have also been constructed and studied [Penrose58, Orgel92, Hong92].

The previous computational models of self-replication in cellular spaces have all been manually designed, a very difficult and time-consuming process. This research introduces the use of genetic algorithms to discover automata rules that govern emergent self-replicating processes. A new model consisting of movable automata embedded in a cellular space is introduced in this context, and is shown to have desirable properties when compared to von Neumann's cellular automaton model. Given dynamically evolving automata, identification of effective performance measures, called fitness functions, for self-replicating structures is a difficult task, and we give multiple solutions to this problem. A genetic algorithm using three fitness criteria was applied to automate rule discovery. The results indicate that the fitness functions employed are effective and that genetic algorithms can be used to successfully discover rules for self-replicating structures. As a result of obtaining large quantities of self-replicating structures, a qualitative classification system is identified and dynamical systems issues are investigated. This investigation indicates that such structures are situated in the phase transition between periodic and chaotic behaviors.

1.1 Motivations

A better understanding of self-replicating systems and the automatic discovery of such systems could be useful in a number of ways, for both practical and theoretical purposes. These are described below.

Nanotechnology

Research concerning atomic-scale manufacturing technologies or "nanotechnology" suggests that self-replicating devices will play a key role [Drexler89], and some researchers have already gained insight from the early work on hand-designed self-replicating systems [Merkle94]. In this technology, *assemblers* are microscopic devices resembling industrial robot arms that are used to build molecular machines. From [Drexler89, pg. 503]:

> If assemblers are to process large quantities of material atom-by-atom, many will be needed; this makes pursuit of self-replicating systems a natural goal.

In addition to the fundamental question of how to bring about such artificial self-replication, issues faced by the designer of self-replicating assemblers include self-inspection, the halting of the self-replication process, size minimization, and the choice of instruction encoding. These potentially difficult design issues could be abated by systems that can automatically design self-replicating assemblers. This is the theme of this dissertation.

Within nanotechnology, researchers are also investigating engineering custom molecules. If the basic physical processes can be identified and represented effectively, automatic discovery approaches might be applied to discover new self-replicating molecular structures.

Programming Massively Parallel Computers

Programming massively parallel computers has traditionally been a difficult task, and to date, only a relatively small number of applications have made use of massive parallelism. It has been proposed that evolutionary bred self-replicating programs could facilitate programming these systems [Ray92]. Self-replicating sub-programs would compete for the available processors, and those that performed better with respect to the target application, would be allowed to create perfect and imperfect (mutation) copies as offspring. Similar experiments performed on sequential computers have shown that self-replicating programs can optimize their algorithms by a factor of 5.75 in a few hours of real time [Ray92].

Programming Cellular Automata

Cellular automata (CA) are a class of discrete dynamical system models in which many simple components interact to produce complicated patterns of behavior. A lattice of cells which represent identical finite state machines defines the space of the CA. The behavior of each cell is governed by a global transition rule which specifies the next state for every possible present state condition. This transition rule is typically a very large table of transitions and is thus very difficult to manually program. Since CAs have found wide application in science and engineering, automatically programming them would be greatly beneficial.

Anti-virus Technology

Computer viruses are programs that use the resources of a host computer system to passively self-replicate and "infect" computers. Because viruses can be destructive, creating effective anti-virus techniques is an important area of research. An important anti-virus method centers around scanning disks and memories for known viruses and then executing a repair operation if possible. A complementary approach is to monitor the computer's behavior and watch for telltale signs of virus activity. These approaches have been somewhat successful, but it is believed that a biologically-inspired "immune system" approach would be effective in keeping up with the accelerated creation of new viruses and the increased interconnectivity of worldwide computers [Kephart94]. Thus, understanding the self-replication processes that govern viruses is an important area of research.

Origins of Life

Contemporary theories [Miller74, Watson87] of the origins of life postulate a prebiotic period of molecular replication before the emergence of living cells. Investigating the fundamental information processing mechanisms underlying self-replication can help us to answer questions concerning the minimum information content needed for emergence of the first replicating molecules. Analyzing "incubation periods" required for spontaneous emergence to occur in artificial systems can shed light on the origin of life under both terrestrial and extraterrestrial conditions. Self-replicating models may also lead to a better understanding of the biology of life on Earth, based on the assumption that the rules underlying biological processes might also apply to artificial environments and structures.

Studying computational models of self-replicating phenomena has certain advantages compared to laboratory-based chemical experiments, which are also underway [Hong92, Orgel92]. Specifically, computational models allow the experimenter to precisely control the details and parameters of experiments. Computer simulations are open to repeated internal inspections and permit large numbers of experiments. They are also helpful in separating specific chemical properties from the information processing properties present in the simulated system (for example [Chou94]). In addition, with the availability of more powerful computers, larger and more complex systems can be simulated.

Artificial Life

The field of Artificial Life (ALife) which studies life-like behaviors (such as self-replication) from a computational perspective was largely born out of studies [Langton86] based on cellular automata. From [Langton88, pg. 1]:

> Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the analysis of living organisms by attempting to synthesize life-like behaviors within computers and other artificial media. By extending the empirical foundation upon which biology is based beyond the carbon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating life-as-we-know-it within the larger picture of life-as-it-could-be.

Thus, biology is seen as a top-down, *analytic* study of the material basis of life, whereas ALife is a bottom-up, *synthetic* study of the formal basis of life. Defining "life" in a precise manner is difficult due to the presence of organisms that are sterile, and organisms that lack a metabolism, such as viruses. However, self-replication is generally seen as a fundamental property of life [Farmer91]. From this perspective, self-replicating systems play a critical role in advancing ALife research.

Other Motivations

Additional incentives for studying the automatic discovery of self-replicating systems include:

- Topics in the study of dynamical systems theory could potentially benefit from this research. Specifically, hypotheses which propose that systems having complexity similar to biological organisms are near the phase transition between complex and chaotic systems may be supported.

- The transport of large numbers of industrial machines to the moon or Mars would be prohibitively costly. If a self-replicating machine that used locally-available materials could be developed, this would make commercialization more feasible. A NASA study of self-replicating lunar factories [Freitas82] investigated the requirements for this type of self-replicating system.

- Previous research [Laing76] has described design possibilities for molecular realizations of automata, especially with respect to self-repair and self-inspection, which are closely related to self-replication.

- Recent work on self-replicating digital electronic hardware [Mange94] seeks to create hardware systems that can self-replicate, self-repair, and evolve.

1.2 Contributions

The main contributions of this thesis are as follows.

Automatic Discovery of Self-Replicating Structures

This is the first work to show proof that it is possible to automatically discover self-replicating structures in cellular space automata models. Genetic algorithms are used in conjunction with novel fitness functions, and the quantities of discovered structures found are shown to be statistically significant. The discovered structures, presented in Chapter 5, compare favorably in terms of simplicity with those generated manually in the past [Reggia93]. However, more interesting is that these replicating structures differ in unexpected ways from those developed in previous automata models. For example, they all move during replication, and all generate active unused components. Furthermore, many past self-replicating structures have relied upon foreign components (i.e. components that do not comprise the original structure) to aid in directing the self-replication process. The automatically discovered structures presented in this thesis are able to self-replicate without such additional components, making them yet simpler than the manually-designed structures.

Fitness Functions for Self-Replication

This is the first work to derive fitness functions for self-replication in any cellular space automata model. These fitness functions, derived in Chapter 4, are general and are applicable to many cellular space models. In addition they may also be used with other optimization and search techniques. Finding appropriate functions is a difficult task for reasons related to assigning partial fitness measures. For example, a function based on counting the number of replicants is useless early on as there will generally be none. It was found that a fitness function based on multiple performance criteria, such as growth of individual components and relative position measures was needed. This multiobjective optimization problem was solved by weighting the three criteria via experimentation, and by using an *adaptive* fitness function – a second genetic algorithm was successfully employed to dynamically evolve higher performing fitness functions.

Another impediment to deriving these fitness functions is the tendency to impose biases on the self-replication process, instead of allowing such processes to evolve "naturally". An example of such biases would be to assign fitness based on how well an evolving structure matched a predefined template. This difficulty is overcome by designing fitness functions that use statistics that do not contain absolute position information in their calculation. To further guard against bias, key fitness function parameters are optimized using a second, higher-level meta-fitness function. Penalty functions are derived and also aid in this regard.

A New Paradigm for Weakly Rotation-Symmetric Cellular Space Models

A new paradigm for weakly rotation symmetric cellular space models is introduced in Chapter 3 which significantly reduces rule table size without adversely affecting the flexibility of the model. Called component-sensitive input, this technique is general enough so that it may be applied to any cellular space model having weak rotational symmetry. Interest in cellular space models that incorporate weak rotational symmetry has grown in recent years, and this technique, introduced in Chapter 3, allows larger models (more states) to be computationally simulated. In the context of automatically programming cellular space rule tables, this technique greatly reduces the search space size, thus facilitating the search process. Experimental results using genetic algorithms are presented which verify this.

A New Cellular Space Model

A new cellular space automata model called Effector Automata (EA) is introduced in Chapter 3 and shown to have the following advantages over similar models such as von Neumann's cellular automata. First, the EA model more closely parallels physical systems by directly incorporating movement and characteristics of mass-preservation physics. In each cell, a new automaton can only be created as a result of cell division, whereas other models generally allow spontaneous generation of such automata. Thus, emergent structures in EA simulations have a higher degree of realism than those of other models. Second, by incorporating movement and automaton division, the EA model is better suited to studying self-replicating systems. Third, simulation of the EA model is shown to be significantly less resource intensive, and hence more computationally feasible, especially as the number of states increases. Lastly, the EA model allows the designer to create cellular space models at a higher level of abstraction using less rules than that of CA models. For example, in CA models, more rules are needed to encode the movement of an automaton. Those CA rules are state transitions which are of a lower-level as compared to EA movement-actions, and thus they can be more difficult and tedious to work with.

Comparison of Search Techniques

Effective techniques for searching extremely large search spaces are compared with respect to the automatic discovery of self-replicating structures in the CA and EA cellular space models. The techniques compared in Chapter 5 are genetic algorithms (GAs), and multiple restart stochastic hillclimbing, and simulated annealing. It is found that GAs outperform the other techniques showing that GAs are indeed effective at finding self-replicating structures.

While genetic algorithms have been previously applied in other computational models involving cellular automata [Richards90, Mitchell94], their use to discover self-replicating structures is daunting because of the large "chromosome" needed. Furthermore, the computational burden of simulating a large population of automata models is enormous, which presumably accounts for the absence of research in this area. In spite of strides made in reducing the computational load (by using the EA model and component-sensitive input techniques), the experimental results presented in this thesis were run on parallel supercomputers, and individual runs often required days to complete.

Classification of Self-Replicating Structures

Influenced by biological models of self-replication, recent work in self-replicating structures has relaxed the requirement for universal computation and construction. Such models have not presented a precise definition and framework regarding self-replicating structures. This thesis presents the first detailed framework for studying self-replicating structures. Beginning with the definition of a self-replicating structure, relevant set-theoretic functions and terms are introduced.

In addition, this is the first work to define a classification system for self-replication in cellular space automata models. Previously, manual derivation of self-replicating structures produced tens of structures. With the capability to automatically generate thousands of such structures, it is possible to demarcate qualitative classes of self-replication processes.

Simulation System

To carry out the research described in this thesis, a large software system was created in which a wide variety of experiments may be conducted. Two cellular space models are supported and along with two rotational symmetries. This system also allows automatic programming of cellular space models via genetic algorithms, and permits researchers to experiment with many model parameters. Since certain experiments can require enormous amounts of processing, a version that runs on parallel supercomputers was developed.

1.3 Content of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 reviews previous related work and background material which is relevant to the thesis. This covers research in cellular space automata models, self-replication in these models, genetic algorithms, and rule discovery using genetic algorithms. Chapter 3 presents a new cellular space model called Effector Automata, develops its theory, and contrasts it with cellular automata models. Of particular concern are the search space sizes in which the genetic algorithm will search to discover rule tables that promote self-replicating behavior. A new paradigm for automata input, called component-sensitive input, is introduced which significantly reduces the search space size (in both cellular and effector automata models), while preserving the desirable properties of the model. By reducing the search space size, search techniques, like the genetic algorithm can discover more rule tables that promote self-replicating behavior. Chapter 4 presents the detailed design of a genetic algorithm for the automatic discovery of self-replicating structures, the first such design to be reported. The problem of deriving fitness functions that promote self-replicating behaviors is shown to be a difficult problem, and novel solutions are given by deriving general fitness functions comprised of multiple criteria. To describe self-replicating processes more formally than previous work in this area, a framework is developed including a precise definition of a self-replicating structure. Chapter 4 also describes how multiobjective optimization was accomplished by the use of a second GA, called a meta-level GA. Chapter 5 presents the results and analysis of the experiments to automatically discover self-replicating structures in both cellular automata and effector automata models. Representative GA-discovered self-replicating structures are shown using varying seed sizes. Statistical significance measures of the experimental results, and GA performance curves are also described and analyzed. Since this is first work to produce hundreds of self-replicating structures, a new classification system is devised to categorize the behavior of self-replicating structures. Chapter 6 contains a summary of the results and suggestions for future work in this area.

Chapter 2

Background and Previous Work

To better understand the results of this dissertation, this chapter briefly reviews cellular space automata models, self-replicating systems, genetic algorithms, and the relevant literature in these areas.

2.1 Cellular Automata

Cellular automata (CA) are a class of discrete dynamical system models in which many simple components interact to produce potentially complex patterns of behavior. CAs have been used to model a broad range of natural phenomena and in engineering applications, for example: astrophysical modeling [Perdang93], heart fibrillation [Burks74], ecological processes [Hogeweg88], fluid dynamics [Frisch86], and image processing [Preston84].

In a cellular automata model, time is discrete, and space is divided into a lattice of cells, each representing a finite state machine or automaton. At each time-step, each automaton uses the same function δ or *rule table*[1] to determine its next state as a function of its current state and the state of neighboring cells. This set of neighboring cells is called a *neighborhood*, the size (n) of which is commonly 3 cells in 1-D CAs, and 5 or 9 cells in 2-D models (see Figure 2.1). Note that, by convention, the center cell is included in its own neighborhood. Each cell can be in one of k possible states, one of which is designated the *quiescent* or inactive state. When a quiescent cell has an entirely quiescent neighborhood, a widely accepted convention is that it will remain quiescent at the next time-step.

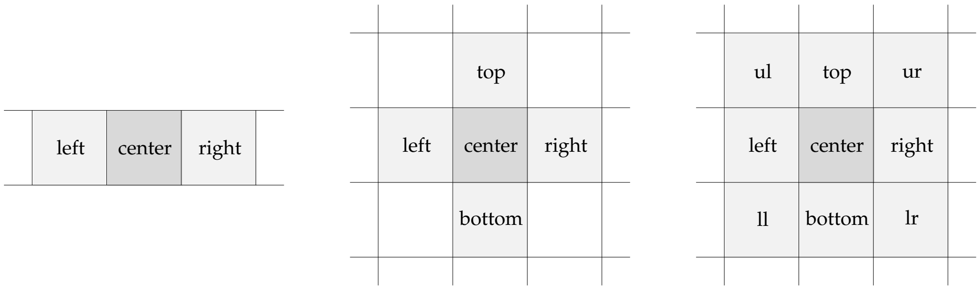
The CA rule table is a complete[2] list of transition rules that specify the next state for every possible neighborhood combination. In a 2-D, 5-neighbor model using the von Neumann neighborhood, the individual transition rules would be of the form:

$$\mathrm{CTRBL} \rightarrow \mathrm{C}'$$

which specifies the states of the Center, Top, Right, Bottom, and Left positions of the neighborhood's present state, and C′ represents the next state of the center cell. Using this neighborhood with 2-state automata, consider the cellular automaton for the parity function shown in Table 2.1.

[1] This is frequently referred to as a "transition function", "transition rule" or simply "rule" in the CA literature, but "rule table" will be used in this work for clarity.

[2] Rule tables are sometimes partially specified by listing only the rules needed to enable a specific behavior.



(a) (b) (c)

Figure 2.1: Common neighborhood templates in 1-D and 2-D CA: (a) 3-cell neighborhood; (b) 5-cell von Neumann neighborhood; (c) 9-cell Moore neighborhood

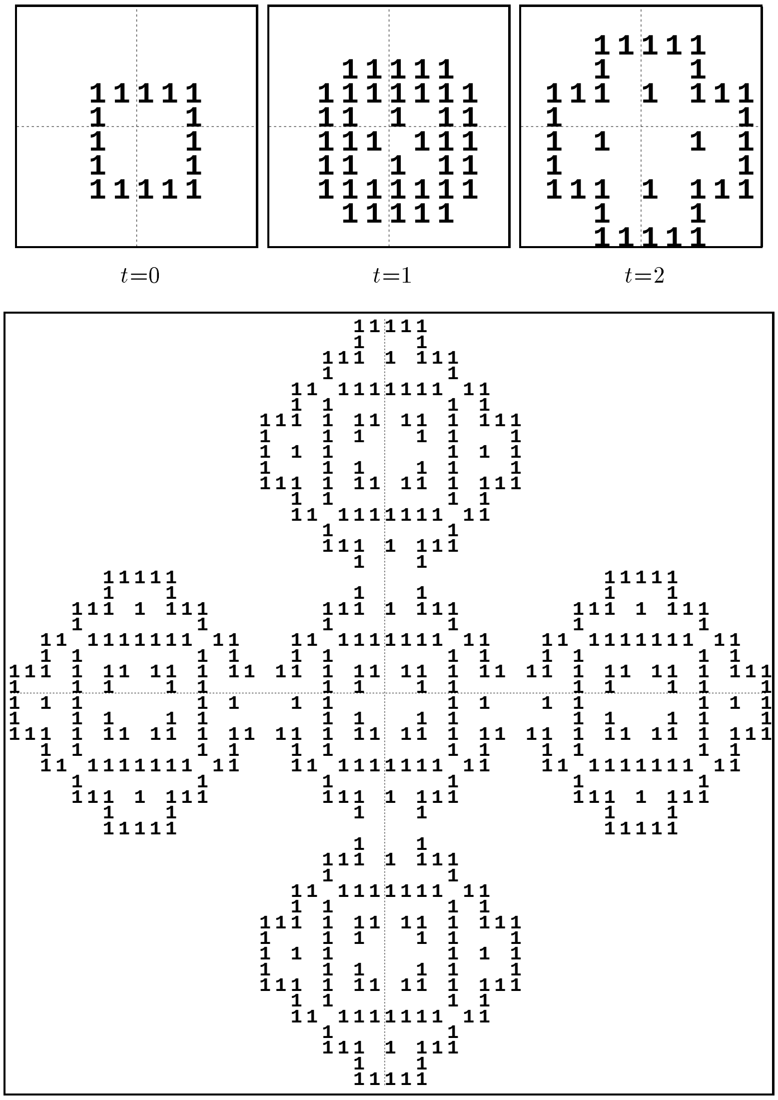
States are represented by 0 and 1, and for each of the $2^5 = 32$ neighborhoods a transition rule is specified. The next state is a 1 if the parity of the neighborhood cells is odd, and 0 if even. Figure 2.2 shows the first three time-steps when the initial configuration is a 5×5 square pattern. Also shown is the complex pattern that emerges at $t = 22$. If the space is not constrained, complex patterns will continue to form and the structure will expand outward indefinitely.

CTRBL	C′	CTRBL	C′	CTRBL	C′	CTRBL	C′
00000	0	01000	1	10000	1	11000	0
00001	1	01001	0	10001	0	11001	1
00010	1	01010	0	10010	0	11010	1
00011	0	01011	1	10011	1	11011	0
00100	1	01100	0	10100	0	11100	1
00101	0	01101	1	10101	1	11101	0
00110	0	01110	1	10110	1	11110	0
00111	1	01111	0	10111	0	11111	1

Table 2.1: Parity rule table for 2-state, 5-neighbor cellular automata. There are 32 transition rules that comprise this rule table.

The underlying space of CA models is typically defined as being *isotropic*, meaning that the absolute directions of north, south, east, and west are indistinguishable. However, the *rotational symmetry* of cell states is frequently varied. *Strong* rotational symmetry implies that all cell states are unoriented, meaning that each neighbor to a cell has no special absolute nor relative position. *Weak* rotational symmetry implies that at least some of the cell states[3] are directionally oriented, meaning that the cell designates specific neighbors as being its top, right, bottom, and left neighbors. For example, the cell state designated $\uparrow$ in von Neumann's work is weakly-symmetric and thus permutes to different cell states $\rightarrow$, $\downarrow$, and $\leftarrow$ under successive 90° rotations. It represents one oriented *component* that can exist in four orientations. In the parity rule of Table 2.1, both states (0,1) are strongly rotation symmetric. In CAs that contain both weak and strong rotationally sym-

[3] The quiescent state is always a strongly rotation symmetric cell state and is thus included in CA models with weak rotational symmetry.



t=0 t=1 t=2

t=22

Figure 2.2: Parity CA configurations for the first three time-steps, and at t=22. Individual cells are at a reduced scale in t=22. Axes are superimposed for frame of reference.

metric states, it is common to represent the "strong" states using symbols that appear rotationally symmetric (e.g. $\circ$, $+$, $\times$), and the "weak" states using symbols that are not rotationally symmetric (e.g. $\uparrow$, `A`, `L`).

In addition to isotropic spaces, non-isotropic spaces are also possible. In a non-isotropic space one direction is specially designated and is known to all automata. Thus every automaton has exactly the same orientation and senses an "absolute north" direction. A diagram summarizing the relationships of the models described above in shown in Figure 2.3.

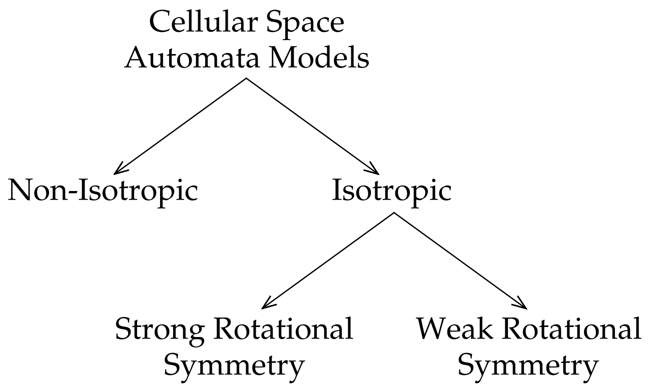


Figure 2.3: Dichotomy of cellular space automata models with respect to the underlying cellular space.

2.2 Self-replicating Structures in Cellular Space Models

2.2.1 Cellular Automata Models

A self-replicating structure in a cellular automata model is informally defined as follows. In the CA model, one state is designated the quiescent state, and the remaining states are considered active. A self-replicating structure is represented as a configuration of contiguous active cells, each of which represents a component of the machine. At each discrete time-step, each automaton (cell) uses an identical rule table to determine its next state as a function of its current state and the state of its immediate neighbor cells. Based solely on these concurrent local interactions, an initially specified structure (at time $t = 0$) goes through a sequence of steps to construct a duplicate copy of itself. The replica can be displaced and perhaps rotated relative to the original at a later time t'. A two-dimensional cellular space model illustrating this is shown in Figure 2.4. In this particular example, cell-states are integers, and the quiescent state (0), is depicted by an empty cell for clarity. The other states (1, 2, 3) are shown forming a five-component structure ($t = 0$) which then self-replicates over time and produces a replicant at a later time ($t = t'$).

The type of self-replication described above is technically asexual reproduction: an offspring is an exact copy of the parent. Sexual reproduction in CA, mentioned in the next section (page 20) with respect to a previous work, is beyond the scope of this thesis.

The mathematician John von Neumann conducted research on self-replicating automata between 1948-1953 [von Neumann66]. Since he was the first person to formally study these systems,

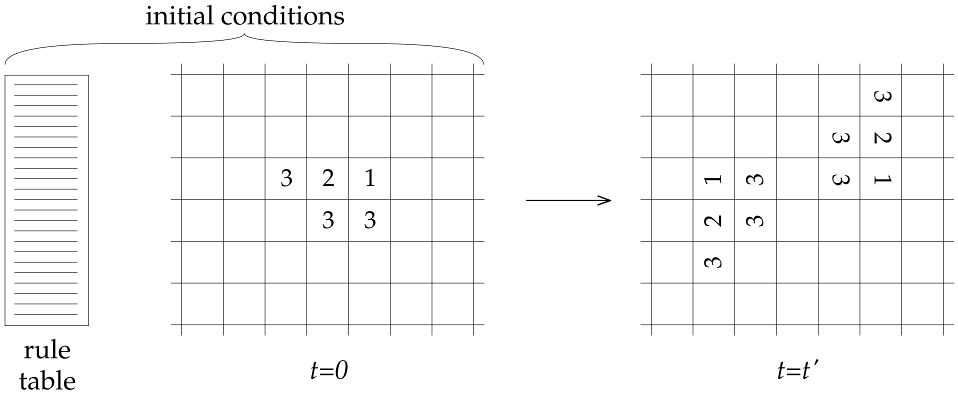


Figure 2.4: Illustration of a self-replicating structure in a 2-D cellular space model.

it is interesting to point out his motivations. In [Burks70] it is suggested that von Neumann was envisioning a systematic theory of natural and artificial automata, mainly because he believed that designing complex systems (e.g. large-scale computers) would be difficult without such a theory. In [McMullin92b] it is asserted that von Neumann was primarily interested in *spontaneous growth* of complexity (particularly through Darwinian evolution) and that studying self-replication was a suitable means to this ends[4].

Von Neumann conceived of five models of self-replication: kinematic, cellular, excitation-threshold-fatigue, continuous, and probabilistic [von Neumann66]. The kinematic model was the forerunner to the cellular model. Stanislaw Ulam suggested the cellular model during a discussion of the kinematic model since it was thought the cellular framework would be more suitable to mathematical and logical analysis. Because of this, the cellular automata model became the only model formally researched by von Neumann, and was used to design the first logical automaton capable of directing its own replication. An overview of this design is shown in Figure 2.5. It consisted of a two-dimensional array of cells, each of which could be in one of 29 states (29 was the least number of states he could devise). A group of cells that comprised the "construction-arm" functioned to construct a new automaton. The tail-like "tape" contained the instructions that specified how to build the new structure. Since the machine would construct any configuration specified on the tape, von Neumann's machine is said to be *construction universal*. Thus, when the instructions on the tape specify how to build a copy of itself, self-replication can proceed.

One measure of the complexity of von Neumann's logical machine is to count the number of 29-state cells that comprise the self-replicating entity. Estimates range from 40,000 – 200,000 cells. This high degree of complexity seemed to be consistent with the remarkable complexity of biological self-replicating systems. However the research of E. F. Codd and Christopher Langton reported simpler self-replicating structures in CA. Codd produced a sheathed loop structure embedded in an 8-state, 5-neighbor, 2-D CA [Codd68]. Langton took a component of Codd's structure and made further reductions. He describes an 8-state, 86-component, sheathed-loop self-replicating structure [Langton84] depicted in Figure 2.6(a). In [Byl89], an even smaller 6-state, 12-cell self-

[4]The work presented in this thesis is very much related to this theme since genetic algorithms, which are based on biological evolution, are used extensively.

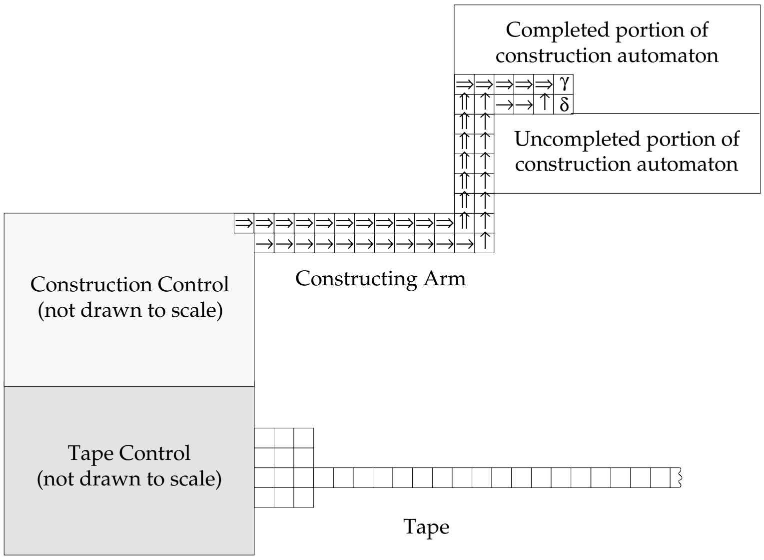


Figure 2.5: Overview of von Neumann's design for a self-replicating automaton (from [Burks70]).

replicating loop is presented (see Figure 2.6(b)).

```
 XXXXXXXX
X0+ 0L 0LX
X XXXXXX X
X+X    X0X
X0X    X0X
X X    X0X
X+X    X0X
X0XXXXXX0XXXXX
X +0 +0 +00000X
 XXXXXXXXXXXXX
```

(a)

```
 XX
XLOX
XL+X
 X*
```

(b)

Figure 2.6: Initial configurations of self-replicating loops: (a) 8-state, 86-component structure of Langton; (b) 6-state, 12-component structure of Byl.

Subsequent research [Reggia93] has shown that even simpler, non-trivial self-replicating structures do exist. For example, Figure 2.7 shows structure `UL06W8V`, so named because it: is an unsheathed loop (UL), is comprised of six components, uses weak rotational symmetry, is embedded in a model in which each cell may be in 1 of 8 states, and uses the von Neumann neighborhood. The partial rule table that governs its self-replication process used only 20 rules. The structure that undergoes self-replication is seen at $t = 0$ in Figure 2.7(a). At $t = 8$ the first replicant can be seen detached from the original structure. Then these two structures each begin a process of self-replication until several time-steps later, a diamond-shaped colony has formed. The center of the colony contains inactive groups composed of four cells each, while the perimeter continues to expand indefinitely.

Related to the previous work are the sexually-reproducing CAs of Paul Vitányi. Using an 8-

```
......      ......      ......      ......      ......      ......
......      ......      ......      ......      ......      ...<..
.OO...      .O<...      .vL...      .LO.O.      .OO^O<      .O<.vL
.L>OO.      .OL>O.      .OOL>.      .>OOL^      .L>OOL      .OLvOO
......      ......      ......      ......      ......      ......
......      ......      ......      ......      ......      ......
```

$t = 0$ $t = 1$ $t = 2$ $t = 3$ $t = 4$ $t = 5$

```
......      ..O...      ..O...      ..O...      ..O...
..#<..      ..O...      ..O...      ..O...      ..^...
.vL.LO      .LO.OO      .OO.O<      .O^.vL      .vL LO
.OO.>O      .>O.L^      .L^.OL      .OL.OO      .OO vO
...>..      ...>#.      ....O.      ....O.      ....O.
......      ......      ....O.      ....O.      ....O.
```

$t = 6$ $t = 7$ $t = 8$ $t = 9$ $t = 10$

(a)

```
             ^
             L
            LO OO
            >O OO

         OO OO OO
       ^ OO OO OO   ^
       L            L
      LO OO OO OO LO OO
      >O OO OO OO >O OO

   OO OO OO OO OO OO OO
 ^ OO OO OO OO OO OO OO  ^
 L                       L
LO vL OO OO OO OO OO OO LO OO
>O OO OO OO OO OO OO OO >O L^
   O                        v
   LO OO OO OO OO OO OO OO  O
   >  OO OO OO OO OO OO OO

      OO vL OO OO OO OO
      OO OO OO OO OO L^
         O            v
         LO OO OO OO  O
         >  OO OO OO

            OO OO
            OO L^
                v
                O
```

(b)

Figure 2.7: Structure UL06W8V [Reggia93] (a) first 10 time-steps; (b) after several generations.

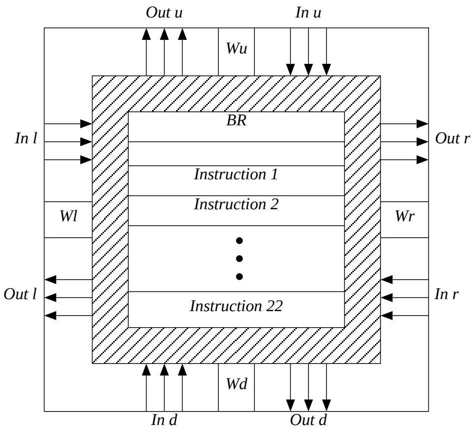


Figure 2.8: Automaton in Arbib's CT-Machine model.

state, 5-neighbor CA model similar to Codd's, he was able to produce two structures capable of sexual reproduction [Vitányi73]. He argues that in transitioning from asexual to sexual reproduction requires a change in the number and structure of instruction tapes. He creates **M**-type (male) and **F**-type (female) automata, each containing two, nearly identical instruction tapes. Although his automata are quite complex, he shows that sexual reproduction of automata is possible, and that the recombination process is similar to that of nature.

2.2.2 Arbib's Model

In the mid-1960s, Michael Arbib observed that the large degree of complexity of von Neumann's and Codd's self-replicating automata could be greatly reduced if the fundamental components were more complex [Arbib66, Arbib69a]. His rationale for doing this was to adopt a hierarchical approach, where his automata would be analogous to cells, as opposed to macromolecules. He created a version of the 2-D cellular space automata model called *Constructing Turing Machines*, or *CT-machines* [Thatcher70]. Each cell in this space contains a finite-state automata that execute short 22-instruction programs (see Figure 2.8). The instructions consist of actions such as weld and move, and internal control constructs such as if/then and goto. Self-replication occurs when individual CT-machines copy their instructions into empty cells. Structures composed of multiple CT-machines are able to move as one unit since individual automata can be welded to each other.

Using elements from the CT-machine model, Arbib also describes [Arbib67] the Mark II cellular space model. In this model, automata are capable of dividing into two automata or self-destruction, and cells in the cellular space are either empty or occupied by an automaton (as opposed to having a quiescent state). The new cellular space model presented in Chapter 3 retains these three properties, however the automata are much simpler than CT-machines.

2.2.3 Holland's Model

In the mid-1970s John Holland explored automatic discovery of self-replicating automata by focusing on *spontaneous emergence* of such structures. He developed a theoretical framework and sought to provide existence proofs for the spontaneous emergence of a class of artificial self-replicating systems [Holland76]. Holland defines a set of model "universes" containing abstract counterparts to rudimentary chemical and kinetic mechanisms such as bonding and movement. He wanted to loosely model natural chemical processes (diffusion, activation) acting on structures composed of elements (nucleotides, amino acids) to show that even with random agitations, the tendency of such a system would not be sustained randomness, but rather, life "in the sense of self-replicating systems undergoing heritable adaptations."

Although these α-Universes, as they are called, are termed cellular automata models in Holland's paper, they actually have little in common beyond discretized time and space. The concept of a state is represented by elements that are logical abstractions of physical entities (e.g. atoms) and obey the conservation of mass. In Figure 2.9 a 1-dimensional example α-Universe is shown along with a table of elements and "codons". Elements are the fundamental units, and codons encode elements (the analogy is that of amino acid triplets encoding protein sequences). Many interactions among the elements are strictly local as in CA, but some are localized to aggregate structures (strings of bonded elements). The elements themselves can be thought of as automata during the first of three "phases" of each discrete time-step. However, during the second and third phases, they are *acted upon* by the physics of the α-Universe. Holland calls these forces "operators" and defines four: bonding, movement, copy, and decode. Because of these global operators, it would be impossible to specify a CA-type rule table for an α-Universe. As an example, the "copy" operator would be activated if the sequence `-0:`$e_1e_2\cdots e_l$`-` formed (e_i being one of the three elements), and it would cause elements to be reshuffled so that a codon-encoded copy of the string $e_1e_2\cdots e_l$ would be assembled.

•••	-	0	:	1	0	0	1	-	N_0	N_1	-	-	0	:	•••

Element	Codon
0	N_0N_0
1	N_0N_1
:	N_1N_0

Figure 2.9: An example of a few cells from an α-Universe

Holland parameterizes important aspects of the α-Universes and then uses these to derive formulas that predict structure lifespans, population densities, and certain event probabilities. One of those predictions is an expression for the expected time required for emergence of a self-replicating system. Substituting reasonable values for the parameters, a waiting time of 1.4×10^{43} time-steps is computed [Holland76, pg. 399]. Since this is a tremendous number (there are roughly 10^{17} seconds in 10 billion years), it can be said to never produce self-replicating entities. Relaxing the requirement from *fully* self-replicating to *partially* self-replicating, a similar expression is derived. Again substituting reasonable parameter values, this time a waiting period of 4.4×10^8 time-steps (4.4×10^8 seconds is about 14 years) is the result. Since this is a reasonable amount, it lends credence to spontaneous emergence of self-replicating structures in general, given that Holland's model and derivations are accurate. No computer simulations of Holland's model were published until very recently, even though such simulations were quite feasible (as the 4.4×10^8 time-step experiment shows). In [McMullin92a], an empirical investigation into Holland's work is reported. There it is claimed that some of the conjectures in Holland's model were flawed since experimental results showed that the self-replicating structures would go extinct after modest time periods. Regardless of whether the original analysis is valid, it remains one of the only studies of its kind reported to date and raises important theoretical questions regarding emergence of self-replicating structures.

2.2.4 Summary of Self-Replicating Automata

As a summary of work done thus far and for comparison purposes, Table 2.2 lists the models and relevant data concerning self-replicating automata research. Rotational symmetry is listed since it is a significant variation of the models shown. The neighborhood sizes 5 and 9 correspond to the von Neumann and Moore neighborhoods, respectively. The sizes of the self-replicating structures are measured in cells, and are frequently rough estimates since many systems were never implemented.

Year	*Model Type*	*Dim.*	*Rot. Symmetry*	*States per cell*	*Neighborhood size(s)*	*Structure size(s)*[a]	*Reference*
1951	CA	2D	weak	29	5	$> 10^4$	[von Neumann66]
1965	CA	2D	strong	8	5	$> 10^4$	[Codd68]
1966	CT-Mach.	2D	weak	$\approx 10^{100}$	5	$\approx 10^2$	[Arbib66]
1973	CA	2D	strong	8	5	$> 10^4$	[Vitányi73]
1976	α-Univ.	1D	strong	5	-[b]	(60)[c]	[Holland76]
1984	CA	2D	strong	8	5	86	[Langton84]
1989	CA	2D	strong	6	5	12	[Byl89]
1993	CA	2D	both	6,8	5,9	5–48	[Reggia93]

[a] Many systems were never implemented, so some values are broad approximations.
[b] No fixed neighborhood size.
[c] Theorized, neither proven nor implemented.

Table 2.2: Summary of self-replicating automata research.

It is important to remember that all of these self-replicating structures were hand designed. Although Holland's model could potentially automatically identify self-replicating structures, no proof of this has been given.

2.3 Genetic Algorithms

A genetic algorithm (GA) is a stochastic search and optimization technique based on ideas from natural genetics and evolution. Genetic algorithms were originally introduced by John Holland [Holland75]. In recent years GAs have become increasingly popular in engineering design, machine learning, and other areas because they perform well in a wide range of applications [Davis91]. For example, GAs have been applied to circuit design [Shahookar90], neural network design [Harp91], robot control [Davidor91], DNA sequence assembly [Parsons93], and protein-structure prediction [Dandekar92].

The solution space or search space for a given problem is a set of points representing all possible solutions. For each potential solution we can imagine a "fitness landscape" where valleys mark the location of poor solutions and the highest point corresponds to the best possible solution. For complex problems, solution spaces are usually enormous[5], and contain convoluted topological features. GAs effectively comb the solution space and home-in on promising regions by combining partial solutions in ways analogous to how biological genes have evolved. However, like other stochastic techniques, there is no assurance that the GA will converge to the global optimum. When applied properly, GAs are robust and generally good at finding "acceptably good" solutions, especially when dealing with extremely large search spaces.

A GA works by manipulating a pool or *population* of candidate solutions called *chromosomes*. Each chromosome is assigned a fitness value using a *fitness function* according to how well it solves the problem at hand. Highly fit chromosomes are given the opportunities to cross-breed with other members of the pool. Thus offspring are produced, and a new population of candidate solutions is formed with generally a higher proportion of good characteristics than the previous population. Each successive population is called a *generation*, and the GA continues in this fashion until a specified convergence criteria is satisfied. This process is summarized in Figure 2.10.

```
initialize population of chromosomes
evaluate fitness of each chromosome
while (termination criterion not reached) do
   select parent chromosomes for mating
   apply crossover and mutation to produce children
   evaluate fitness of each chromosome
end
```

Figure 2.10: Traditional genetic algorithm.

2.3.1 Genetic Operators

The key mechanisms in the GA are the genetic operators: fitness-based reproduction, crossover, and mutation[6]. To illustrate their use, consider designing a GA to find the global maximum of a function $f(x, y)$. Since chromosomes are frequently encoded as binary strings, x and y are represented as 10-bit numbers giving a chromosome of 20 bits as shown in Figure 2.11(a). A population of such chromosomes is generated randomly and seen in Figure 2.11(b). Then for each chromosome fitness values are computed using the fitness function f. These values are shown adjacent to the chromosomes in Figure 2.11(c). Based on these fitness values, pairs of chromosomes are selected to undergo crossover and mutation to produce the next generation.

It has been argued that the GA derives most of its strength from recombination of partial solutions through the action of crossover. Crossover takes two chromosomes and cuts them into

[5]For example, in a typical chess game, there are about 10^{60} strategies possible [Holland92].

[6]Other operators are known, however only those used in this thesis are discussed here.

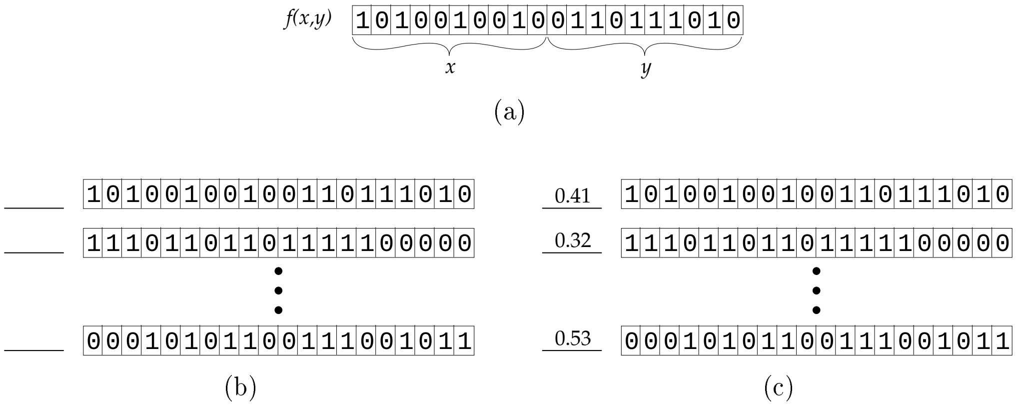


Figure 2.11: Examples of 20-bit chromosomes: (a) single chromosome encoding x and y; (b) randomized initial population before computing fitness function for each chromosome; (c) same population with fitness values.

two pieces at some randomly chosen point, producing two head and two tail segments. The tail segments are then interchanged to produce two new chromosomes (Figure 2.12(a)). After crossing-over, mutation is applied to each child chromosome by complementing randomly selected bits (Figure 2.12(b)). Crossover and mutation are applied probabilistically so that, for example, crossover may not be applied to a given pair of chromosomes[7]. Typical values for crossover and mutation probabilities are 0.6–1.0 and 0.001–0.2, respectively.

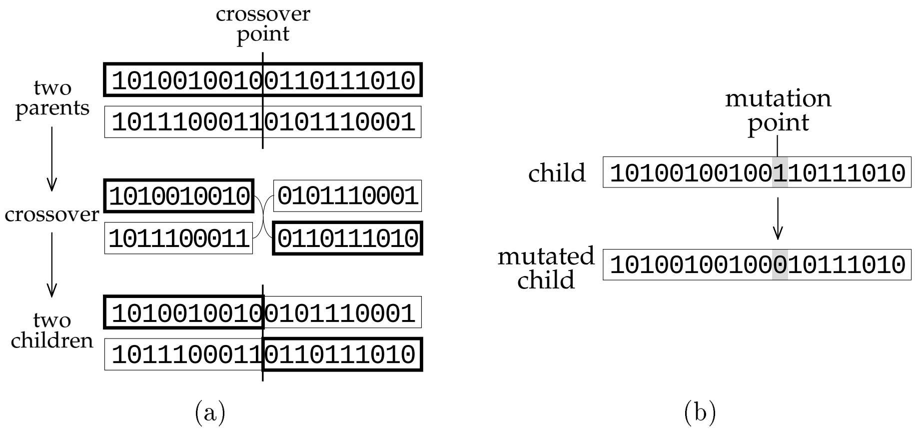


Figure 2.12: Genetic operators: (a) single-point crossover; (b) single point mutation.

As seen in the example of Figure 2.11, before applying a GA to a given problem, the GA designer must find a suitable way to encode solutions so that the genetic operators can act on them. This task is called the *encoding* or *representation* problem, and can be difficult to solve since it is specific to the application domain. A binary encoding scheme was chosen in Figure 2.11 since binary representations are commonly chosen and work well in many applications. Other representation

[7]In such a case the children are duplicates of the parents.

options include many-character, real-valued, and tree encodings. In addition, the natural encoding (i.e., how the problem is presented to the computer before the GA is applied) of a problem has may also work well. At present there is no one method of representation that performs best for all problems.

2.3.2 GA Theory

The fundamental mechanism of a GA is its manipulation of a special class of building blocks called *schemas*. A schema [Holland75] is a template that describes a subset of strings with similarities at certain string positions. For a problem encoded with the binary alphabet 0,1, a schema is represented as a string containing the symbols $\{0, 1, *\}$, where the asterisk represents a "don't care". A schema matches and thus represents a particular string if in each position the schema contains a 0, the string contains a 0, and in each position that the schema contains a 1, the string contains a 1. For example, for strings of length 4, the schema *101 matches the two strings {0101, 1101}. As another example, the schema 0***1 represents the set of all bit strings of length five that start with a 0 and end with a 1.

The benefit of schemata is that they provide a compact way to represent important similarities among strings with high fitnesses. A string of length l is a member of 2^l different schemata since each position may contain its actual value or a don't care symbol. Therefore, a population of size n contains between 2^l and $n \cdot 2^l$ schemata. The GA works to increase the growth of important schemata through reproduction, crossover, and mutation. Since strings with higher fitness functions have a higher probability of being selected, as the genetic algorithm progresses, on average an increasing number of samples contain schema with high fitness. Since crossover may disrupt schemas of large length, genetic algorithms have the result of propagating short schemata of high fitness.

There are many variations on the way in which crossover may be performed. Two-point, multiple-point, and uniform crossover operators have been devised and met with success. These techniques essentially add more crossover points at which to swap segments. The reasoning behind this has to do with the fact that single-point crossover cannot combine certain schemas. For example, an instances of schemas `1********111` and `****1*******` cannot be combined to form `1***1****111`.

2.3.3 Rule Discovery Using GAs

In this section two studies involving rule discovery using genetic algorithms in CA are briefly described. A third important research study can be found in [Jefferson91]. These are briefly mentioned since they are evidence that genetic algorithms can be used successfully in finding high-performance CA rules that yield a desirable emergent phenomena.

The first study describes a method where CA rules are extracted directly from experimental data using a genetic algorithm [Richards90]. The idea was to evolve a CA whose resulting space-time patterns closely reproduced the solidification of NH_4Br from a supersaturated aqueous solution. The model used was a 2-dimensional, 2-state, probabilistic CA with an interesting neighborhood template: in addition to the 5-neighbor von Neumann neighborhood, the authors used an elaborate neighborhood template consisting of additional neighborhood sites as well as previous neighborhood sites. A genetic algorithm was used to search for CA rules. To calculate fitness of a given rule, sequences of digitized images photographed during solidification were compared to candidate rules based on how well the CA's next values correlated to past and present values. The authors reported encouraging results: the genetic algorithm was able to discover CA rules that qualitatively reproduced the dynamical patterns of the solidification process.

The second study involved using a genetic algorithm to discover CA rules for emergent global computation [Mitchell93]. The specific task chosen (called the $\rho_c = 1/2$ task) concerned density classification: given a 149-cell, 2-state, radius 3 neighborhood, 1-dimensional CA model with a random initial configuration (IC), the CA should become all 1s as quickly as possible if the IC is comprised of more than half 1s (analogously for 0s). This problem is trivial for a computing system with global information available, but quite difficult when only local information is available. The authors use a genetic algorithm to discover CA rules that excel at this task, and report finding rules that are 65%-77% accurate[8]. Some of the evolved strategies relied on the presence of large blocks of 1s or 0s as predictors. The innovative strategies focused on large space-time distances to do the computation since communication throughput among cells is limited by locality.

[8]During the writing of this thesis, a genetically evolved rule with 82% accuracy was reported in [Andre96].

Chapter 3

Effector Automata

This chapter presents a new cellular space automata model called Effector Automata (EA) [Lohn95]. The EA model was created in order to have a model whose behaviors would more closely resemble physical systems and characteristics of mass-preservation physics as compared to cellular automata, while retaining many of the desirable properties of cellular automata, such as strictly local interactions among simple rule-based automata, emergent behavior, and massive parallelism. The EA model retains many of the desirable properties of cellular automata models, such as strictly local interactions among simple rule-based automata, emergent behavior, and massive parallelism. However, it more closely resembles physical systems by directly incorporating movement and characteristics of mass-preservation physics. It is shown that the EA model has the following advantages over conventional cellular automata models, especially regarding the development of self-replicating structures. First, because the EA model can incorporate aspects of mass-preservation physics, emergent structures in EA simulations have a higher degree of realism than those of other models. In each EA cell, a new state can only be created as a result of cell division, whereas other models generally allow spontaneous creation of arbitrary cell states. Second, by incorporating movement and automaton division, the EA model is better suited to studying self-replicating systems. This will be discussed further in Chapter 4. Third, simulation of the EA model is shown to be significantly less resource intensive, and hence more computationally feasible, especially as the number of states increases. This is of great significance when evolving behavior through simulated evolution in such models since the computational requirements far exceed resources typically available, including the use of present-day supercomputers.

The Effector Automaton model derives its name from the fact that each automaton can *effect* changes to neighboring cells (primarily through cell movement), whereas in most cellular space models, a given automaton simply changes its own state at each time step. This property is illustrated in Figure 3.1 where a single active cell is seen moving one cell to the right. The cell's original neighborhood consisting of five cells is shown outlined. At $t+1$ the rightmost neighborhood cell is changed as a result of the center cell moving to the right. If this were a cellular automaton, the center cell could only change its own state, and not that of neighboring cells. Note however that the behavior illustrated in Figure 3.1 can also be produced by a cellular automaton with a different mechanism (the right neighbor changes its state). In the general case, it is always possible to construct a CA that can simulate a given EA. However, as shown later, the CA will typically require a rule table that is much larger than that of the EA. This is a significant drawback from a computational perspective.

The EA model was inspired from observing the fundamental mechanisms employed by self-

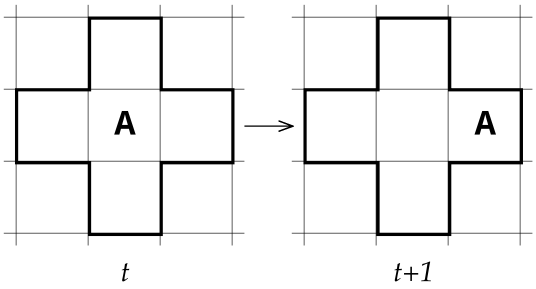


Figure 3.1: Example of a single active EA cell at time t influencing a neighboring cell at $t+1$ by moving to the right.

replicating structures in previous CA models. The key mechanisms observed were that of automata movement, automata division, and automata self-destruction. By codifying these primitives directly into condition-action rules, the automata execute actions instead of state transitions. Thus, EA automata are thought of as being closer to physical machines than to information processors. Although the EA model was not specifically designed to be biologically realistic, it is interesting to note that the actions mentioned above all have biological counterparts. Cell division (mitosis), cell locomotion, and programmed cell death (apoptosis) are processes that biological cells are capable of performing.

Movable Automata

An alternate way of viewing the EA model is from the perspective of *movable automata*. From this perspective, active EA cells contain finite state machines capable of moving, and inactive EA cells are empty space. In contrast, each cell of a CA model contains a fixed finite state machine: active cells are those automata in non-quiescent states, and inactive cells are in the quiescent state. Cellular space automata models emphasizing actions, especially movement actions, have been investigated previously, and are briefly discussed next.

The first models that included movable automata were the kinetic automaton of von Neumann [Burks70] and the CT machines of Arbib [Arbib66]. These models included finite state automata capable of movement and other actions such as joining (called fusing or welding) and dis-joining. The Movable Finite Automata (MFA) model [Goel89] attempts to model biochemical processes such as self-assembly of bacteriophages and polypeptide chain growth in protein biosynthesis. MFA automata are characterized by their ability to move and allow bond formation and disocciation. The Computational Metabolism (ComMet) class of models [Lugowski89] consists of automata called "tiles" that move about on a 2-D grid. Tiles are grouped into different species, in which tiles of the same species execute the same rules. A tile is capable of sensing and acting upon neighboring tiles. For example, two neighboring tiles may both agree to swap places. The Creatures model [Stephenson92] consists of automata occupying a 2-D cellular array. Each automata is capable of moving, producing offspring, self-destruction, and changing its rules. A distinguishing property of Creatures is that multiple automata are permitted to occupy the same cell. Also, automata cannot sense their surrounding neighbors – they can only sense other automata that are co-located. The Creatures model has been used to model ideal gases and disease transmission.

Lastly, a model of movable automata for use in simulating biochemical reactions of oligonucleotides has been reported [Chou94]. In this model, automata represent molecules and are governed by rules derived from chemical reactions. The automata are capable of movement, rotation, and bonding, which permits aggregate structures to form. Using this model, self-replicating oligonucleotide systems were simulated and found to compare favorably with laboratory experiments.

Cellular automata models are capable of simulating movement of single cells or of aggregate structures. The famous game of Life rule table [Gardner70] is one such example where multi-cell propagating structures may be seen. The concept of hierarchies of structures (called virtual state machines) forming, moving, constructing, etc., in CA models has also been investigated [Langton86].

Automata Complexity

As described in the previous chapter, Arbib created a new cellular space model with more complex cells than that of cellular automata [Arbib66]. His rationale for doing so was twofold. First, he wanted to adopt a hierarchical approach where his automata would be analogous to higher-order structures than in CA. Second, by adding complexity to each automaton, he felt that the complexity of the self-replicating structure could be greatly reduced. He was successful in designing a much simpler self-replicating structure using less cells and a more straightforward design than that of von Neumann. However, his automata each have on the order of 10^{100} states, which is significantly larger than previous models. Presumably some of this complexity is due to his requirement of universal computation and construction.

The EA model, like Arbib's, adds complexity to individual automata. However, rather than using an automaton having on the order of 10^{100} states, EA cells typically have 10–20 states, a range comparable to previous CA-based self-replicating structures. Thus on the scale of automata complexity for cellular space models, the EA model is positioned close to traditional CA models as compared to Arbib's model.

Outline

The remainder of this chapter is organized as follows. The EA model is first formally defined. Its theory is then developed, including a set of axioms that guarantee self-replication. Growth theorems are presented as well as a comparison to the standard cellular automata model.

3.1 Model Definition

The EA model is formally defined in this section. Of particular interest is the model's ability to support self-replicating structures, and the limits on the growth of such structures. Since the EA model shares many of the same properties as the CA model, much of the same terminology and definitions can be applied to both models[1]. Specifically, the notation of [Codd68] is used where appropriate, and the notation in [Lohn95] has been modified slightly for consistency. The generalized EA model is described in this section, and the specific form of it used to study self-replicating structures is described in section 4.1.

[1]The CA model is described in Section 2.1 on page 15.

3.1.1 Cellular Space

The EA model is a spatially-distributed, deterministic dynamical system which iterates in discrete time. Space is an infinite, isotropic N-dimensional lattice of cells. Cells are either empty or occupied by at most one automaton. The position of a cell with respect to an arbitrarily chosen origin is denoted α. Automata are finite control automata capable of executing one action from a set of actions A at each time-step. Automata receive input from a fixed set of local cells called the neighborhood. The set A allows any finite number of designer-specified actions, provided that each affects only neighborhood-local cells as a result of its execution.

Individual automata are classified according to their *component* type, $\hat{v}$, a notation adapted from the CA cell state v. A component represents the set of weakly symmetric cell states obtained under successive orientations of the cell. For example, in a 2-D model, an `A` component type represents the four cell states `A`, ⊳, ∀, and ⊲. The set of all component types defined for an EA model is $\hat{V} = \{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_c\}$, containing c distinct component types. The component type of a cell located at position α is denoted $\hat{v}(\alpha)$. Since the EA model also allows for strongly rotation symmetric cell states, the set $V_s = \{v_0, v_1, \ldots, v_{k_s-1}\}$ denotes such cell states, where k_s is the number of cell states with strong rotational symmetry, and cell state v_0 is distinguished as the quiescent state as is done in [Codd68]. In the same manner, V_w is the set of k_w weakly rotation symmetric cell states. As in the CA literature, k denotes the total number of cell states in the EA model (i.e., cell states with both strong and weak rotational symmetries), such that

$$k = k_s + k_w \tag{3.1}$$

In keeping with CA notation, the set V contains all k cell states ($|V| = k$) and is given by

$$V = V_s \cup V_w \tag{3.2}$$

Calculation of the number of cell states in an EA model when the number of components is known is given by

$$k = \beta c + k_s \qquad (k_s \geq 1) \tag{3.3}$$

where β represents the number of coordinate system rotations permitted in the space (for example, $\beta = 4$ for a 2-D model having 4 90° rotations). Typically $k_s = 1$ since the empty cell is equivalent to quiescent cell state in computing k. Although it is generally more useful to speak of components with respect to the EA model, it is sometimes convenient to use states instead, and so both terms may be used keeping in mind Equation 3.3. To illustrate these sets, consider an example EA model having $\beta = 4$, two component types ($c = 2$) and two strongly rotation symmetric cell states ($k_s = 2$). The automata in such a model could be written

$$\begin{array}{ll}
\hat{V} = \{\mathtt{L}, \uparrow\} & (|\hat{V}| = c = 2) \\
V_w = \{\mathtt{L}, ⌐, ⅂, ⅃, \uparrow, \rightarrow, \downarrow, \leftarrow\} & (|V_w| = k_w = 8) \\
V_s = \{\bullet, \circ\} & (|V_s| = k_s = 2) \\
V = \{\bullet, \circ, \mathtt{L}, ⌐, ⅂, ⅃, \uparrow, \rightarrow, \downarrow, \leftarrow\} & (|V| = k = 10)
\end{array}$$

where the $\circ$ is strongly rotation symmetric and $\bullet$ represents the empty/quiescent cell state.

3.1.2 Configurations

Definition 3.1 A *configuration* C is an allowable assignment of states to cells in the cellular space.

A sequence of configurations (sometime called a *simulation* or *propagation*) is generated as the space iterates over time:

$$C_0, C_1, \ldots, C_t, \ldots \tag{3.4}$$

with C_0 denoting the *initial* or *seed* configuration. The set of all non-empty cells in a configuration C is known as the *support*, denoted $\sup C$, and is defined as

$$\sup C = \{\alpha \in C \mid v(\alpha) \neq v_0\} \tag{3.5}$$

Two configurations C and C' are *disjoint* if

$$\sup C \cap \sup C' = \emptyset \tag{3.6}$$

C' is a *subconfiguration* of C if $\sup C \cap \sup C' = \sup C'$. The number of components of type $\hat{v}$ at time t in configuration C_t is called the *multiplicity* of $\hat{v}$, and is denoted $M^t_{\hat{v}}$. Summing multiplicities over all c component types, the total number of component-occupied cells is

$$|\sup C_t| = \sum_{\hat{v} \in \hat{V}} M^t_{\hat{v}} \tag{3.7}$$

Calculation of the multiplicity $M^t_{\hat{v}}$ plays a critical role in deriving the genetic algorithm fitness functions discussed in the next chapter.

Definition 3.2 A configuration S is a *structure* if the following are satisfied:

1. All cells in S are non-quiescent, i.e.

$$S = \sup S \tag{3.8}$$

2. It is possible to reach any cell in $\sup S$ from any other cell in $\sup S$ by traversing neighborhood-adjacent $\sup S$ cells.

In this manner a structure is seen as a set of contiguous non-empty cells. Figure 3.2 shows four examples illustrating this definition. If the initial configuration is a structure, then it is called a *seed structure*, S_0. A given structure at time t, S_t, may not retain the properties of a structure at a later time t', however it may be desirable to associate its original cells, which now form a subconfiguration, with S_t. Such a subconfiguration is called a *metamorphosing structure* and is denoted by $\tilde{S}_t$.

Definition 3.3 A metamorphosing structure $\tilde{S}_t$ is a set of cell states that forms a structure S_t at time t, potentially changes shape from t+1 through t', and is identifiable as the original structure at t'+1, i.e. $S_t = S_{t'+1}$.

This definition is used in defining a self-replicating structure in Section 4.3, and is useful since many structures may temporarily change size or shape while moving or evolving, and re-form the original structure at a later time.

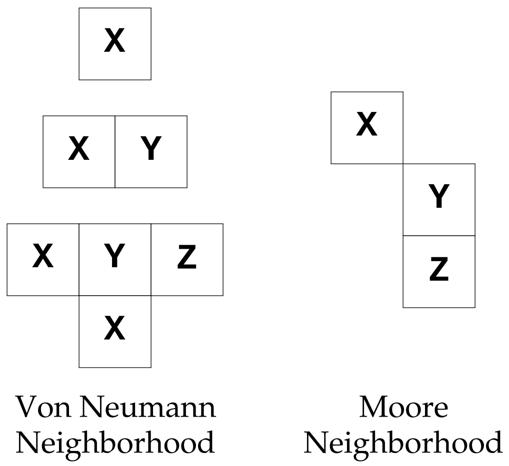


Figure 3.2: Examples illustrating the definition of *structure*: three structures in the Von Neumann neighborhood and one structure in the Moore neighborhood.

3.1.3 Rules

Each automaton in the EA model is governed by a rule table δ which induces a mapping of neighborhood states onto itself. Each entry in δ corresponds to a condition-action rule of the following format:

$$\textit{neighborhood pattern} \rightarrow \textit{action}$$

The set of actions A is comprised of any set of instructions which modify the local neighborhood upon the next time step. The `NULL` action, which does not change any cells, may also be included in A. As an example, A may contain actions to move, rotate, duplicate, and/or create new automata. The number of distinct actions in A is computed as follows for a k-state, n-neighbor EA. Since each cell may be occupied by one of k cell states, the number of possible actions is

$$|A|_{\max} = k^n \tag{3.9}$$

Equation 3.9 gives an upper bound on the number of allowed actions. In practice, however, the set of actions is typically far less than the upper bound. For example, in the main EA model studied in this thesis, $|A| = 210$, whereas the upper bound for this model is $|A|_{\max} \simeq 400,000$.

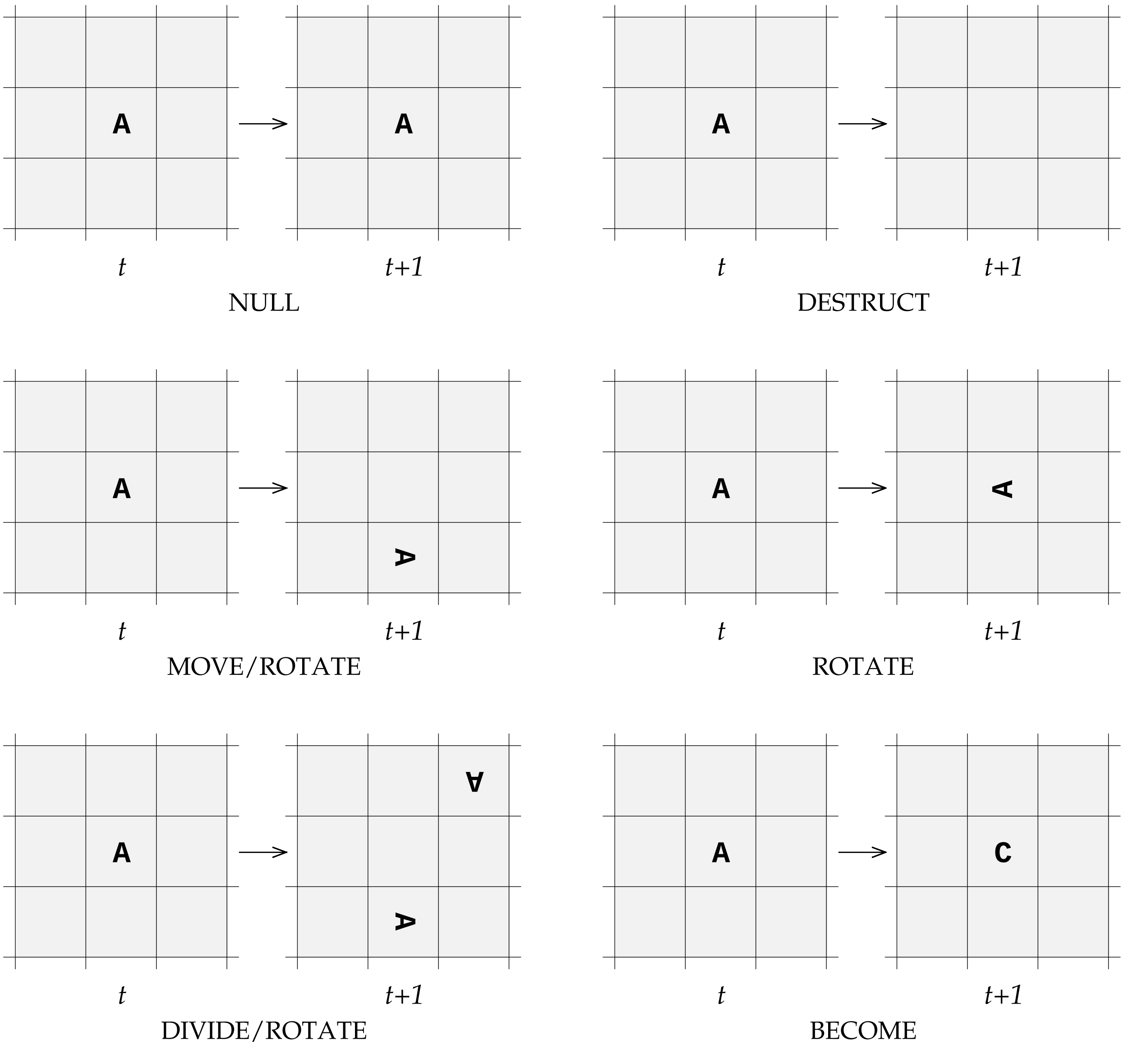


Figure 3.3: Examples of six actions in a 2-D EA model using the 9-cell Moore neighborhood.

To give an overview of the range of actions that A could contain, Figure 3.3 contains examples of six representative actions. The names of the actions appear below each diagram, and in some case directional parameters are implied. For example, the `MOVE` action requires a relative direction parameter specifying the direction in which to move. Also note that the `ROTATE` action is combined in some cases for convenience (actions may be composite actions the EA model definition).

3.1.4 Cell Contention Resolution

Because automata actions can modify neighboring cells, the situation arises in which more than one automata may attempt to co-locate at the same cell, causing contention for that cell. As illustrated in Figure 3.4, components `A` and `B` are separated by one cell and arrows indicate that both have rules directing them to move into the same cell. Since the EA model, like the CA, prohibits cells from having more than one automata, a *cell contention policy* is required. An example of such a policy is *mutual annihilation* (as termed in [Codd68]) which results in all automata moving into the same cell being destroyed. Another policy could be defined in which one of the contentious automata is randomly selected to occupy the cell in question. Biases toward certain component types could also be enforced.

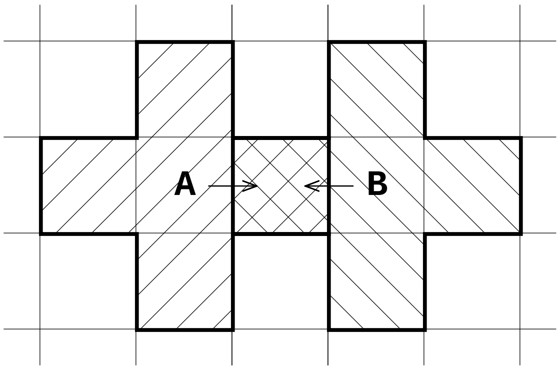


Figure 3.4: Cell contention occurs when two automata whose neighborhoods overlap attempt to occupy the same cell in a 2-D 5-neighbor EA model.

A cell contention policy is a global property of the space similar to other properties such as fixed propagation velocity and the limit on the actions allowed. Collectively, such properties are analogous to physical laws of nature. The important point is that although these properties are global, they are *static*, and thus the dynamics of the model are based solely on local interactions. In contrast, some previous models of movable automata [Goel89], have relied on dynamic globally-available information, which violates having strictly local interactions.

3.1.5 Summary of EA Notation

A summary of the notation used in the definition of the EA model is shown in Table 3.1. For clarity, CA and EA designations are parenthetically noted when symbols are primarily used in a particular model.

Symbol	*Description*
A	set of actions (EA)
α	position of cell
β	number of coordinate system rotations
n	neighborhood size
$v, \hat{v}$	state (CA), component type (EA)
V	set of cell states $\{v_1, v_2, \ldots v_k\}$
$\hat{V}$	set of component types $\{\hat{v}_1, \hat{v}_2, \ldots \hat{v}_c\}$
k	number of cell states (CA)
c	number of component types (EA)
k_s	number of strongly rotation symmetric cell states
k_w	number of weakly rotation symmetric cell states
C_t	configuration at time t
$M_{\hat{v}}^t$	multiplicity of component $\hat{v}$ at time t
S_t	structure at time t
S_0	seed structure
$\tilde{S}_t$	metamorphosing structure
δ	rule table function
$\|\delta\|$	number of entries in rule table

Table 3.1: Summary of EA model notation.

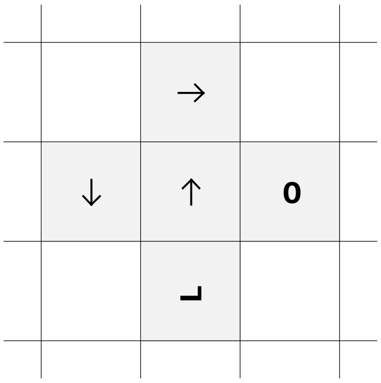
3.2 Component-Sensitive Input

A rule table compression method which has not been studied in the cellular space modeling literature is the technique of *component sensitivity*. For models that incorporate weakly rotation-symmetric cell-states, it is possible to simplify the rule table function by modifying the way automata receive input.

In cellular space models to date, an automaton is sensitive to the *states* of its neighboring cells, and uses this input to make a transition. This method of cell input is called *state-sensitive input* (SSI). An alternative input technique is one in which an automaton receives only *component* information from the cells in its local neighborhood. This is called *component-sensitive input* (CSI). In SSI, the center cell senses both the component type *and* orientation of cell states in its neighborhood, whereas in CSI, the center cell senses only the component types. Figure 3.5 shows an example of a cell's input patterns under both SSI and CSI. There it is seen that the cell-state $\uparrow$ senses an `L` component having a $-90°$ orientation below it (SSI case), and an `L` component without orientation (CSI case).

Rule tables are reduced under component-sensitive input since there are far fewer permutations of neighborhood patterns. This is a significant advantage for reasons of increased computational tractability and decreased search space sizes. In section 3.3.2 expressions for rule table sizes using

Cell-states: {•, ∘, L, Γ, ⅂, ⅃, ↑, →, ↓, ←}
Components: {L, ↑}



Input pattern under SSI: → ∘⅃↓

Input pattern under CSI: ↑ ∘L↑

Figure 3.5: Example illustrating automata input sensitivity.

both CSI and SSI will be derived and compared. There it will be shown that SSI rule tables are β^{n-1} larger than the equivalent CSI rule tables.

It should be remembered that component-sensitive input is only possible in cellular space models having weak rotational symmetry. Since the EA model is defined to have weak rotational symmetry, CSI may be used with any EA model. Cellular automata models with weak rotational symmetry may also be specified using this input method.

3.3 Comparison of CA and EA Models

In this thesis, both cellular automata and effector automata models are studied in the context of supporting self-replicating structures. In this section certain comparisons are made between the two models which are required to understand later results.

3.3.1 Model Equivalence

In comparing the EA and CA models to each other, it is useful to ask whether one model is more general and can simulate the behavior of the other, and under what conditions this can occur. In the generalized EA model, automata are afforded a wide range of complexity and thus it is not surprising that an EA model can be derived to simulate the behavior of any weakly rotation-symmetric CA. Conversely, EA behavior can be designed into a CA provided that the complexity of the transition function is increased sufficiently. These points are made in the following theorems.

Theorem 3.1 An effector automata model with a single `BECOME` action can simulate the behavior of any weakly rotation-symmetric n-neighbor cellular automata model using an n-neighbor effector automata model having a rule table of equal size.

Proof: By inclusion of a `BECOME` action, the condition-action rule of an EA is equivalent to a state transition of a finite state machine. The `BECOME` action changes the automaton's current state to any state in V. In a CA, the state transition rules are of the format: CTRBL $\rightarrow$ C$'$. If each C$'$ is replaced by the `BECOME` C$'$ action, an equivalent transition rule of is formed in the EA model. Since empty EA cells do not contain automata, CA quiescent cells are simulated by inclusion of a cell-state having strong rotational symmetry. □

Before presenting the next theorem, some background on neighborhood functions [Codd68] is necessary. The function $g(\alpha)$ generates the set of cells comprising the neighborhood of cell α

$$g(\alpha) = \{\alpha, \alpha + \zeta_1, \ldots, \alpha + \zeta_{n-1}\} \tag{3.10}$$

where $\zeta_i (i = 1, .., n-1)$ are coordinates relative to α and n is the neighborhood size as defined previously. As an example, the von Neumann neighborhood is expressed as

$$g(\alpha) = \{\alpha, \alpha + (1,0), \alpha + (-1,0), \alpha + (0,1), \alpha + (0,-1)\} \tag{3.11}$$

which generates the set of five cells: center, top, left, bottom, right.

In comparing the CA and EA models to each other, the concept of a *second-order* neighborhood function is required. The second-order neighborhood of a cell α is the set of cells comprising the neighborhood of α as well as the cells in those neighboring cells' neighborhood. This is expressed as

$$g'(\alpha) = g(g(\alpha)|_1) \cup g(g(\alpha)|_2) \cup \cdots \cup g(g(\alpha)|_n) \tag{3.12}$$

where $g(\alpha)|_i$ denotes the position of the ith cell of $g(\alpha)$. Figure 3.6 illustrates how the second-order neighborhood is obtained from the von Neumann neighborhood.

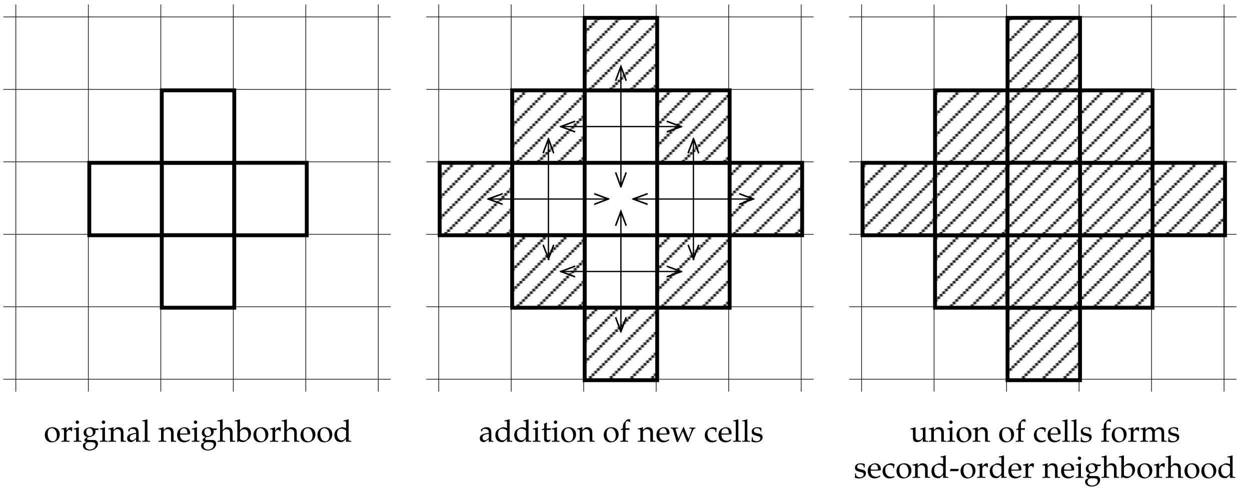
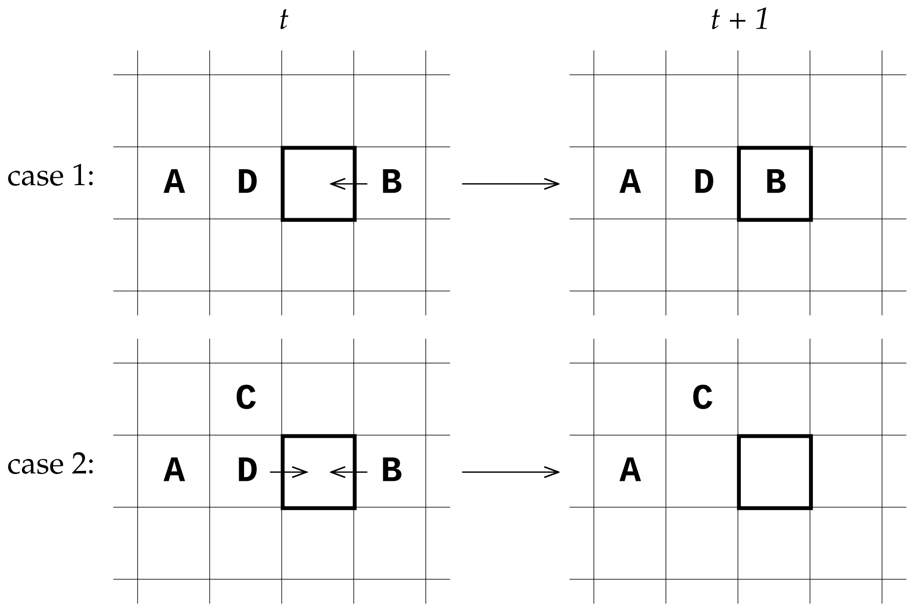


Figure 3.6: Obtaining a second-order neighborhood from the von Neumann neighborhood.

The second-order neighborhood function is required because, from a CA perspective, automata located in a second-order neighborhood cell can affect the automata located in the center cell of the

EA. In addition to actions, the cell contention policy (section 3.1.4) of the EA model can affect the contents of a cell at each time step. For example, consider the cell shown highlighted in Figure 3.7. In case 1, a `B` component moves left as governed by rule 2 and occupies the highlighted cell. The other components execute `NULL` actions. In case 2, a `C` component is added which changes the `D` component's behavior (it now uses rule 5 instead of rule 4). Because `D` and `B` attempt to occupy the same cell, causing cell contention, they are both removed under the policy of mutual annihilation. Thus the highlighted cell remains empty in this case, in contrast to the first case. This simple example underscores the influence of components located in the second-order neighborhood.



Partial Rule Table

1. `A•D••` $\rightarrow$ `NULL`
2. `B••••` $\rightarrow$ `MV LEFT`
3. `C••D•` $\rightarrow$ `NULL`
4. `D•••A` $\rightarrow$ `NULL`
5. `DC••A` $\rightarrow$ `MV RIGHT`

Figure 3.7: Example behavior of the EA model illustrates how the addition of a `C` component in case 2 influences the contents of the highlighted cell.

Theorem 3.2 In 2-D square tessellations, a cellular automata model with weak rotational symmetry can simulate the behavior of a 2-D n-neighbor effector automata using a neighborhood of size $n' \geq 2n - 1$ for integers $n > 0$.

Proof: The next state $v(\alpha)$ of an EA cell α is influenced by the cells in its second-order neighborhood, $g'(\alpha)$. The number of cells generated by $g'(\alpha)$ is $|g'(\alpha)|$, which is the size of the CA neighborhood n'. The value n' varies depending on the EA neighborhood size n and pattern of the EA neighborhood as follows:

$$|g'(\alpha)| = \begin{cases} 2n-1 & \text{pattern } p_1 \\ 3n-3 & \text{pattern } p_2 \\ \vdots & \vdots \\ (z+1)n - z(z-1) - 1 & \text{pattern } p_z \end{cases} \tag{3.13}$$

where z is a positive integer which enumerates different neighborhood patterns, and Equation 3.13 is subject to the condition $|g'(\alpha)| > |g(\alpha)|$ (this states that the second-order neighborhood may not be smaller than the original neighborhood). From the set of functions generated by $(z+1)n-z(z-1)-1$, the function that is bounded by all others (subject to the restrictions above), and hence minimal, occurs at z=1 where the neighborhood size is $2n-1$. The pattern p_1 corresponds to the linear contiguous set of cells, examples of which are shown in Figure 3.8. Thus the lower bound on CA neighborhood size is $2n-1$. □

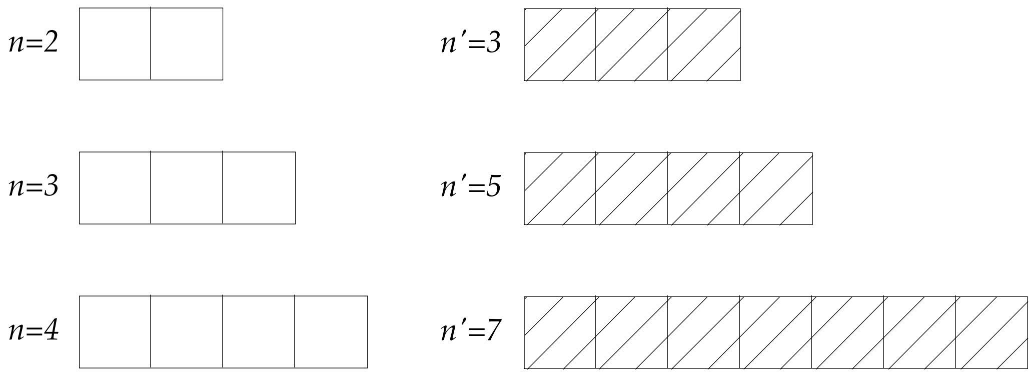


Figure 3.8: Examples of linear neighborhood patterns for various n, corresponding to p_1 in Equation 3.13. All patterns which have minimal second-order neighborhood sizes $|g'(\alpha)|$. Second-order neighborhood patterns are shown hatched.

The rule table sizes of comparable n-neighbor k-state EA and CA models are both $O(k^n)$. Applying Theorem 3.2 yields the ratio

$$\frac{O(|\delta|_{CA})}{O(|\delta|_{EA})} = \frac{O(k^{n'})}{O(k^n)} = \frac{O(k^{2n-1})}{O(k^n)} \simeq k^n \tag{3.14}$$

which implies that the rule table of a CA must be approximately k^n times larger than that of the EA, demonstrating a significant increase in CA complexity is required.

3.3.2 CA Rule Tables and Search Spaces

The *rule table size* of a CA, $|\delta|$, is the number of individual state transitions in the rule table function, and plays an important role in this thesis. A k-state, n-neighbor non-isotropic CA will have a rule table containing $|\delta| = k^n$ state transition rules, corresponding to the the number of length-n sequences of k unique objects, when each may be repeated any number of times. The set of all possible rule tables for a CA is denoted D_n^k. Since there are k possibilities for the next state in each transition in a rule table, the number of possible rule tables is:

$$|D_n^k| = k^{|\delta|} \tag{3.15}$$

Equation 3.15 is an expression for the size of the CA *search space* when trying to learn a rule table, and is an important parameter when genetic algorithms are applied as a search technique. A search space is a collection of candidate solutions to a given problem. In the context of designing a CA to exhibit a certain behavior, D_n^k represents all possible candidate solutions and hence comprises the entire search space. A related term, fitness landscape, refers to the quality of each of the candidate solutions, where better performing solutions correspond to higher points. As an example, in a 6-state, 5-neighbor non-isotropic CA, there are $|\delta| = 6^5 = 7776$ rules and $|D_5^6| = 6^{7776} \simeq 10^{6050}$ possible rule tables, an extremely large number. The size of this search space indicates it would be impossible to exhaustively explore the space of all such D_5^6 CAs.

As mentioned in Section 2.1, a given CA rule table can have weak or strong rotational symmetry when the underlying space is isotropic. For isotropic spaces, $|D_n^k|$ becomes significantly smaller as compared to that of non-isotropic spaces. A reduction in rule table size occurs because redundant transition rules may be removed due to symmetry conditions. This results from the removal of redundant permutations. Under strong and weak rotational symmetries, in an n-neighbor CA only $n-1$ positions are relevant since the center cell has no defined symmetry relative to itself. Thus for cell states with weak rotational symmetry, distinct permutations are counted using k^{n-1}. For strongly rotation symmetric cell states *circular* permutations are used to count distinct neighborhood patterns, denoted ${}_k\mathrm{CP}_{n-1}$. Recall that k_s and k_w represent the number of strongly and weakly rotation symmetric cell states ($k = k_s + k_w$). Let $|\delta_s|$ be the number of strongly symmetric transition rules, and $|\delta_w|$ be the number of weakly symmetric transition rules. Then,

$$|\delta_s| = k_s \cdot {}_k\mathrm{CP}_{n-1} \tag{3.16}$$

and

$$|\delta_w| = \frac{k - k_s}{\beta} \cdot k^{n-1} \tag{3.17}$$

where β represents the number of coordinate system rotations permitted in the space (for example, $\beta = 4$ for a 2-D model having 4 90° rotations). For a CA model to have strong rotational symmetry, all k states must have strong rotational symmetry. Thus $k = k_s$, and the total rule table size is:

$$|\delta| = k \cdot {}_k\mathrm{CP}_{n-1} \tag{3.18}$$

In a weakly rotation symmetric model, at least one state (the quiescent state) has strong rotational symmetry. With $k = k_w + k_s$ and $k_s \geq 1$, then the total rule table size is:

$$\begin{aligned} |\delta| &= |\delta_s| + |\delta_w| \\ &= (k_s \cdot {}_k\mathrm{CP}_{n-1}) + \left(\frac{k - k_s}{\beta} \cdot k^{n-1}\right) \end{aligned} \tag{3.19}$$

Combining equations 3.15, 3.18, and 3.19 yields expressions for the search space size under both weak and strong rotational symmetries:

$$|D_n^k| = k^{(k_s \cdot {}_k\mathrm{CP}_{n-1}) + \left(\frac{k-k_s}{\beta} \cdot k^{n-1}\right)} \qquad \text{(weak rot. symm.)} \tag{3.20}$$

$$|D_n^k| = k^{(k \cdot {}_k\mathrm{CP}_{n-1})} \qquad \text{(strong rot. symm.)} \tag{3.21}$$

The calculation of circular permutations involves more advanced combinatorics and is outlined in Appendix A. Comparing circular permutations to k^n as a function of k analytically is difficult due to the complex nature of the circular permutation function. However, for small values of k, these functions can be compared empirically. Figure 3.9 shows curves for both functions when an $n = 5$ neighborhood size is assumed. By doing a simple regression, it is found that these functions differ by a factor of approximately four for a given value of k. This agrees with intuition since a strongly symmetric transition rule is "rotated four times" for each transition rule in the non-isotropic model. Also this implies that rule table sizes for an isotropic CA with strong rotational symmetry will be approximately four times smaller than that of a non-isotropic CA:

$$|\delta|_{\text{non-iso}} \simeq 4 \cdot |\delta|_{\text{strong}} \tag{3.22}$$

Substituting Equation 3.22 into Equation 3.15, we can determine the relationship between these two search space sizes as follows:

$$\begin{aligned} |D_n^k|_{\text{non-iso}} &= k^{(|\delta|_{\text{non-iso}})} \\ &\simeq k^{4(|\delta|_{\text{strong}})} \\ &\simeq (|D_n^k|_{\text{strong}})^4 \end{aligned} \tag{3.23}$$

Equation 3.23 shows that search spaces for CA models with strong rotational symmetry are roughly four orders of magnitude smaller than comparable non-isotropic spaces.

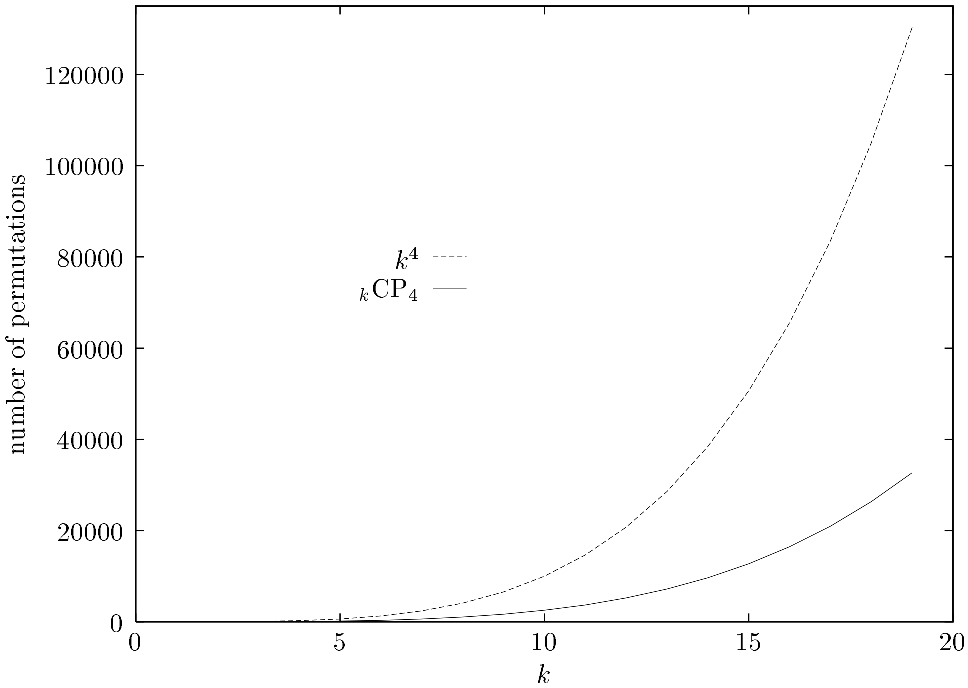


Figure 3.9: Number of permutations as a function of k for an $n = 5$ neighborhood.

As an example, the parity rule table in Table 2.1 (page 16) contains all 2^5 transition rules since it assumes a non-isotropic space. If an isotropic space is assumed, each finite state automaton in each cell is strongly rotation symmetric. Since ${}_2\mathrm{CP}_4 = 6$, $|\delta| = 12$, as compared to 32 for the original table. The reduced rule table is shown in Table 3.2, where $C\pi_c$(TRBL) is used to denote the circular permutations of the four neighborhood positions.

$C\pi_c$(TRBL)	C′	$C\pi_c$(TRBL)	C′
00000	0	10000	1
00001	1	10001	0
00011	0	10011	1
00101	0	10101	1
00111	1	10111	0
01111	0	11111	1

Table 3.2: Reduced rule table for the parity function when assuming strong rotational symmetry for each of the two cell states.

Table 3.3 shows computed values for $|D_n^k|$ for small numbers of states under different symmetries. For k=2, 4096 rule tables are possible with strong symmetry (the parity table above is one such table). It can be seen that the rule table space is reduced many orders of magnitude when isotropic spaces are used. However, for $k > 2$, $|D_n^k|$ values are nonetheless astronomically large. Values for rule table sizes $|\delta|$ are the exponent for the base k in this table. For k=5 it is seen that an isotropic space can reduce rule table size by a factor of four ($3125/825 \simeq 4$) which was borne out by Equation 3.23. This will later become an important factor from a computational perspective when evolving such models using a genetic algorithm. Also note that five states are needed for weak symmetry since one state is quiescent and four states comprise a single rotated component.

	$\lvert D_n^k \rvert$			
k	Non-isotropic	Strong Rot. Symm.	Weak Rot. Symm.	(c)
2	$2^{32} \simeq 10^{10}$	$2^{12} = 4096$	–	–
3	$3^{243} \simeq 10^{116}$	$3^{72} \simeq 10^{34}$	–	–
4	$4^{1024} \simeq 10^{617}$	$4^{280} \simeq 10^{169}$	–	–
5	$5^{3125} \simeq 10^{2184}$	$5^{825} \simeq 10^{577}$	$5^{790} \simeq 10^{552}$	(1)
6	$6^{7776} \simeq 10^{6051}$	$6^{2016} \simeq 10^{1569}$	$6^{1968} \simeq 10^{1531}$	(1)
7	$7^{16807} \simeq 10^{14203}$	$7^{4312} \simeq 10^{3644}$	$7^{4249} \simeq 10^{3591}$	(1)
8	$8^{32768} \simeq 10^{29592}$	$8^{8352} \simeq 10^{7543}$	$8^{8272} \simeq 10^{7470}$	(1)
9	$9^{59049} \simeq 10^{56347}$	$9^{14985} \simeq 10^{14299}$	$9^{14787} \simeq 10^{14110}$	(2)

Table 3.3: Values of search space sizes $|D_n^k|$ for various k-state, n=5 neighbor cellular automata with different symmetries.

3.3.3 EA Rule Tables and Search Spaces

In this section the rule table and search space sizes are calculated for the EA model. For purposes of comparison, notation and symbols germane to CAs are used as appropriate. Analyses for both component sensitive input (CSI) and state sensitive input (SSI) EA models are given. For those models, the notations $|\delta|_{\mathrm{CSI}}$ and $|\delta|_{\mathrm{SSI}}$ are used to distinguish the rule table sizes under the different input sensitivities. The same notation applies to search space sizes as well. As with CA models, the set of all possible rule tables for a k-state, n-neighbor EA is denoted D_n^k. Since there are $|A|$ possible actions for each entry in a rule table, the number of possible rule tables under each input sensitivity is

$$
\begin{aligned}
|D_n^k|_{\mathrm{CSI}} &= |A|^{|\delta|_{\mathrm{CSI}}} && (3.24)\\
|D_n^k|_{\mathrm{SSI}} &= |A|^{|\delta|_{\mathrm{SSI}}} && (3.25)
\end{aligned}
$$

3.3.3.1 EA Rule Tables Under CSI

As defined in section 3.1, EA models have weak rotational symmetry. This condition does not exclude having strongly rotation symmetric cell states in an EA model. Rather it means that at least one weakly rotation symmetric cell state must be present. With $k = k_s + c\beta$ these conditions imply that $c > 0$, $k_s > 0$, and $k \geq 1 + \beta$, meaning that at least one component and one strongly rotation symmetric cell state (empty/quiescent cell state) are required. Since the quiescent cell state does not require any condition action rules, there are $k_s - 1$ strongly rotation symmetric cell states. Thus, the number of strongly rotation symmetric rules is

$$|\delta_s|_{\mathrm{CSI}} = (k_s - 1) \cdot \ {}_{c+k_s}\mathrm{CP}_{n-1} \tag{3.26}$$

Under component sensitive input, the number of weakly rotation symmetric rules is the number of components c times the number of possible neighborhood arrangements:

$$|\delta_w|_{\mathrm{CSI}} = c(c + k_s)^{n-1} \tag{3.27}$$

The overall rule table size, $|\delta|$, is the number of condition-action rules in the rule table function. Combining equations 3.26, 3.27, and $|\delta| = |\delta_s| + |\delta_w|$,

$$|\delta|_{\mathrm{CSI}} = (k_s - 1) \cdot \ {}_{c+k_s}\mathrm{CP}_{n-1} + c(c + k_s)^{n-1} \tag{3.28}$$

From 3.24, the search space size in the EA model under CSI is thus expressed

$$|D_n^k|_{\mathrm{CSI}} = |A|^{(k_s - 1)\cdot \ {}_{c+k_s}\mathrm{CP}_{n-1} + c(c+k_s)^{n-1}} \tag{3.29}$$

Is is clear from Equation 3.29, that search space sizes are very sensitive to the neighborhood size n and number of components c. Table 3.4 lists rule space sizes for small values of c for various 2-D EA models having one strongly rotation symmetric state, $\beta = 4$, and using the von Neumann neighborhood.

3.3.3.2 EA Rule Tables Under SSI

Using state-sensitive input, the derivation of the rule table size is similar to that of component sensitive input. For SSI, the permutations for the neighborhood patterns are for k states as opposed to $c + k_s$ in CSI. Thus the $c + k_s$ terms in equations 3.26 through 3.29 are replaced by k to give

$$
\begin{aligned}
|\delta_s|_{\mathrm{SSI}} &= (k_s - 1) \cdot \ {}_k\mathrm{CP}_{n-1} && (3.30)\\
|\delta_w|_{\mathrm{SSI}} &= ck^{n-1} && (3.31)\\
|\delta|_{\mathrm{SSI}} &= (k_s - 1) \cdot \ {}_k\mathrm{CP}_{n-1} + ck^{n-1} && (3.32)\\
|D_n^k|_{\mathrm{SSI}} &= |A|^{(k_s-1)\cdot \ {}_k\mathrm{CP}_{n-1} + ck^{n-1}} && (3.33)
\end{aligned}
$$

c	k	$\lvert D_n^k\rvert_{\mathrm{CSI}}$
1	5	$\lvert A\rvert^{16}$
2	9	$\lvert A\rvert^{162}$
3	13	$\lvert A\rvert^{768}$
4	17	$\lvert A\rvert^{2500}$
5	21	$\lvert A\rvert^{6480}$

Table 3.4: Values of search space sizes $|D_n^k|_{\mathrm{CSI}}$ for various k-state, n=5 neighbor effector automata with $k_s = 1$ and $\beta = 4$.

Equation 3.33 again shows the search space size to be very sensitive to neighborhood size n and number of states k. Table 3.5 lists rule space sizes for small values of c for various 2-D EA models having one strongly rotation symmetric state, $\beta = 4$, and using the von Neumann neighborhood.

c	k	$\lvert D_n^k\rvert_{\mathrm{SSI}}$
1	5	$\lvert A\rvert^{625}$
2	9	$\lvert A\rvert^{13122}$
3	13	$\lvert A\rvert^{85683}$
4	17	$\lvert A\rvert^{334084}$
5	21	$\lvert A\rvert^{972405}$

Table 3.5: Values of search space sizes $|D_n^k|_{\mathrm{SSI}}$ for various k-state, n=5 neighbor effector automata with $k_s = 1$ and $\beta = 4$.

3.3.4 Effect of Input Sensitivity on EA and CA Models

In this section a comparison is made of rule table and search space sizes under different input sensitivities in the EA and CA models. Using the expressions for rule table sizes under both CSI and SSI, their magnitudes can be compared as follows. First assume there is only one strongly rotation symmetric state so that $k_s = 1$. This is a reasonable assumption since it is common for weakly rotation symmetric models to have only one strongly symmetric state which represents the empty/quiescent cell state. The ratio for the rule table sizes is

$$\frac{|\delta|_{\mathrm{SSI}}}{|\delta|_{\mathrm{CSI}}} \tag{3.34}$$

For cellular automata this ratio is

$$\frac{{}_{\beta c+1}\mathrm{CP}_{n-1} + c(\beta c+1)^{n-1}}{{}_{c+1}\mathrm{CP}_{n-1} + c(c+1)^{n-1}}$$

and for effector automata the ratio is

$$\frac{c(\beta c+1)^{n-1}}{c(c+1)^{n-1}}$$

Both of these ratios converge to the same constant as the number of components c is increased. The circular permutation functions in the CA ratio are insignificant compared to the other terms as c increases. Thus in the limit we have

$$\lim_{c\to\infty} \frac{{}_{\beta c+1}\mathrm{CP}_{n-1} + c(\beta c+1)^{n-1}}{{}_{c+1}\mathrm{CP}_{n-1} + c(c+1)^{n-1}} = \lim_{c\to\infty} \frac{c(\beta c+1)^{n-1}}{c(c+1)^{n-1}} = \beta^{n-1} \tag{3.35}$$

Equation 3.35 states that as c increases, models (EA or CA) using component sensitive input have rule tables that are approximately β^{n-1} smaller than models (EA or CA) using state sensitive input. For a typical coordinate system with $\beta = 4$, and using the von Neumann and Moore neighborhoods, it is seen that the $|\delta|$ values will differ by a factors of 256 and 65536, respectively, as the number of components increases. This multiplicative increase translates into orders of magnitude increases in search space sizes. From Equation 3.25 which expresses the EA rule table size:

$$\begin{aligned} |D_n^k|_{\mathrm{SSI}} &= |A|^{(|\delta|_{\mathrm{SSI}})} \\ &\simeq |A|^{\beta^{n-1}(|\delta|_{\mathrm{CSI}})} \\ &\simeq (|D_n^k|_{\mathrm{CSI}})^{\beta^{n-1}} \end{aligned} \tag{3.36}$$

From Equation 3.36 it can be seen that by using component sensitive input, the search space is decreased approximately β^{n-1} orders of magnitude. As an example, the models used in this work have $\beta = 4$ and $n = 5$, giving a difference of 256 orders of magnitude.

3.3.5 Summary of Rule Table Sizes

A summary of expressions for rule table sizes is shown in Table 3.6. An entry of "–" denotes "not applicable". Figure 3.10 shows these functions graphically for small values of k, with Figure 3.11 showing the lower portions of the curves in detail. The curves for models using CSI have a sawtooth-like appearance because $\beta = 4$: every fourth k the curve diminishes since four cell states are converted into a single component. For example, at $k = 12$, $k_s = 3$ and $c = 2$, but at the next k ($k = 13$), $k_s = 1$ and $c = 3$. It is clear from these curves that using component sensitive input significantly reduces the rule table sizes as compared to the other model parameters.

3.4 Growth of Self-Replicating Structures

In this section we analytically investigate the growth of self-replicating structures in 2-D, 5-neighbor EA models. It appears reasonable to assume that a self-replicating structure could produce a population of replicants that grow exponentially with time (for example, the population might double in size every generation). However, as the following theorem indicates, this cannot occur. A similar conclusion is reached in [Moore62] for specific 2-D cellular automata models.

Theorem 3.3 If a self-replicating seed structure S_0 is capable of producing $\gamma(t)$ replicants by time t, then there exists a constant $K > 0$ such that $\gamma(t) \leq Kt^2$.

		CA	EA
State Sensitive Input	Non-isotropic	k^n	–
	Strong Rot. Symm.	$k \cdot {}_k\mathrm{CP}_{n-1}$	–
	Weak Rot. Symm.	$k_s \cdot {}_k\mathrm{CP}_{n-1} + ck^{n-1}$	$(k_s - 1) \cdot {}_k\mathrm{CP}_{n-1} + ck^{n-1}$
Component Sensitive Input	Non-isotropic	–	–
	Strong Rot. Symm.	–	–
	Weak Rot. Symm.	$k_s \cdot {}_{c+k_s}\mathrm{CP}_{n-1} + c(c+k_s)^{n-1}$	$(k_s - 1) \cdot {}_{c+k_s}\mathrm{CP}_{n-1} + c(c+k_s)^{n-1}$

Table 3.6: Summary of rule table sizes $|\delta|$ for n-neighbor CA and EA models under different rotational symmetries. "–" denotes "not applicable".

Proof: Let the smallest rectangle enclosing S_0 be of dimensions $l \times w$. Then at each time t, the largest number of non-empty cells in the configuration C_t is given by

$$|\sup C_t|_{\max} = lw + 2lt + 2wt + 2(t^2 - t)$$

Dividing by the size of each replicant, the number of replicants at time t is at most

$$\frac{lw + 2lt + 2wt + 2(t^2 - t)}{lw}$$

which is $O(t^2)$. Therefore $\gamma(t)$ is bounded by Kt^2. □

The limit on the population size results from the finite "velocity" in which automata may propagate into the empty region of space: using the von Neumann neighborhood, EA automata may move one cell at each time step. This restriction has been called analogous to the physical limitation imposed by the speed of light.

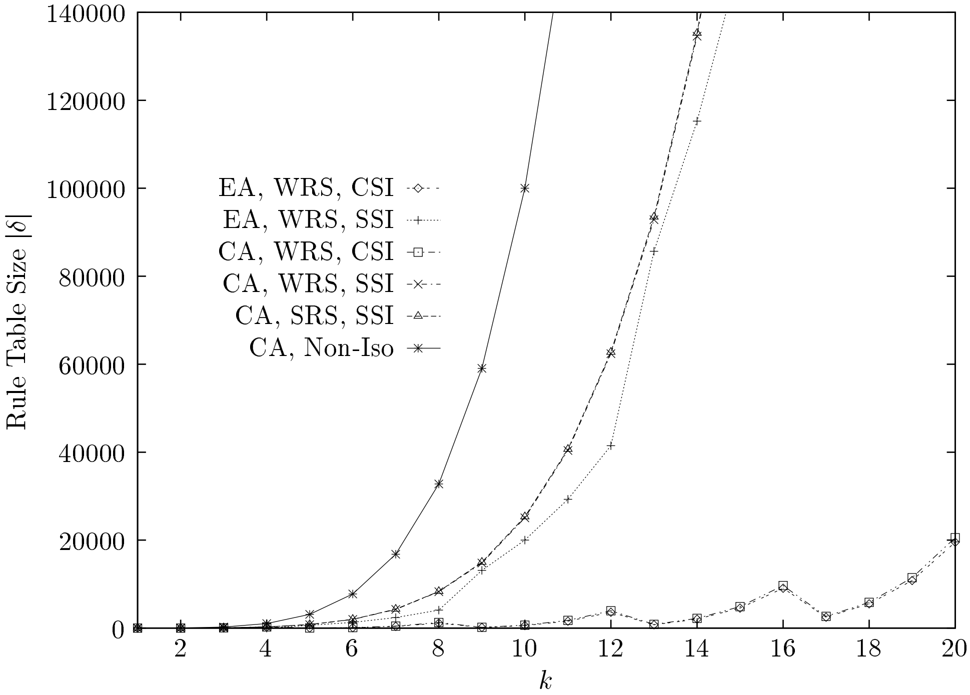


Figure 3.10: Rule Table Size $|\delta|$ as a function of k for various n=5 neighborhood EA and CA models.

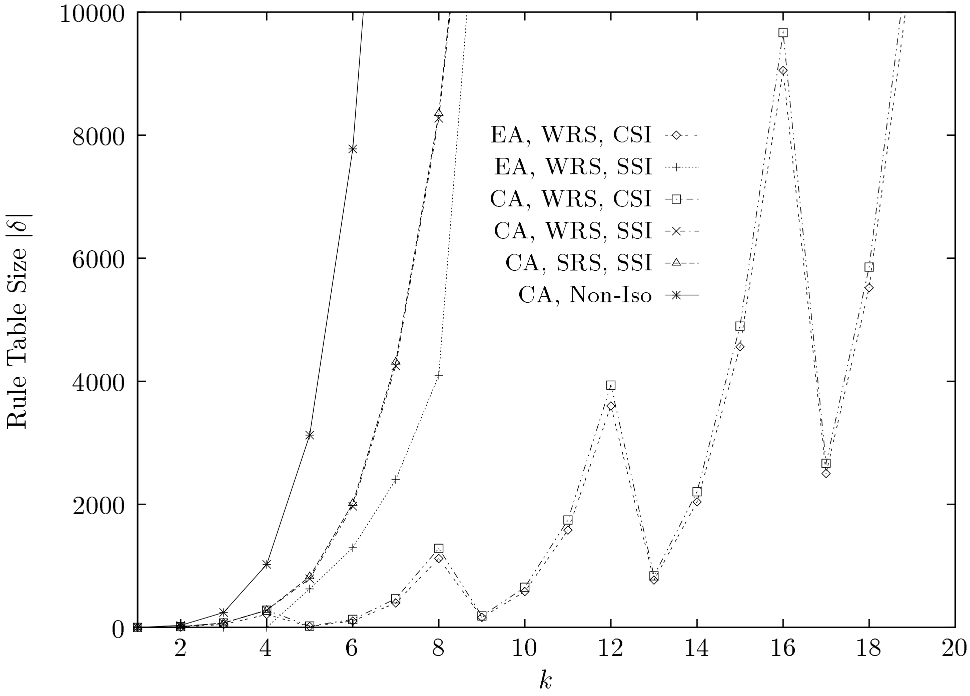


Figure 3.11: Graph of Figure 3.10 with smaller $|\delta|$ values to show lower portions of curves in greater detail.

Chapter 4

Designing GAs for Automatic Discovery of Self-Replicating Structures

Over the decades since von Neumann first demonstrated that structures in cellular automata can self-replicate [von Neumann66], a substantial body of theoretical and modeling studies have led to progressively simpler and smaller models [Codd68, Langton84, Reggia93]. However, all such past models have been manually designed, a process that is very difficult and time-consuming, and is prone to subjective biases of the implementor. With ever smaller hand-designed self-replicating structures being reported, and ever increasing computational resources available, the hypothesis that it would be possible to generate self-replicating structures *automatically* appeared to be testable. Since this problem had never been attempted, it was of great interest to show that the automatic discovery of self-replicating structures was even possible.

As noted in Section 2.3.3 of Chapter 2, relatively few studies have reported using genetic algorithms to automatically produce rule tables for cellular space automata models. However, until [Lohn95] there were no reports of using GAs to produce cellular space automata models for self-replicating structures, and self-replicating structures were the very subject cellular automata were first invented to study. Such research was most likely not undertaken for at least two reasons. Firstly, the computational load can become enormous. As shown in Chapter 3, the rule tables for modest CA systems can quickly grow extremely large (e.g., 25,000 for a k=10 states strongly rotation symmetric CA), and manipulating numerous such large "chromosomes" in a GA can quickly exhaust the memory capacity and processing capabilities on many computer systems. Secondly, and most importantly, identification of effective fitness functions is a difficult task. Apparently obvious fitness functions such as those that count the number of replicants are useless early on as there will typically be none. In general, assigning small values of fitness to behaviors that do not resemble self-replication yet have potential to evolve into such a process is a very difficult problem. The solution to this problem is one of the key contributions of this chapter.

The novel fitness functions reported in this chapter are general in three senses: they may be applied to a large number of 2-D cellular space automata models, any size and shape seed structure containing unique components may be used, and they may be used in conjunction with a variety of search techniques. Evidence of this generality is presented in Chapter 5 where the fitness functions are used in both CA and EA models, and under four search techniques. In addition, the fitness functions do not impose undue biases towards any particular process of self-replication. That is, in their definitions, the fitness functions do not assign credit based on aspects such as: the contents of specific cell locations at specific instants, whether/how the structure should translate or rotate itself over time, the quantity/timing of replicant production, or the extent to which configurations match a predefined configuration.

Since the primary search technique for the rule discovery system here is the genetic algorithm, areas specifically concerning the use of the GA are also presented. This includes the choice of genetic operators, associated parameters, penalty functions, and multiobjective optimization issues.

4.1 Models Used in Experiments

Like CA models, the range of potential EA models is vast. For the purpose of studying self-replicating structures, a small set of specific EA models are adapted from the general EA model described in Chapter 3. In selecting these models, several criteria were of great importance:

- For comparison purposes, fundamental model parameters should be kept the same as or closely parallel to previous work in hand-designed, self-replicating structures. For example, parameters such as the dimensionality of the cellular space, neighborhood size and shape, and the number of coordinate system rotations were kept the same as in several previous studies (for example [Reggia93]).
- The set of actions A should include the fundamental operations observed in previous models of self-replicating structures. An exception to this is the omission of the `BECOME` action. Because the `BECOME` action changes an automaton's component type, and thus the manner in which it behaves, it compromises the physical relevance of the automata. An analogy using biological cells is fitting. An amoeba cell may be capable of movement, but is not capable of becoming a blue-green algae cell.
- Seed structures should be similar in size to those of the smallest known self-replicating structures.
- The model should allow computational feasibility when used in conjunction with a genetic algorithm.

The EA model used in the experiments reported in this thesis is as follows. A 2-D cellular space is chosen since it has been used almost exclusively to study self-replicating structures[1]. The neighborhood template is the von Neumann neighborhood which consists of five neighbors including the center cell. All automata are weakly rotation-symmetric so that each distinguishes the relative locations of its four neighboring cells as top, right, bottom, and left. Each automaton is represented by a symbol in {`A`,`B`,`C`,`D`} indicating its component type. The set of actions used are described in Table 4.1.
Automata may move (both translation and rotation are included in the same action for convenience), divide into two copies (again movement is included for convenience), self-destruct, or remain inactive. Note that the `DV-ROT` action directly enables replication at the level of *individual* automata, not the self-replication of multi-automata (aggregate) structures. Although the divide action may appear to be "too powerful" (in the sense of making self-replication less difficult), we note that all previous self-replicating structures use a similar mechanism in their self-replication

[1]See Table 2.2 on page 24 for a summary of previous research.

Action	*Description*
`MV-ROT <dir> <rot>`	move one cell in the specified direction and rotate the specified number of degrees
`DV-ROT <dir> <rot> <dir> <rot>`	divide into two daughter automata according to the specified directions and rotations
`DESTR`	cease to exist
`NULL`	no action

Table 4.1: Set of actions A used in the EA model.

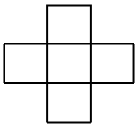
processes. For example, it is a simple matter to program a CA to have two quiescent cells become the same state of a shared neighboring cell. Again, the difficulty lies in achieving the self-replication of an entire structure. With β=4, values for the direction parameter (shown as `<dir>`) are either top, right, bottom, or left, and the rotation parameter (shown as `<rot>`) can be either 0, 90, -90, or 180 degrees. The key EA model parameters chosen are summarized in Figure 4.1.

4.2 Rule Discovery

The problem to be solved in this chapter is that of automatically finding rule tables that yield self-replicating structures in cellular space automata models. Problems of this kind are called *rule discovery* problems since the goal is to search a large search space composed of sets of simple rules and discover rules that have high performance. The rule discovery technique used here is the genetic algorithm. The application of GAs to rule discovery problems is most well-known in the study of classifier systems[Holland80, Booker90], which are massively parallel, rule-based machine learning systems that learn rules through the use of credit-assignment and rule discovery.

An overview of the specific rule discovery system used is illustrated in Figure 4.2. The main component of the system is the technique called the rule discovery process. Techniques other than the GA may be used instead, and these are discussed in Chapter 5. Inputs to the rule discovery process are as follows. The description of the cellular space model may be either the EA or CA models[2]. This description informs the rule discovery process of the relevant definitions concerning the cellular space model being investigated, including: the manner in which rules are processed, the type of space defined in the particular model, and how the space iterates over time. The evaluation criteria specify the manner in which discovered rule tables are judged. In the context of the genetic algorithm, these criteria are called fitness functions. This is the most difficult part of the system to design since it is not obvious how to apportion fitness to encourage and sustain self-replicating behaviors. This subject is described later in this chapter and is a key innovation of this thesis. The initial conditions specify the configuration of cells at $t = 0$ (the seed structure S_0), and parameters associated with the rule discovery process. In the case of a GA as the rule discovery process, such parameters would include mutations and crossover rates, population size, the number of generations, and convergence criteria. Also shown in Figure 4.2 (lower right) is

[2]Other cellular space models may be used, such as stochastic automata, however they are beyond the scope of this work.

Parameter	*Value(s)*
N	2 dimensional space
β	4 coordinate system rotations (90°)
n	5 cell von Neumann neighborhood 
A	$\{$MV/ROT(d,r), DV/ROT(d_1,r_1,d_2,r_2), DESTR, NULL$\}$
k_s	1 strongly rotation symmetric cell state (quiescent)

(a)

	Parameter Sets		
	Set 1	*Set 2*	*Set 3*
S_0	$\hat{v}_1$ $\hat{v}_2$	$\hat{v}_1$ $\hat{v}_2$ $\hat{v}_3$	$\hat{v}_1$ $\hat{v}_2$ $\hat{v}_3$ $\hat{v}_4$
c	2	3	4
k	5	13	17
k_w	4	12	16
$\hat{V}$	$\{\hat{v}_1, \hat{v}_2\}$	$\{\hat{v}_1, \hat{v}_2, \hat{v}_3\}$	$\{\hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}_4\}$

(b)

Figure 4.1: CA and EA model parameters used in the genetic algorithm: (a) parameter values used for every GA; (b) sets of parameters for varying seed structures.

the final step in the rule discovery system. Because the rule discovery processes examined here do not guarantee finding a rule table that promotes self-replicating behavior, the discovered rule table requires simulation and subsequent analysis to determine if the structure self-replicates. The criteria for such determination is described next, where a definition of a self-replicating structure is presented.

4.3 Self-replicating Structures

A structure S and a metamorphosing structure $\tilde{S}$ were formally defined in Section 3.1.2. Briefly, a structure is a set of contiguous non-empty (non-quiescent) cell states, and a metamorphosing structure is a set of cell states that forms a structure at time t, changes shape from $t+1$ through t', and is identifiable as the original structure at $t'+1$. A *self-replicating* structure S^r builds upon those definitions, and understanding the definition of a self-replicating structure is a prerequisite to understanding the fitness functions presented in subsequent sections.

In defining a self-replicating structure the notion of separation between structures needs to be

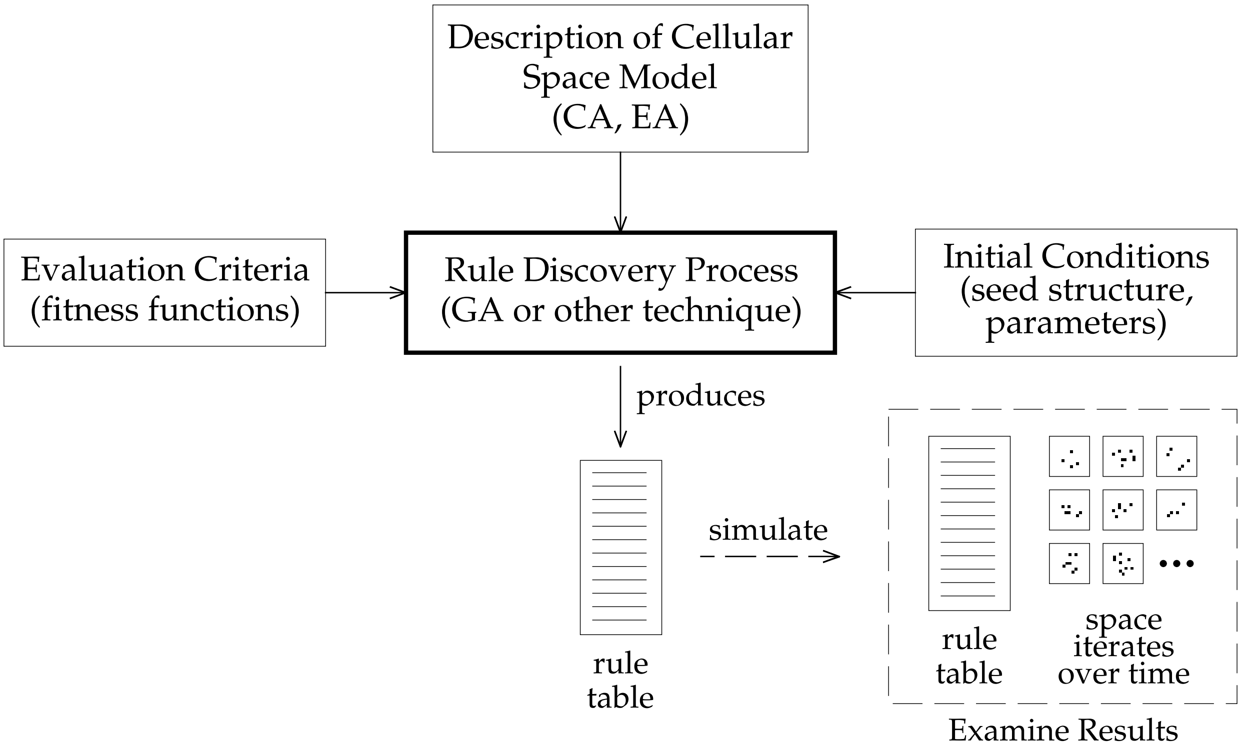


Figure 4.2: Overview of the rule discovery system showing the major components, production of a discovered rule table, and the manner in which the discovered set of rules is analyzed.

made precise. The three degrees of separation among two structures (or in general, configurations), are noted. Recall that the set of all non-empty cells in a configuration C is the support function, $\sup C$. Two configurations C and C' are *distinct*[3] if

$$\sup C \neq \sup C' \tag{4.1}$$

C and C' are *disjoint* (Equation 3.6 repeated for convenience) if

$$\sup C \cap \sup C' = \emptyset \tag{4.2}$$

The third and strongest form of separation is called *isolation.* Recall Equation 3.11 which defines the neighborhood function of a cell α as $g(\alpha)$. Let the *neighborhood function of a configuration* C be defined as the set of all cells that are in the neighborhood of C's non-quiescent cells. This function is denoted $G(C)$ and is expressed as

$$G(C) = \bigcup_{\alpha \in \sup C} g(\alpha) \tag{4.3}$$

[3] Term used in [Moore62, pg. 22]

A configuration C is *isolated* from configuration C', denoted $C \dashv\vdash C'$, if the set of cells common to both configuration's neighborhoods is not in $\sup C$. This is expressed as

$$\sup C \cap (G(C) \cap G(C')) = \emptyset \tag{4.4}$$

Figure 4.3 illustrates with an example the differences between the degrees of separation among configurations.

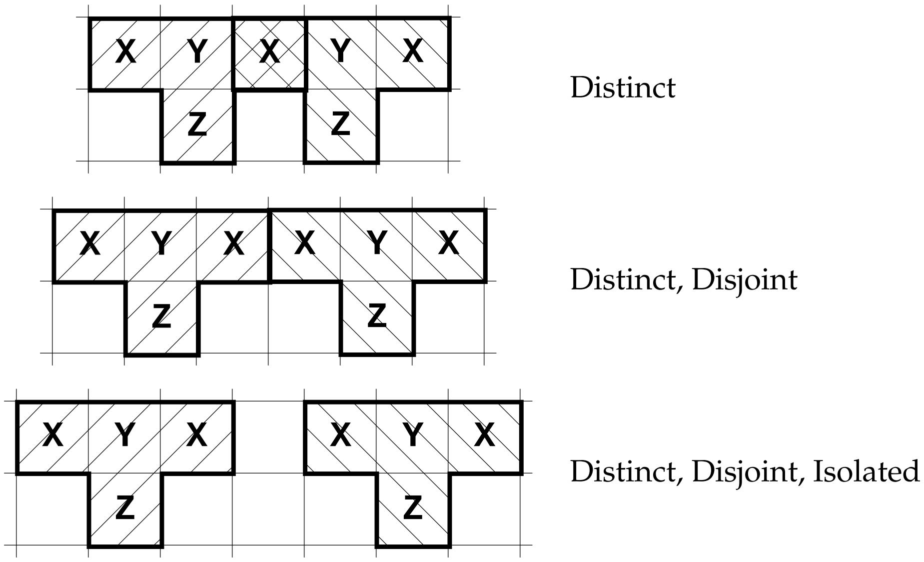


Figure 4.3: Illustration of the terms distinct, disjoint, and isolated with respect to two example 4-component structures. The von Neumann neighborhood is assumed.

Definition 4.1 A configuration S^r is a *self-replicating structure* if all the following criteria are met.

1. S^r is a structure (Definition 3.2), and is comprised of more than one non-quiescent cell:

$$|S^r| > 1 \tag{4.5}$$

2. S^r becomes a metamorphosing structure $\tilde{S}^r$ (Definition 3.3) during its self-replication process.

3. Copies, possibly translated and/or rotated, of S^r, called replicants, are created in neighbor-adjacent cells by the metamorphosing structure $\tilde{S}^r$. Such replicants are denoted S_i^r with i representing the ith generation offspring ($i = 1, 2, \dots$).

4. There exists a time t such that S^r can produce i replicants, for any positive integer i, for infinite cellular spaces (Moore's criteria [Moore62, pg. 22]).

5. If the self-replication process begins at time t, there exists a time $t + \Delta t$ (for finite Δt) with

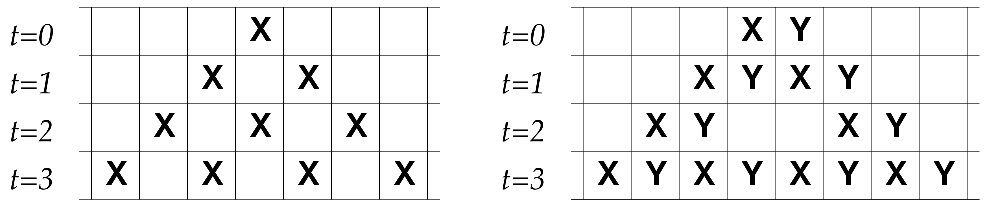
$$\Delta t > 1 \tag{4.6}$$

such that replicant S^r_1 becomes isolated from the parent structure:

$$S^r \dashv\vdash S^r_1 \tag{4.7}$$

The above definition, taken as a whole, is more precise than previously reported definitions in which construction universality was not required[4]. This is definition used for the self-replicating structures presented in this thesis, and we note here its benefits. Firstly, it encompasses the more recently reported models of self-replication (those starting with [Langton84]). Secondly, it precludes many trivial self-replication processes, discussed in more detail below. And lastly, it precludes "artifact" replicants – structures that form the appropriate size and shape, for example, from a supply of unused components without being directed to do so. Such "artifact" replicants are constructed in a random fashion, and are more likely to appear as the seed size (number of components in the seed) becomes smaller.

An important issue that arises from any definition of a self-replicating structure concerns *trivial* self-replicating structures. Such structures are seen as not requiring a stored instruction sequence that is interpreted during the replication process. For example, a 1-D, 3-neighbor CA can easily be made to give the behaviors shown in Figure 4.4. In both examples shown, the seed structures are shown at t=0 and at t=3 replicants can be seen isolated. Note that in the definition of self-replicating structure Figure 4.4(a) would not be included because of the requirement that the structure size be greater than one ($|S| > 1$). Also the definition above states that self-replication processes must have a duration of at least one time step as specified in Equation 4.6. This precludes many structures having a trivial self-replication process. While Figure 4.4(b) does meet this requirement, it is considered trivial because all of its components simultaneously split at t=1.



(a) (b)

Figure 4.4: Examples of trivial self-replicating structures in a 1-D, 3-neighbor cellular space model. Seed structures are seen at t=0, and with three more time steps shown. (a) 2-state, 1-component structure; (b) 3-state, 3-component structure.

A precise definition of a trivial self-replication structure has not appeared in the literature, although it has been mentioned numerous times [Moore62, von Neumann66, Thatcher70, Langton84]. The distinction adopted here is from [Langton84, pg. 137, original emphasis]:

> It seems clear that we should take the "self" of "self-reproduction" seriously, and require that the construction of the copy be *actively directed* by the configuration itself. That

[4] The requirement of construction universality was used in earlier models of self-replicating systems in order to avoid trivial self-replicators. However, such a requirement has been abandoned over the last decade or so (including the present work) mainly because biological self-replicating systems do not have such a requirement.

> is, responsibility for the production of the offspring should reside primarily within the sequences of actions undertaken by the parent structure. Note that we want to require that responsibility reside *primarily* with the parent structure itself, but not *totally*. This means that the structure *may* take advantage of certain properties of the transition function... ...but not to the extent that the structure is merely passively copied by mechanisms built into the transition function.
>
> ...the configuration must treat its stored information in two different manners... *interpreted*, as instructions to be executed (translation), and *uninterpreted* as data to be copied (transcription).

Thus, we distinguish between trivial and non-trivial self-replication by insisting that the structure actively directs the construction of offspring, as opposed to trivial cases where all component automata simultaneously split to form two copies.

With Definition 4.1, the goal of the genetic algorithm, or, more generally, any rule discovery process, can be stated as follows. Given an initial seed structure S_0 such that $C_0 = S_0$, find the rule table function δ

$$C_t = \delta(C_{t-1}) \qquad (t > 0) \tag{4.8}$$

which generates the sequence of configurations

$$\mathcal{C} = \{C_1, C_2, \dots\} \tag{4.9}$$

such that S_0 satisfies the requirements of a self-replicating structure as specified in Definition 4.1 during the propagation $\mathcal{C}$.

4.4 The Choice of Genetic Algorithms

Before describing the genetic algorithm as it is used in this thesis, the motivations for choosing the genetic algorithms as the rule discovery technique are as follows. As described in Chapter 3, the size of the search spaces for both EA and CA cellular space models can be incredibly large. GAs are a well-known strategy for searching such extremely large search spaces quickly [Mitchell96]. In addition to its size, the search space landscape is not well understood. Except for reports examining small (k=2) cellular space models [Wolfram94], apparently no studies have been reported which attempt to understand the larger search spaces ($k > 2$). Such search spaces are very unlikely to be smooth and unimodal, which would suggest gradient-ascent algorithms such as steepest-ascent hill climbing.

In this work, the goal of automatically finding self-replicating structures is not directly concerned with finding the optimal self-replicating structure, the definition of which would subjective. Rather, finding a diverse set of such structures is of greater importance and more interesting. Thus finding sufficiently good solutions instead of the global optimum is required. GAs are well-suited to such goals.

Experimental results from other search techniques are presented in Chapter 5 for purposes of comparison to the GA. The techniques used are multiple restart stochastic hillclimbing and population-based incremental learning. The results show that these techniques were not as effective as the genetic algorithm for the problem examined. In most cases the other search techniques failed to discover any self-replicating structures.

4.5 Genetic Algorithm Design

The theory of genetic algorithms was briefly reviewed in Section 2.3 (page 24). In this section the genetic algorithms employed in this thesis are described, with special emphasis on the derivation of the fitness functions used. Two genetic algorithms were designed in the course of this research. The primary GA was used to discover many self-replicating structures, and is the main focus of this section. An auxiliary GA, called a meta-level GA [Grefenstette86], was used to optimize certain parameters for the primary GA. This use of a second GA for multiobjective optimization is discussed in Section 4.7. Both genetic algorithms are variants of the traditional genetic algorithm [Davis91]. Accepted notation found in the genetic algorithm literature is used when appropriate. However, to avoid clashes with the notation used for cellular space models presented in Chapter 3, some GA symbols were modified slightly. The notation is shown in Table 4.2.

Symbol	*Meaning*
g	generation number
$\mathcal{P}$	population: set of chromosomes
a	population member: a chromosome
a_g^i	ith population member of generation g
n_a	population size: number of chromosomes

Table 4.2: Notation used for genetic algorithms.

An overview of the primary genetic algorithm as it is used in this thesis is depicted in Figure 4.5. Each area of the GA is discussed in detail in the sections below. Here some general remarks about the GA are made. The GA begins by assembling a population of randomly initialized rule tables, also called chromosomes in this context, which are on the order of 1000 elements long for the models studied. The GA then proceeds to iterate in a loop until a specific convergence criterion is satisfied. In Figure 4.5 the two overall phases of processing are seen: an evaluation of the population, and creation of a new population. Evaluating the population of chromosomes is the most time consuming operation since 100 simulations are executed and complex fitness calculations are made for each simulation. The creation of a new population of chromosomes is where genetic operators are applied with the intention of creating a new set of chromosomes with higher fitness values.

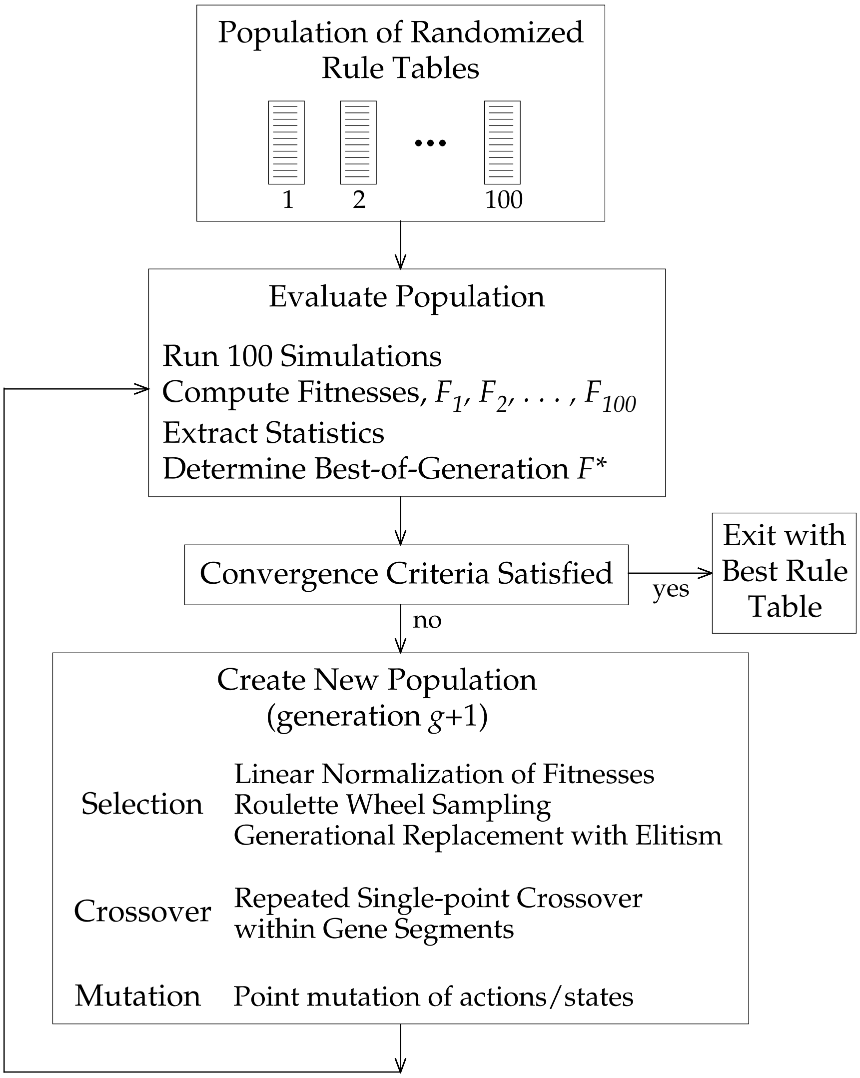


Figure 4.5: Overview of primary genetic algorithm as used in this thesis.

4.5.1 Encodings

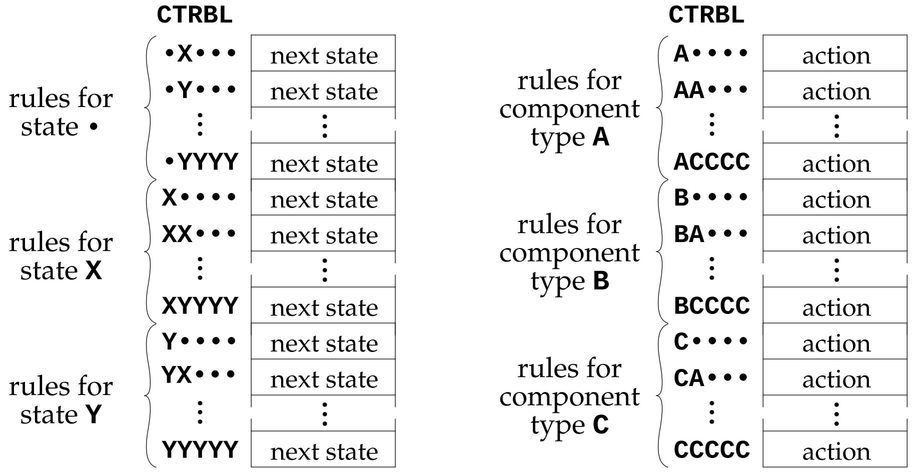
As discussed in Section 2.3, an artificial *chromosome* refers to a candidate solution for a given problem. In genetic algorithms, chromosomes are often encoded as binary strings, although other encodings, such as real-valued and tree encoding schemes are possible [Mitchell96]. The following discussion concerns the choice of encoding system for rule tables, and the specific encodings for rule tables used in this thesis. However, it is noted here that a binary encoding was chosen for use in an auxiliary genetic algorithm to be discussed in Section 4.7.

In addition to the encoding schemes mentioned above, another choice for representing candidate solutions to the GA is to use the *natural encoding* of the problem at hand. This approach is stated as part of the "principle of minimal alphabets" [Goldberg89, pg. 80]. The natural encoding for chromosomes in cellular space automata models is the rule table itself. This is the encoding strategy approach taken for the genetic algorithm described in this thesis. In [Davis91] it is argued that using the natural encoding of a problem confers two advantages over artificial encodings. Firstly, it allows the researcher to work with the GA in a more natural way given that he or she is already familiar with candidate solutions to the problem. Secondly, a natural encoding guarantees that domain expertise embodied in the encoding will be preserved.

Two additional motivating factors for using the natural encoding of a problem are as follows. In transforming the problem into an artificial encoding, a second step of decoding it back to the original form is required. This process incurs overhead to the GA, and given that the cellular space models are quite large and require large amounts of memory and CPU time, saving this encode/decode overhead reduces the computational load. A second factor concerns the fact that other encodings can be unwieldy for the large chromosomes required for representing rule tables.

As mentioned, the chromosomes in the GA are comprised of rule tables. As an example, the representation chosen to encode CA and EA rule tables for two and three component systems is depicted in Figure 4.6. In both example chromosomes, the rule tables are shown indexed implicitly by the neighborhood pattern `CTRBL` (center, top, right, bottom, left). From the derived equations in Chapter 3, the size of these chromosomes are computed as 838 (CA) and 768 (EA). As shown in Figure 4.6, rules for each component are grouped together within the chromosome. Note that because of rotational symmetry some groups will be larger than others. In GA terminology, such blocks of related adjacent elements in the chromosome are called *genes*. Grouping rules together as genes allows the GA to optimize rules for individual components separately. This can be thought of as programming `A` type "machines" separately from `B` type, etc. This grouping presumably aids in GA performance in light of the building block hypothesis of GA theory (reviewed in section 2.3). Partitioning the chromosome into such genes also allows for flexibility in applying the crossover operator (discussed in section 4.5.3).

The size of the chromosomes corresponds to $|\delta|$, the rule table size as defined in Chapter 3. This implies that the complete rule table is used as the chromosome, which is important for two reasons. Firstly, for larger models ($k > 5$) it is likely that during a fitness evaluation, some rules may never be activated. That being the case, because all possible are represented in the chromosome, the GA still manipulates these inactive rules. This is analogous to the introns ("junk DNA") in biological chromosomes. Note however, that due to the genetic operators in the GA, rules that are inactive in one generation, may be recombined and/or mutated and become active in the next. Thus segments of inactive loci on the chromosome may still contain valuable genetic information. Secondly, having the complete rule table in the chromosome implies that rules that are active (executed by the



(a) CA (b) EA

Figure 4.6: Examples of chromosome representation in the GA. Shown are weakly rotation symmetric models comprised of three components: (a) cellular automata; (b) effector automata.

automata) but do not change the automaton's state are manipulated as genetic material by the GA and are relevant to the model's behavior. An example of such rules is `XXYZX` → `X`, which shows the cell state `X` remaining as `X` for the next time step.

The ordering of rules within the chromosome plays an important role regarding the performance of the GA. Many rules in the rule table will need to be coadapted. However, if these rules were close to each other in the rule table, the probability of those rules being disrupted by crossover is reduced, and thus by the building block hypothesis, the GA would perform better. In GA terminology having such chromosome loci being near each other is called *linkage*. In the encoding method described above, the rules are ordered lexicographically within each state/component-type segment of the chromosome. Thus the ordering within in each segment is of an arbitrary nature. Based on the linkage argument above, this arbitrary ordering probably impedes the GA from finding better solutions more quickly. However if one knew how to properly order the rules within the chromosome a priori, much of the problem is solved. This is a famous paradox encountered by GA researchers and practitioners. The ordering of rules within the rule tables tries to limit the arbitrariness by grouping rules for each component/state together. However within each of these groups, it would be difficult to choose in advance specific orderings.

In addition to including the rule table in the chromosome, one might also include the initial configuration of the cellular space, known as the seed structure. In this manner the chromosome encodes all of the initial conditions for a simulation. Such an encoding was used in preliminary GA experiments. However, because encoding the rule table alone produced positive results using less computational resources, this encoding scheme was not investigated further.

4.5.2 Selection

Selection is the process of choosing individuals (chromosomes) from a population so that they may mate and produce "offspring" for the next generation. Numerous artificial selection methods for use with GAs have been reported in the literature [Goldberg89, pg. 121]. Selection of chromosomes to mate is based on the fitness of the chromosomes, and the goal is to give more reproductive chances, overall, to fitter chromosomes so that their offspring will in turn garner even higher fitness. If too many higher-performing chromosomes are selected, the GA may converge prematurely with a suboptimal group of high-performing chromosomes dominating the population. Conversely, if not enough high-performing chromosomes are chosen, the evolution will proceed slowly. There is no single selection technique that stands out as always being the best. For the GA experiments reported here, the selection technique involves three methods described below: linear normalization of fitnesses, roulette wheel selection, and elitism.

Linear Normalization of Fitnesses

After a population has been evaluated and the fitnesses for each of the chromosomes is known, the first step in selecting parents to mate is to apply linear normalization of the fitnesses. Linear normalization, a variant of rank selection, involves ordering the chromosomes linearly based on their fitness scores. For example, five chromosomes could have their fitnesses normalized and ordered as 30, 25, 20, 15, 10, with 30 representing the highest-performing chromosome. The distance between fitnesses (five in this example), is called the *decrement* and can be chosen as desired. A decrement value of 1 was chosen, and using a population size of 100 chromosomes, the normalized fitnesses are thus ordered $100, 99, \ldots, 1$. An ordering using a decrement value of 1 is a ranking method with a rank of 1 denoting the least fit chromosome.

Roulette Wheel Sampling

After the linear normalization of fitnesses, stochastic sampling of the ordered chromosomes is used to randomly select parents, with each parent's chance of being selected directly proportional to its fitness. This technique is referred to as roulette wheel sampling since it may be thought of as allocating sectors of a circular roulette wheel with each sector sized according to a chromosome's fitness. Using linear normalization of fitnesses as described above, the probability of selecting chromosome a^i from a population of n_c chromosomes is given by

$$\text{prob(selecting } a^i) = \frac{i}{\sum_{j=1}^{n_c} j} \tag{4.10}$$

The population size n_c used in the GA experiments herein was 100. For illustration purposes, Figure 4.7 shows how the sampling probabilities breakdown for a population size of $n_c = 5$. In this case, the value of the denominator in Equation 4.10 is 15. Thus the chromosome ranked highest in fitness (number 5), will be selected to mate $\frac{5}{15}$ or 33.3% of the time.

Elitism

The number of times roulette wheel sampling is performed in each generation depends on the generational replacement policy used. Given a population at generation g, $\mathcal{P}_g$, the question becomes, how many new chromosomes will be created for the next population $\mathcal{P}_{g+1}$, and how many existing chromosomes will simply be copied over. The fraction of new chromosomes placed into $\mathcal{P}_{g+1}$ is called the *generation gap* [Goldberg89, pg. 111]. For the GAs in this thesis, a generation gap of 98% was used.

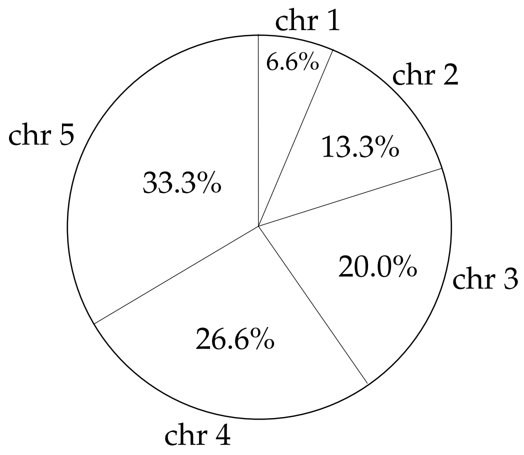


Figure 4.7: Illustration of roulette wheel sampling with five chromosomes. "chr i" denotes chromosomes and percentages shown correspond to the probability of being selected.

Elitism [De Jong75, pg. 102] is a technique in which the GA is forced to retain a specific number of best individuals at each generation. The method of elitism as it is used here is as follows. Let a_g^* and a_g^{**} be the individuals with the highest and second-highest performance, respectively, from population $\mathcal{P}_g$ at generation g. Population $\mathcal{P}_{g+1}$ is constructed as usual, with the exception that the first two members of $\mathcal{P}_{g+1}$ are copies of a_g^* and a_g^{**}. Thus it is seen that these two elite individuals may propagate from generation to generation. Through experimentation, selection using elitism was found to outperform the same selection technique without elitism.

4.5.3 Crossover and Mutation Operators

This section describes how the genetic operators of crossover and mutation were adapted in the GA. A discussion of these operators can be found in Section 2.3. Many variations of crossover are possible. Performing single-point crossover on bit-string encodings is a relatively straightforward procedure (see Figure 2.12), however when applied to rule tables, some aspects of this technique were modified.

As previously discussed, the chromosomes are rule tables. Example rule tables for both CA and EA models are shown in Figure 4.6. In bit-string chromosomes, the lowest level genetic information is the bit, and crossover points are chosen between adjacent bits. For rule tables, crossover points are chosen at the boundary points between rules. The type of crossover used here is a version of multi-point crossover whereby single-point crossover is applied within gene segments, as shown in Figure 4.8. As mentioned earlier, genes correspond to segments of the rule table that contain rules for a single state (in the CA model) or component (in the EA model). These segments are labelled and marked by a heavy line in the diagram. A crossover point is randomly selected within each gene segment, and single-point crossover occurs. The diagram shows an EA model where each component has only five rules (an EA model with so few rules is unrealistic, but this is shown only to illustrate the crossover technique clearly). Two children chromosomes are shown with the results of the per-gene segment crossover.

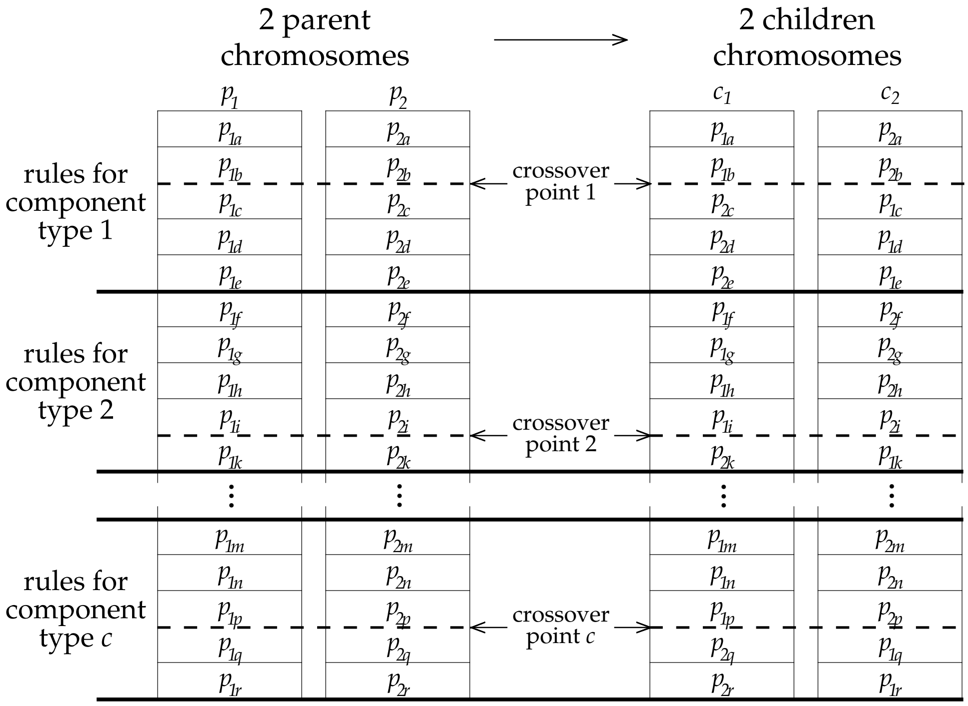


Figure 4.8: Illustration of crossover using EA rule tables. Parent chromosomes p_1 and p_2 are recombined to form offspring c_1 and c_2 by segmenting the rule tables into c partitions (genes) according to component type, and crossing over rules within each partition (as in Figure 2.12(a)), with each crossover point chosen at random.

This approach was taken for the following reason. Since we are mainly concerned with weakly rotation symmetric cellular space models (i.e., models having components), each gene segment of the rule table specifies the behavior of a specific automaton. Performing crossover within each segment allows the GA a more detailed granularity in optimizing the behavior of each automaton individually. Thus at a low level, the GA evolves each component type ("species") separately. At a higher level, because the fitness functions are rewarding cooperation among components, component types are evolved together in a coadapted manner. Indeed, empirical results comparing this crossover technique to that of single-point crossover (across the entire rule table) showed higher performance for the multiple application of crossovers.

Crossover is only applied to a fraction of the population of chromosomes. This fraction is determined by the crossover probability p_c, a parameter of the GA. After each set of two parent chromosomes are chosen, a biased decision is made whether to perform crossover or copy the parents directly into the next generation. If it is decided to perform crossover, then each gene segment of the chromosome is crossed-over in the manner described above.

After selection and crossover have produced two children chromosomes, each is subject to the mutation genetic operator. As with crossover, mutation is applied in only a fraction of rules in the rule table, based on the mutation probability parameter p_m. Mutation works in a similar manner for both CA and EA chromosomes. For a CA rule table, when a rule is selected for mutation, it is replaced by randomly choosing one state from the k states specified in the model. Similarly, for an EA rule table, a randomly selected action from the set of actions A is chosen to replace the original rule. An example of this is shown in Figure 4.9.

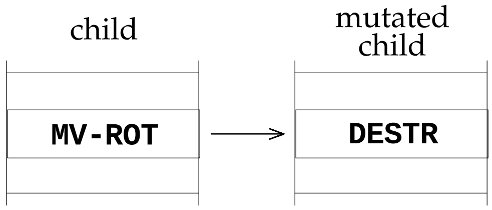


Figure 4.9: Example of an EA rule undergoing mutation. A movement/rotation rule is mutated into a destruct rule.

4.5.4 Fitness Functions

The purpose of a GA fitness function is to assign a measure of performance to each chromosome in the population, depending on how well each chromosome (rule table) encodes rules that result in an initially-specified structure exhibiting self-replicating behavior. Designing a fitness function to evaluate self-replication is difficult because self-replication is a dynamic and complex process. Naive measurement of the number of replicants is not useful early on as none of the initial random chromosomes produce replicants. This has been borne out in extensive testing of randomly initialized chromosomes, and agrees with intuition, given the immense search space sizes discussed in Chapter 3. Further, comparing an evolving structure to a predefined template of seed structure copies by way of pattern matches fails to give partial credit during the replication cycle itself, when the structure has changed shape as it directs its self-replication. Having a predefined template also imposes a strong bias on the self-replication process, which is undesirable since it severely limits the types of self-replicating behaviors that could possibly emerge.

Another difficulty is that, since the length of the desired self-replication cycle is unknown, using data from a single time-step would require knowing a priori which configuration replicants appeared in and assumes that replicants appear all at once rather than at different time-steps. Clearly, data from multiple time-steps are needed so as to identify replicants as they are produced. This leads to this problem of deciding which configuration to start with, and how many subsequent configurations to examine for self-replicating behavior.

Since the GA begins with a population of randomized rule tables (Figure 4.5), it is extremely unlikely that such rule tables will lead to self-replicating behavior. If the fitness functions of the GA assign positive fitness values only to rule tables that lead to self-replicating behavior, then all rule tables will have fitnesses of zero, and the GA will not be able to apply its genetic operators effectively. In such cases the GA degenerates into an inefficient random search process. Assigning small values of fitness to behaviors that do not resemble self-replication yet have potential to evolve into such a process is needed to allow the GA to search effectively. It is noted here that no other research to date[5] has reported techniques that attempt to cope with this problem.

[5] With the exception of [Lohn95] where preliminary advances are reported.

The issues raised above can be summarized as two questions: which simulation configurations should be used and what criteria should be applied for an effective evaluation of self-replicating processes. The following sections present novel solutions to these questions. The fitness functions derived are *general* in three senses. Firstly, they may be applied to a large number of 2-D cellular space automata models (both EA and CA models of varying sizes). Secondly, any size and shape seed structure containing unique components may be used. Thirdly, the fitness functions are not specific to GAs, and may be used in conjunction with a variety of search techniques. Evidence of these points is presented in Chapter 5 where the fitness functions are used in both CA and EA models, with varying seed structures, and under different search techniques. In addition, the fitness functions do not impose a strong bias toward any particular process of self-replication. That is, in their definitions, the fitness functions do not assign fitness based aspects such as: the contents of specific cell locations at specific instants, whether/how the structure should translate or rotate itself over time, the quantity/timing of replicant production, or to what extent do configurations match a predefined configuration.

4.5.4.1 Evaluations

Prerequisite to understanding how the fitness functions are derived is to recognize the manner in which they are used. As mentioned previously, fitness functions associate fitness values to each rule table (chromosome) in the population of a GA. This section discusses how each rule table is evaluated, and the reasoning behind how the evaluations were set up. Specifically, details concerning initial conditions and progress of the simulation are presented. Figure 4.10 depicts the evaluation phase starting with the selection of one rule table from the population at generation g.

The evaluation of each chromosome requires that a complete EA (or CA) simulation be executed (Figure 4.10, middle). As in other dynamical systems, initial conditions play a critical role in determining the cellular space's behavior. The initial conditions for each evaluation are comprised of a rule table, δ, and seed structure S_0. While it is possible to have the GA evolve both δ and S_0 simultaneously, the results of preliminary experiments in this direction were disappointing. Thus, the decision was made to keep S_0 fixed – every evaluation that a single GA performs uses the same S_0. The seed structures were comprised of the two, three, and four unique components as shown in Figure 4.11. The choice to use small structures was based on research showing that self-replicating structures as small as five and six components existed [Reggia93]. When using small seed structures such as these, the nature of the self-replication process concerns the self-influencing forces/behavior that different interacting components have on each other. For example, because individual components generally move at each time step, their inputs (the automata located in neighboring cells) change regularly. With each new set of inputs, the component can execute a new rule. Thus when a seed structure moves or rotates as a whole, components of the seed structure can influence other components, creating a self-influencing, self-directing process. The analogy of epistatic interactions in biological genes is appropriate here: like genes at different locations on the chromosome which can suppress the expression of other genes, the components of a self-replicating structure can affect the behavior of other components in the same structure. As in previously reported unsheathed self-replicating structures, the components are thought of in two ways: as the instruction sequence, and as the machinery to read the instructions.

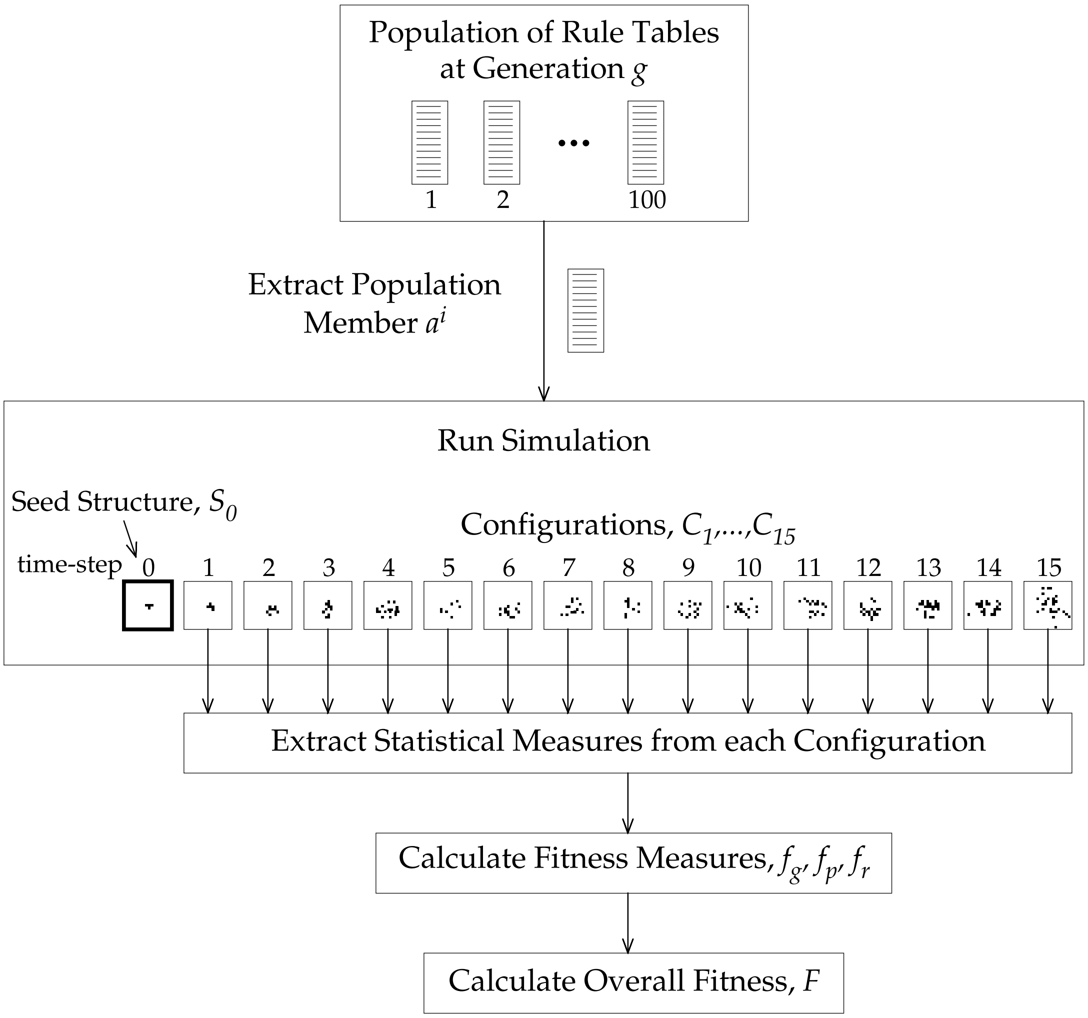
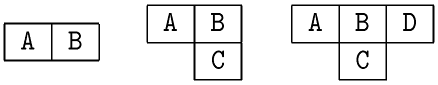


Figure 4.10: Evaluation phase of genetic algorithm.

Configurations Used in Fitness Functions

As seen in the block labelled "Run Simulation" in Figure 4.10, the set of configurations to be used by the fitness functions is shown as $C_1, C_2, \ldots, C_{15}$. The choice of which configurations to use in the evaluation of a self-replication process is an important design parameter and is discussed now.

Letting t_0 and Δt denote the first time-step and the duration of time which will be examined for fitness calculations, respectively, the tradeoffs can be stated as follows. If Δt is too small, this may not give enough time for a self-replicating process to emerge. If Δt is too large, two undesirable situations will arise. First, the efficiency of the GA will go down since the GA will be spending more time examining behaviors that, in general, do not exhibit self-replication. As seen earlier in Figure 4.5 on page 59, the simulations are inside two loops of the GA: one for each population member, and one for each generation. The product of these two numbers is on the order of 200,000 for our experiments. Thus the expression $200,000\Delta t$ represents the total number



(a) (b) (c)

Figure 4.11: Seed structures: (a) 2-component; (b) 3-component; (c) 4-component.

of simulation time-steps executed during the GA. Later, in Chapter 5, it will be seen that 100 GAs are required for statistical sampling purposes. Therefore each increment to Δt adds 20,000,000 more time-steps to the overall GA, which becomes a significant computational burden. Second, as Δt increases, the likelihood of spurious seed structure copies appearing increases, which could potentially disrupt fitness function calculations. Such spurious copies could then inflate the fitness values and steer the GA in the wrong direction. Based on previous studies of hand-designed self-replicating structures [Reggia93] and considering these tradeoffs, a value of $\Delta t < 10$ was determined too restrictive and $\Delta t > 20$ too large for the reasons cited above. Thus a value of 15 time steps was chosen.

The first time step at which fitness calculations begin (t_0) is also very important, however an easier choice to make. Since the seed structures that we deal with are very small, fast replication cycles are very likely [Reggia93]. Such cycles are generally less than 10 time-steps, with critical steps of the self-replication process occurring very early on, generally in the first five time steps. Therefore, choosing a value $t_0 > 5$ runs the risk of excluding valuable information from the fitness function. So as not to exclude any useful information that could occur early on, t_0 was chosen as the first time-step. Summarizing these parameters, we have

$$t_0 = 1, \qquad \Delta t = 15 \tag{4.11}$$

which implies that the following set of configurations are to be used in conjunction with fitness calculations:

$$C_1, C_2, \dots, C_{15} \tag{4.12}$$

This set of configurations is called the set of *critical configurations*, and is defined as, in general,

$$C_{t_0}, C_{t_0+1}, \dots, C_\tau \tag{4.13}$$

where $\tau = t_0 + \Delta t - 1$.

Outline of Fitness Calculation

An overview of the fitness function calculation is seen in the lower three blocks of Figure 4.10. Running a simulation generates 15 configurations, from which statistics are collected. These statistics are described in more detail in later sections, but briefly, they can be classified as time-averaged component counts (multiplicities), adjacency information, and replicant counts. Multiplicity values $M^t_{\hat{v}}$ record the quantity of each component type $\hat{v}$ over time. Adjacency information includes relative positioning data regarding each component type over time. Continuing downward in Figure 4.10, after collection of these statistics, fitness measures are computed and then combined in the fitness function F to give the overall fitness value of each simulation.

After carefully studying the behaviors of previously reported hand-designed self-replicating structures [Reggia93], and performing experiments with an initial set of simple fitness functions, it became obvious that a sophisticated fitness function was required to properly evaluate potential self-replicating structures. It was concluded that the problem of fitness function design involved multiple, independent, criteria that would need to be combined into a single fitness value. Problems of this type are called *multiobjective* optimization problems.

Three independent criteria, called *fitness measures* here, were hypothesized and later tested. The first is a *growth* measure, denoted f_g, which correlates growth of individual component types with high performance. The second criteria is called the *relative position* measure, denoted f_p. This measure is concerned with awarding fitness to component types that have a high percentage of neighboring-cells positioned in the same manner as is seen in the seed structure. The third criteria is one that measures *isolated replicants*, denoted f_r. This function scans configurations looking for isolated[6] replicants and awarding proportionate amounts of fitness depending upon the number of replicants seen over time.

As mentioned previously, components of the same component type have identical behavior, and differing component types will generally behave in differently. In other words, all component of type $\hat{v}$, when presented with identical neighborhood-adjacent cells, will execute the same rule. Thus it is appropriate to treat components within the same component type as a group that can be evolved separately. Because of this property, two out of the three fitness measures introduced above judge and assign fitness based on component type properties.

A surprising insight learned during the design process was that, in general, one needs to keep *relaxing* fitness function criteria, instead of making it more stringent. It might be thought that by tightening the requirements, the GA would home-in on the appropriate rule tables faster. Quite the opposite was found to be true. By imposing less on requirements (encoded in the fitness function), partial fitness credit is gained faster, and the GA is more free to explore the search space, resulting in less restrictions placed on the self-replication process. For example, in calculating the relative position measure f_p, rather than using both position *and* orientation information, which yielded poor results, using position-based statistics alone gave much better results. Of course, relaxing the fitness measures too much will result in less positive reinforcement to the GA and thus is detrimental too.

The fitness measures described above are combined to give the overall performance of the simulation (Figure 4.10, bottom). This function F calculates the overall fitness value of the chromosome being evaluated, which then is used in the selection process of the GA. Since the relative importance of each fitness measure is unknown, rather than always apportioning equal weight to each, we define the fitness measure vector as

$$\mathbf{f} = (f_g, f_p, f_r) \tag{4.14}$$

and a weight vector

$$\mathbf{w} = (w_g, w_p, w_r) \tag{4.15}$$

The overall fitness is the dot product of these vectors:

$$F = \mathbf{f} \cdot \mathbf{w} \tag{4.16}$$

[6]Isolation has a precise meaning and is defined in Equation 4.4 on page 55.

For convenience, the fitness measure functions in $\mathbf{f}$ are each normalized to values in $[0, 1]$, and weights are such that $w_g + w_p + w_r = 1$. Two approaches were used to set the weight values in $\mathbf{w}$: manual settings guided by experimental results, and by using a meta-level GA as described in Section 4.7. Both approaches were successful, and experimental results are shown in Chapter 5. Given an appropriate weight vector $\mathbf{w}$, we next turn to the design of the fitness measures in $\mathbf{f}$.

4.5.4.2 Growth Measure

In order for a self-replicating process to emerge, one would expect to observe, over time, increasing quantities of the individual components. Such behavior can be seen in any of the reported hand-designed self-replicating structures, for example in Figure 2.7 on page 21. In analyzing past self-replicating structures it was seen that individual component counts, or multiplicities, generally increase over time, punctuated by periods of plateaus and small decreases in value. Again using the self-replicating structure in Figure 2.7, the graph in Figure 4.12 shows the multiplicity profile over the first 50 time-steps. Note that the multiplicities generally increase over time. One exception is the # component, which is not technically part of the structure, although it is used during the self-replication process. Its multiplicity remains at zero much of the time since it appears approximately every 10 time-steps.

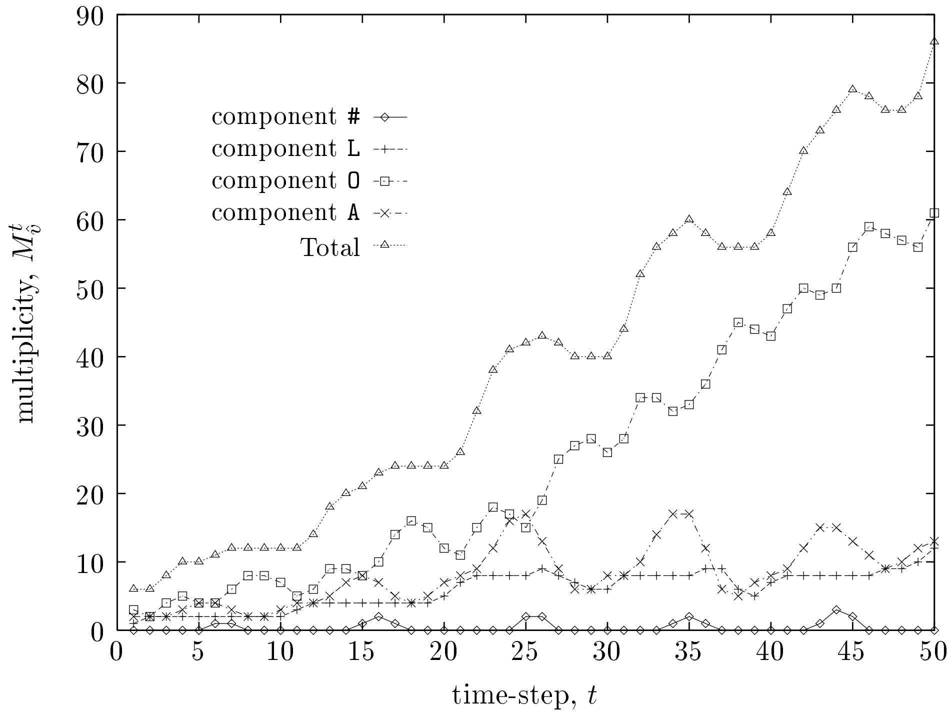


Figure 4.12: Multiplicity profile for self-replicating structure of Figure 2.7.

From observations like these described above, a growth measure function based on the production of individual components was designed. It measures to what degree each component type maintains an increasing supply of components from one time-step to the next. Recall that the quantity called multiplicity represents the number of components of type $\hat{v}_i$ at time t, and is denoted $M_{\hat{v}_i}^t$. The multiplicity data forms a $\tau \times c$ table, since τ time-steps are used and c components are present in the simulation:

$$\begin{matrix} M^1_{\hat{v}_1} & M^1_{\hat{v}_2} & \cdots & M^1_{\hat{v}_c} \\ M^2_{\hat{v}_2} & M^2_{\hat{v}_2} & \cdots & M^2_{\hat{v}_c} \\ \vdots & \vdots & \ddots & \vdots \\ M^\tau_{\hat{v}_1} & M^\tau_{\hat{v}_2} & \cdots & M^\tau_{\hat{v}_c} \end{matrix}$$

In order to distill these values into into a single meaningful value, multiplicities are first converted via a production function $\rho_{\hat{v}}$, which assigns fitness based on whether a given component type increased its production or stayed the same, and no fitness if it decreased:

$$\rho_{\hat{v}}(t) = \begin{cases} 1 & \text{if } M^t_{\hat{v}} > M^{t-1}_{\hat{v}} \\ 0.5 & \text{if } M^t_{\hat{v}} = M^{t-1}_{\hat{v}} \\ 0 & \text{if } M^t_{\hat{v}} < M^{t-1}_{\hat{v}} \end{cases} \qquad 0 < t \leq \tau \tag{4.17}$$

Equation 4.17 shows that $\rho_{\hat{v}}(t)$ is a function that assigns relative-worth values on the basis of growth. For example, if there were 12 `Y` components at $t = 5$ and 14 at $t = 6$, then $\rho_{\mathtt{Y}}(t = 6)$ would be assigned a value of 1. Note how ρ encourages increased quantities of components from one time-step to the next. However, it does not harshly penalize when production declines. Equation 4.17 can be thought of as awarding twice as much fitness to components that divide versus components that do not divide, yet remain active.

The growth measure f_g is then calculated by summing all $\rho_{\hat{v}}$ values (i.e., over all times and all components) and then dividing by the total attainable fitness as follows

$$f_g = \frac{1}{\tau c} \sum_{\hat{v} \in \hat{V}} \sum_{t=1}^{\tau} \rho_{\hat{v}}(t) \tag{4.18}$$

The summations in the function f_g total the $\rho_{\hat{v}}$ values earned for each component type over each of the $\tau = 15$ time-steps. Thus f_g calculates a measure of how well the supply of components increased. One might propose simply using a function that assigns high fitness when the total component count increases over time. However, since this does not distinguish among individual component types, such a function will encourage growth of only one or possibly two components, as this will satisfy such a function.

4.5.4.3 Relative Position Measure

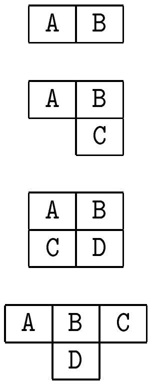
The relative position measure is the most critical fitness measure of the three presented in this chapter. Again, the overall goal of the three fitness measures is to encourage self-replicating behaviors. The growth measure approaches this goal from the perspective of supplying components for the self-replicating process. Assuming it is successful, it is desired to position this increasing supply of components in such a way that they influence each other, and that such influences produce self-replication. In order to encourage such positioning, it was hypothesized that over time, an individual component should, quite frequently, find itself surrounded by the same components that surrounded it in the seed structure. In other words, if components $\hat{v}_i$ and $\hat{v}_j$ are neighbor-adjacent and part of a self-replicating structure S_0, $\hat{v}_i$ should regularly have $\hat{v}_j$ positioned in the same relative manner found in S_0. The function f_p measures the degree to which such relative positions are satisfied over time.

It is important to realize that correct relative positions do *not* necessarily have to occur *simultaneously* (i.e., during the same time-step) among the components of the structure in order for partial fitness to be awarded by f_p. The ability of f_p to give partial fitness in this manner is critical to providing the GA with the initial positive reinforcement needed to search effectively.

The seed structure S_0 plays a critical role in deriving the function f_p since it contains the relative positioning information. The adjacencies contained in S_0 are formulated in terms of an *adjacency vector*, $\mathbf{s}$ which contains c elements representing the number of neighborhood-adjacent components for each component type:

$$\mathbf{s} = (s_{\hat{v}_1}, s_{\hat{v}_2}, \ldots, s_{\hat{v}_c}) \tag{4.19}$$

where $s_{\hat{v}_i}$ represents the number of components that are neighborhood-adjacent to component $\hat{v}_i$. Examples of $\mathbf{s}$ are shown in Figure 4.13.



(a) $\mathbf{s} = (1, 1)$

(b) $\mathbf{s} = (1, 2, 1)$

(c) $\mathbf{s} = (2, 2, 2, 2)$

(d) $\mathbf{s} = (1, 3, 1, 1)$

Figure 4.13: Examples illustrating the adjacency vector of various seed structures.

The function $m_{\hat{v}}(t)$ represents the number of neighbors of component $\hat{v}$ at time t that were the same type and in the same relative position as in the seed. The function $\sigma_{\hat{v}}(t)$ represents to what degree, at time t, all the components of component type $\hat{v}$ have the same neighbors as in the seed and is defined as:

$$\sigma_{\hat{v}}(t) = \begin{cases} 0 & \text{if } M^t_{\hat{v}} \leq 1 \\ \frac{m_{\hat{v}}(t)}{M^t_{\hat{v}} \cdot s_{\hat{v}_i}} & \text{if } M^t_{\hat{v}} > 1 \end{cases} \tag{4.20}$$

When $M^t_{\hat{v}} \leq 1$, component $\hat{v}$ is extinct or is presumably part of the seed. When $M^t_{\hat{v}} > 1$, $\sigma_{\hat{v}}(t)$ is the ratio of $m_{\hat{v}}(t)$ to the maximum possible. As in the growth fitness measure, a $\tau \times c$ table of values is generated by σ:

$$\begin{array}{cccc} \sigma_{\hat{v}_1}(1) & \sigma_{\hat{v}_2}(1) & \cdots & \sigma_{\hat{v}_c}(1) \\ \sigma_{\hat{v}_1}(2) & \sigma_{\hat{v}_2}(2) & \cdots & \sigma_{\hat{v}_c}(2) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{\hat{v}_1}(\tau) & \sigma_{\hat{v}_2}(\tau) & \cdots & \sigma_{\hat{v}_c}(\tau) \end{array}$$

We then define f_p to be the mean of $\sigma_{\hat{v}}(t)$ over all component types and all time-steps. Columns of the above table are summed and these c sums are then weighted by $\mathbf{s}$ as follows:

$$f_p = \frac{1}{\tau \sum_{\hat{v} \in \hat{V}} s_{\hat{v}}} \sum_{\hat{v} \in \hat{V}} \sum_{t=1}^{\tau} s_{\hat{v}} \sigma_{\hat{v}}(t) \tag{4.21}$$

The adjacency vector $\mathbf{s}$ as used in 4.21 gives higher priority to components that have more neighbors in the seed structure. For example, the `B` component in Figure 4.13(d) receives a weight of $\frac{3}{6}$ or 50% and the other components each receive $\frac{1}{6}$ or 17% weight.

4.5.4.4 Isolated Replicant Measure

The isolated replicant fitness measure f_r correlates fitness with increasing numbers of isolated replicants formed during the course of a simulation. In contrast to the relative position fitness measure, f_r provides little if any positive reinforcement to the GA during the beginning of the discovery process. Its main purpose is to guide the GA toward fitter and fitter self-replicating structures once nascent ones have been discovered. Experimental data confirming that this occurs is presented in Chapter 5.

Letting r_t represent the number of detached replicants in configuration C_t. We calculate the maximum number of isolated replicants that have appeared during the entire simulation, from $t = 1$ to $t = \tau$. This is then scaled by a sigmoid function centered at θ as follows:

$$f_r = \frac{1}{1 + \exp(-(\max(r_t) - \theta))} \qquad 0 < t \leq \tau \tag{4.22}$$

The constant θ represents the number of isolated replicants at which the rate of increasing fitness decreases (i.e., the inflection point of the sigmoid). As an example, Figure 4.14 shows the scaling for $\theta = 4$. Thus fitness is assigned at a faster rate during periods when two or three isolated replicants are seen. The production of small quantities of such replicants is a great importance since it is usually a sign that a viable self-replicating process has been initiated.

4.6 Convergence Criteria and Parameter Values

This last description concerning the GA design concerns the criteria for convergence and the parameter values used. Regarding convergence, the GA continues to iterate over many generations until one of the following criteria are satisfied.

- If the best-of-generation chromosome achieves a fitness greater than 0.9, the GA is considered to have converged.
- Otherwise the GA continues until it reaches generation $g_{\max}$.

The GA parameters used are shown in Table 4.3. Ranges of parameters denote that during the course of experimentation, parameters were varied slightly. It has been argued in the GA literature that using large population sizes and small numbers of generations produce better results [Goldberg89]. In Table 4.3, it can be seen that the chosen parameters do not align with this argument. The reason for this is a practical one involving computer system resources. To be able to compare GA performance across numerous experiments, the largest population size feasible was

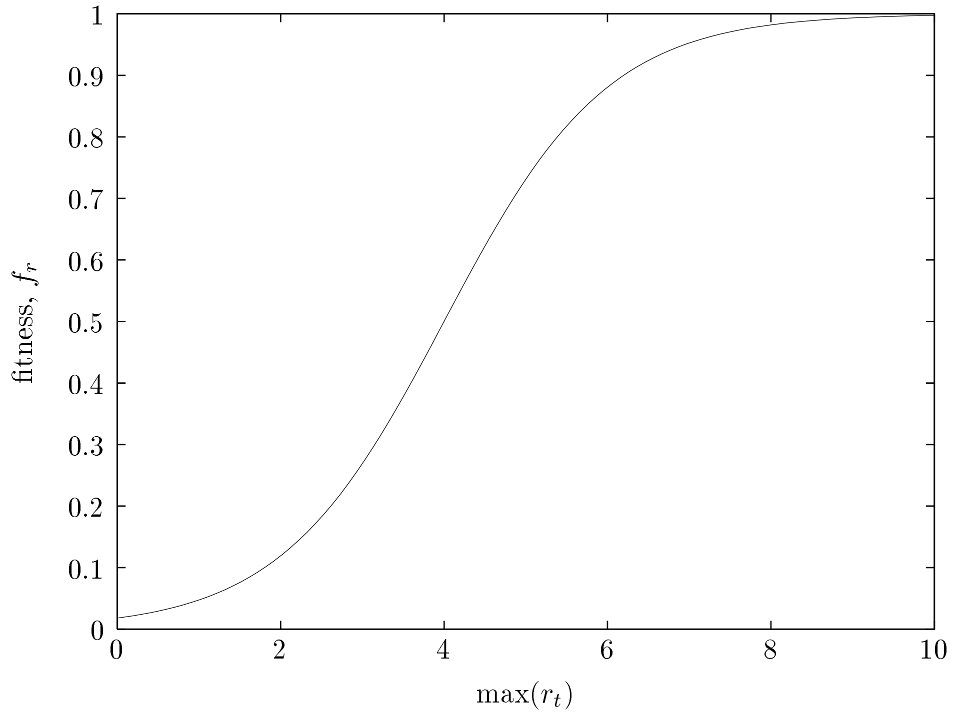


Figure 4.14: Sigmoid scaling function for isolated replicant fitness measure.

100 (keep in mind the large rule table sizes derived in Chapter 3). The main limitation was the memory capacity and of the computers used and the enormous run times required. Given this constraint, the number of generations $g_{\max}$ to run was arrived at experimentally, where it was seen that most GAs were converging between generation 200 and 800. A value of 2000 was chosen to allow for the rare cases in which the GA converged late.

Parameter	*Value(s)*
n_c	100
p_c	0.6–0.8
p_m	0.08–0.10
$g_{\max}$	2000

Table 4.3: GA parameter values used in experiments.

4.7 Multiobjective Optimization

Multiobjective optimization involves the optimization of two or more independent criteria that must be combined into a single value. In the context of the fitness calculation of F (Equation 4.16), it is desired to optimize F by finding an ideal weight vector $\mathbf{w}$ to weight the independent fitness measures f_g, f_p, and f_r. A second GA, called a *meta-level* GA [Grefenstette86], was used to find this weight vector. Thus, under the control of the meta-GA, the primary GA (as described in this chapter) was executed repeatedly. Being that the primary GA was extremely resource intensive by itself, to experiment with the meta-GA required that smaller GA parameters be used.

The meta-GA is a variant of the traditional GA [Davis91], and here we present a brief summary of it. The encoding choice for the chromosomes used was 7-bit Gray-coded binary strings. The first seven bits represent w_1 and the subsequent seven bits encode weights w_2 and w_3 as follows. Let d represent the decimal value of the latter seven bits of the chromosome. Weight w_2 is expressed as

$$w_2 = (1 - w_1) \cdot d \tag{4.23}$$

Then to obtain the third weight, we have

$$w_3 = 1 - (w_1 + w_2) \tag{4.24}$$

Given the above encoding method, a population size of 20 7-bit chromosomes was chosen. As mentioned earlier, computer resources limited the size of the populations that were feasible to run. After decoding each chromosome into a weight vector $\mathbf{w}$, the "primary" GA is run using the decoded weight vector. The fitness function employed for the meta-GA was the f_r fitness measure described in section 4.5.4.4. Thus if a "primary" GA run was able to find isolated replicants, this would give high fitness to a specific weight vector, which would then be bred into the next population of the meta-GA.

Chapter 5

Automated Discovery of Self-Replicating Structures

This chapter presents the results and analysis of the experiments involving automatic discovery of self-replicating structures in both cellular automata and effector automata models. Using the genetic algorithm designed in Chapter 4, experiments were conducted which show, for the first time, that self-replicating structures can be automatically produced. Representative samples of discovered self-replicating structures are shown using varying seed sizes, The amount of self-replicating structures discovered through the experiments described in this chapter, is shown to be statistically significant. Performance graphs showing the behavior of the genetic algorithm are presented and give further insight into how the fitness measures work effectively in searching the space of rule tables. Since this is the first work to produce hundreds of self-replicating structures, a new qualitative classification system is devised to categorize the behavior of self-replicating structures. The experiments themselves are discussed in Section 5.1 and the results are presented in Section 5.2. The chapter concludes in Section 5.3 with a discussion of the software system that was used to conduct the experiments.

5.1 Experiments

In order to understand the results presented in this chapter, the goals of the experiments and the experimental method are briefly described. The primary goals are as follows:

1. The fundamental goal is to show that it is possible to automate the process of creating self-replicating structures in three cellular space automata models. To accomplish this, it must be shown that statistically significant quantities of such structures were produced. If such a goal is achieved, the experiments should allow investigation into how to increase the numbers of discovered self-replicating structures.

2. Given the vast search spaces and unknown fitness landscapes of the cellular space models studied, another goal is to gain an understanding of what impediments exist when searching for self-replicating structures. This involves issues concerning the genetic algorithm design, choice of seed structure, and cellular space model.

3. With sufficient quantities of self-replicating structures discovered, a third goal is to analyze the underlying processes of self-replication from quantitative and qualitative perspectives.

An overview of the technique that was used in performing the experiments reported in this chapter is depicted in Figure 5.1. As mentioned earlier in Section 2.3, the genetic algorithm is a stochastic search method that is not guaranteed to converge to the global optimum. Therefore, the approach taken here is to execute numerous independent GAs, and use statistical methods to analyze the results of the set of GAs. As it is used here, one "experiment" is taken to be a set of 100 *trials*, with each trial being an identical GA except that the stream of random numbers differs from one instance to the next. The top box of Figure 5.1 depicts the common inputs to all of the independent GAs. While executing, each GA stores the highest-fitness rule table it has ever evaluated, and stops when the convergence criteria (see Section 4.6 on page 73) is met. At that point, the highest-fitness rule table is its output (Figure 5.1, middle). The outcome of each trial is either *success* (a self-replicating structure found) or *failure*. Such a decision must be made by human examination of a subsequent simulation based on each rule table, since the rule table with the highest fitness value may not always conform to the requirements of Definition 4.1. The quantity of self-replicating structures found divided by 100 (trials) is called the *yield*. The goal of a given experiment is to maximize the yield. The computational load of a single experiment is enormous since 100 instances of the GA must be run. Since each GA processes up to 200,000 fitness evaluations, this equates to a total of 20,000,000 possible fitness evaluations needed for a single experiment. To reduce the total execution time needed to run one experiment from weeks to days, software that runs on a parallel architecture was designed and implemented, and is discussed in Section 5.3.

The experiments conducted can be classified according to the size of the seed structure used. With the computational resources available, structures having two, three, and four components were feasible to use in the experimental method described above (running 100 genetic algorithms each with a population of 100 large rule tables presents an enormous computational load). Experiments using four-component structures required approximately one week of dedicated time on a 40-node Alpha-processor farm supercomputer. The limitations on this resource allowed for only three experiments to be conducted using four-component seed structures.

For each seed structure, three cellular space models were used: the Effector Automata (EA) model (introduced in Chapter 3), and two variations of the Cellular Automata (CA) model. We call the CA model using state-sensitive input the "standard" CA model, since it is identical to what has been used in research to date. We include it in our experiments because it is has been studied the most with respect to self-replicating structures and it is desirable to see how it performs compared to the other models introduced in this thesis. The other CA model uses the paradigm of component-sensitive input (a new technique introduced in Section 3.2) which is a method for reducing rule table size by having automata ignore orientations of any neighboring weakly rotation-symmetric cell states. The specific EA model used in the experiments, which uses component-sensitive input, is described in Section 4.1. An experiment using the state-sensitive version of the EA model would have nearly the same computational load as a CA model with state-sensitive input. Given the limitations of the computing facilities available, it was decided to conduct the state-sensitive CA experiments instead of the state-sensitive EA experiments, since the state-sensitive CA is the most researched model for self-replicating systems.

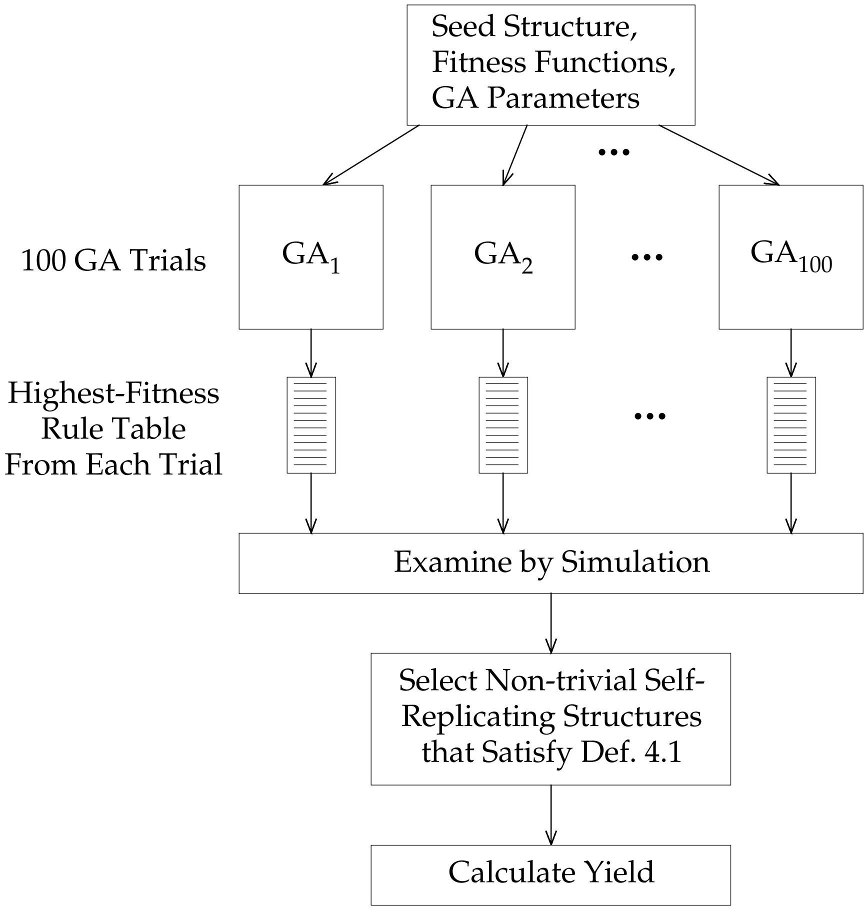


Figure 5.1: Overview of experimental method.

5.2 Experimental Results

With the goal of consistently discovering self-replicating structures in an automatic manner, the most important metric to be taken from the experimental results is the yield – the percentage of self-replicating structures found during an experiment. In this section, we present and analyze the yields found. We find that by using the new technique of component-sensitive input, the highest yields are obtained in the CA model. It is also seen that the "standard" CA model (i.e., having state-sensitive input) is not the best choice for evolving self-replicating structures. The correlation coefficients between the yields and the search space sizes are calculated and suggest a potential correlation between decreasing search space sizes and increasing yields.

5.2.1 Results from Experiments

In Table 5.1, the yield of self-replicating structures found during 100 trials are presented for the cellular space models and seed structures studied. The names "CA_{CSI}" and "CA_{SSI}" denote the cellular automata model using component-sensitive input and state-sensitive input, respectively.

Beginning with the 2-component structures, it is seen that high yields were produced. The

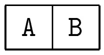
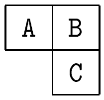
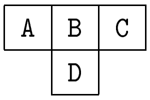
	Yields		
	Model		
Seed Structure	EA	$\mathrm{CA}_{\mathrm{CSI}}$	$\mathrm{CA}_{\mathrm{SSI}}$
	0.43	0.93	0.49
	0.08	0.22	0.03
	0.00	0.02	0.00

Table 5.1: Experimental results showing the highest numerical yields found from each of the experiment conducted.

CA model with component-sensitive input had the most successful results with 93 discovered self-replicating structures. While each of the 93 rule tables are distinct, many of the self-replication processes were qualitatively similar. The other two models show comparable yields of 0.43 and 0.49, however the qualitative behavior of the EA structures is more diverse than that of the $\mathrm{CA}_{\mathrm{SSI}}$ structures. The reason this occurs is due to the fact that each CA rule in this model can transition to one of $k = 3$ states, while each EA rule can transition to one of $|A| = 210$ possible actions, allowing for a more diverse rule table.

For the 3-component experiments, it is seen that the CA model with component-sensitive input again had the highest yields, followed by the EA model. The state-sensitive input CA model had a yield of 3%, which is shown, in Section 5.2.4, to not be considered statistically significant at the 95% level of significance. Three-component yields were lower than that for 2-components, suggesting that the discovery process is more difficult for larger structures. This agrees with the intuition that self-replication process for 3-component structures is more complex than for structures having two components.

In the 4-component experiments, the $\mathrm{CA}_{\mathrm{CSI}}$ model had the only non-zero yield. Although not considered statistically significant at the 95% significance level, it is of interest that the GA was able to discover 2 self-replicating behaviors in only the $\mathrm{CA}_{\mathrm{CSI}}$ model, the model which gave the best results in the other experiments.

These results suggest that by using the component-sensitive input paradigm, higher yields of self-replicating structures are discovered. For 3-component structures, it is also seen that the EA model produced more self-replicating structures than that of the "standard" CA model, $\mathrm{CA}_{\mathrm{SSI}}$ (the same model commonly used in the CA literature). Such a result implies that the EA model is advantageous with respect to the automatic discovery of self-replicating structures.

One of the potential reasons why the highest yields occurred in the $\mathrm{CA}_{\mathrm{CSI}}$ model is that it has the smallest search space size of the three models, and thus the GA may have a slightly easier search task. Table 5.2 presents the search space sizes $|D_n^k|$ derived in Chapter 3 for the experiments conducted. The search space sizes are enormous in all instances, however we note the that relative size differences are also extremely large. To quantify the correlation between increasing yields and decreasing search space sizes, we calculate the sample correlation coefficient r between corresponding rows of Tables 5.1 and 5.2. Values of r are shown in Table 5.3, and all are seen to be negative, indicating that as the search space search increases, the yields decrease. However a value of $r = -1$ would be needed to show a strong correlation of this type. With values of r ranging between -0.2 and -0.5 we can posit that there is some degree of correlation, but not strongly so.

	Search Space Size, $\lvert D_n^k \rvert$		
Seed	*Model*		
Structure	EA	CA_{CSI}	CA_{SSI}
A B	10^{376}	10^{177}	10^{14110}
A B C	10^{1783}	10^{933}	10^{103454}
A B C D	10^{5806}	10^{3279}	10^{436864}

Table 5.2: Approximate search space sizes for the experimental results.

Seed Structure	r
A B	-0.237
A B C	-0.406
A B C D	-0.499

Table 5.3: Values for the sample correlation coefficient r used to measure the correlation between search space size and experimental yields. Negative values indicate that as the search space search increases, the yields decrease.

5.2.2 Discovered Structures

This section presents representative samples of the automatically discovered self-replicating structures. A naming convention is established so that each structure can be given a unique name and the underlying cellular space model can be easily identified. Structures are then presented divided according to the underlying cellular space model used. Self-replicating structures in cellular automata models are presented first, followed by structures in effector automata. For convenience, discussion of the behaviors of the self-replicating structures is placed in figure captions. Appendix C contains a small archive of further self-replicating structures.

5.2.2.1 Naming Convention

In order to identify the self-replicating structures presented here, a naming system from the literature is adopted. In [Reggia93], self-replicating structures in cellular automata are uniquely identified using the following convention. Names begin with the type of loop the structure forms (SL, sheathed loop; UL, unsheathed loop) followed by the number of components that comprise the structure, the rotational symmetry of the individual cell states (S, strong; W, weak), the number of possible states in which a cell may be, and the type of neighborhood (V, von Neumann; M, Moore). For example the structure named `UL06W8V`[1], which appears as

0	0		
L	>	0	0

is an unsheathed loop comprised of six components, has weakly symmetric cell states with each assuming one of 8 possible states, and its transition function is based on the von Neumann neighborhood. This notation may appear somewhat cumbersome, however it is systematic and quite convenient when identifying structures. In Appendix C where numerous self-replicating structures are cataloged, the naming system makes it easy to identify many properties of the cellular space quickly.

The above notation is augmented to allow for the structures studied in this thesis. To distinguish between the techniques of state-sensitive input and component-sensitive input, the letter "C" is added prior to the number of states field to denote the type of input. Because state-sensitive input has been the standard model for studying cellular automata, the notation only changes for the component-sensitive case. The following examples illustrate the difference

`UL3W13V`	state-sensitive input, CA
`UL3WC13V`	component-sensitive input, CA

The above examples were for the CA model. To accommodate EA models, which, by definition have weak rotational symmetry, we allow the rotational symmetry field to contain an "E" to denote effector automata. To illustrate:

`UL3WC13V`	CA, weak rotational symmetry
`UL3EC13V`	EA, (weak rotational symmetry by definition)

The last modification needed is the manner in which identical structures having different rule tables may be distinguished. The convention we adopt is to subscript the name with a number, where the number is arbitrary and only for identification purposes. For example, multiple 3-component self-replicating structures in the EA model may be distinguished,

$$\texttt{UL3EC13V}_1, \texttt{UL3EC13V}_2, \ldots$$

[1] A simulation of `UL06W8V` is shown in Figure 2.7 on page 21 of this dissertation.

The experiments reported in this chapter, were performed using three models and three seed structures. The reasons for the choices of particular of seed structures were discussed in Section 4.5.4.1. Table 5.4 shows the structure names along with the seed structures and models.

Seed Structure	EA	CA_{CSI}	CA_{SSI}
A B	UL2EC9V	UL2WC9V	UL2W9V
A B C	UL3EC13V	UL3WC13V	UL3W13V
A B C D	UL4EC17V	UL4WC17V	UL4W17V

Table 5.4: Naming convention as it applies to the seed structures used in experiments.

5.2.2.2 Representative Self-replicating Structures

Representative examples of the self-replicating structures discovered during the experiments appear on the following pages.

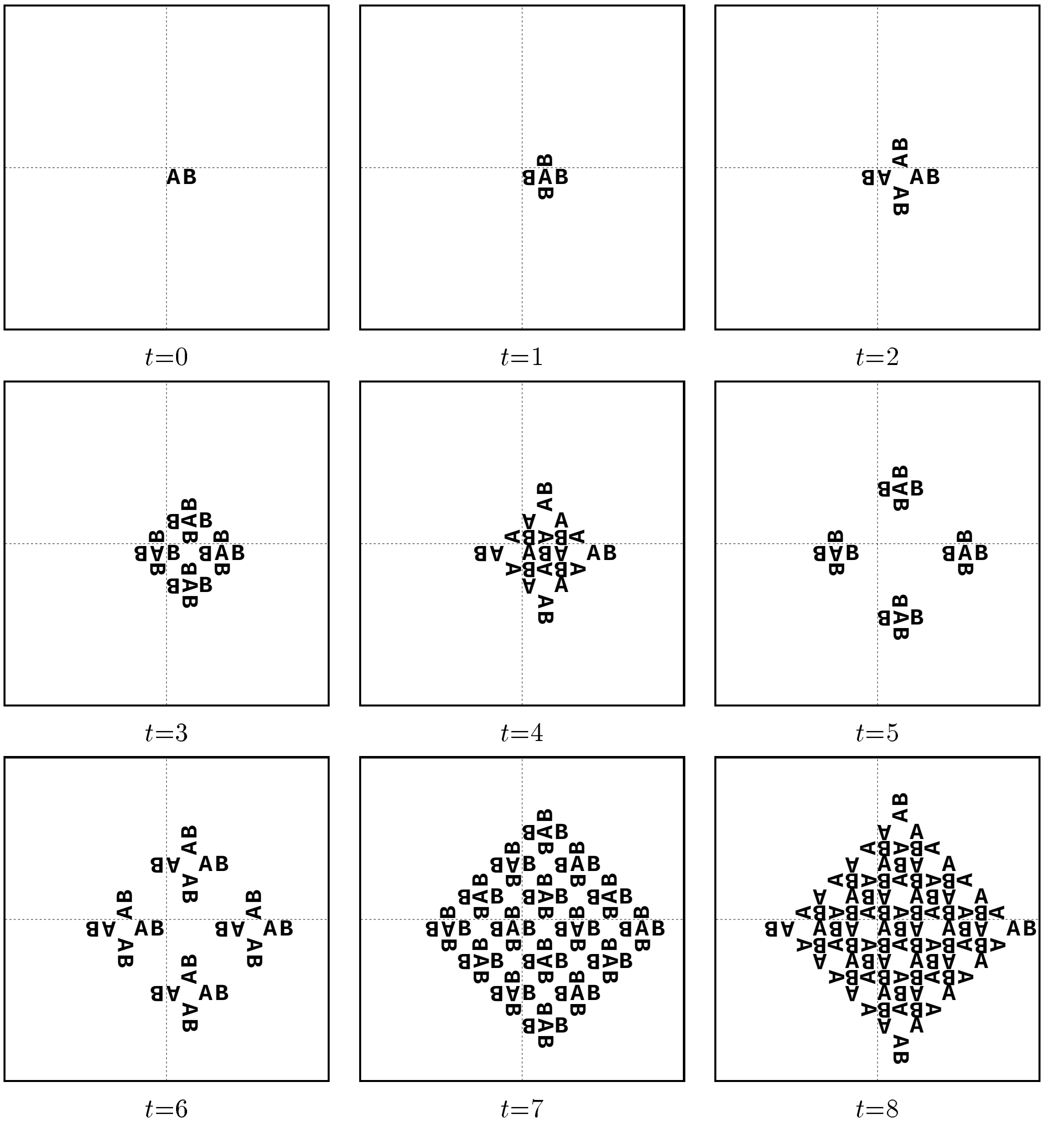


Figure 5.2: Self-replicating structure `UL2WC9V`$_4$. Identical behavior to what is seen here occurred often in the discovered 2-component CA_{CSI} structures. The two-step self-replication process begins at t=1 where the `B` component is seen divided into four copies. At t=2, the `A` component does the same and four replicants are seen. Crowding prevents 8 replicants from appearing at t=4, but then crowding subsides, and 16 replicants appear at t=6.

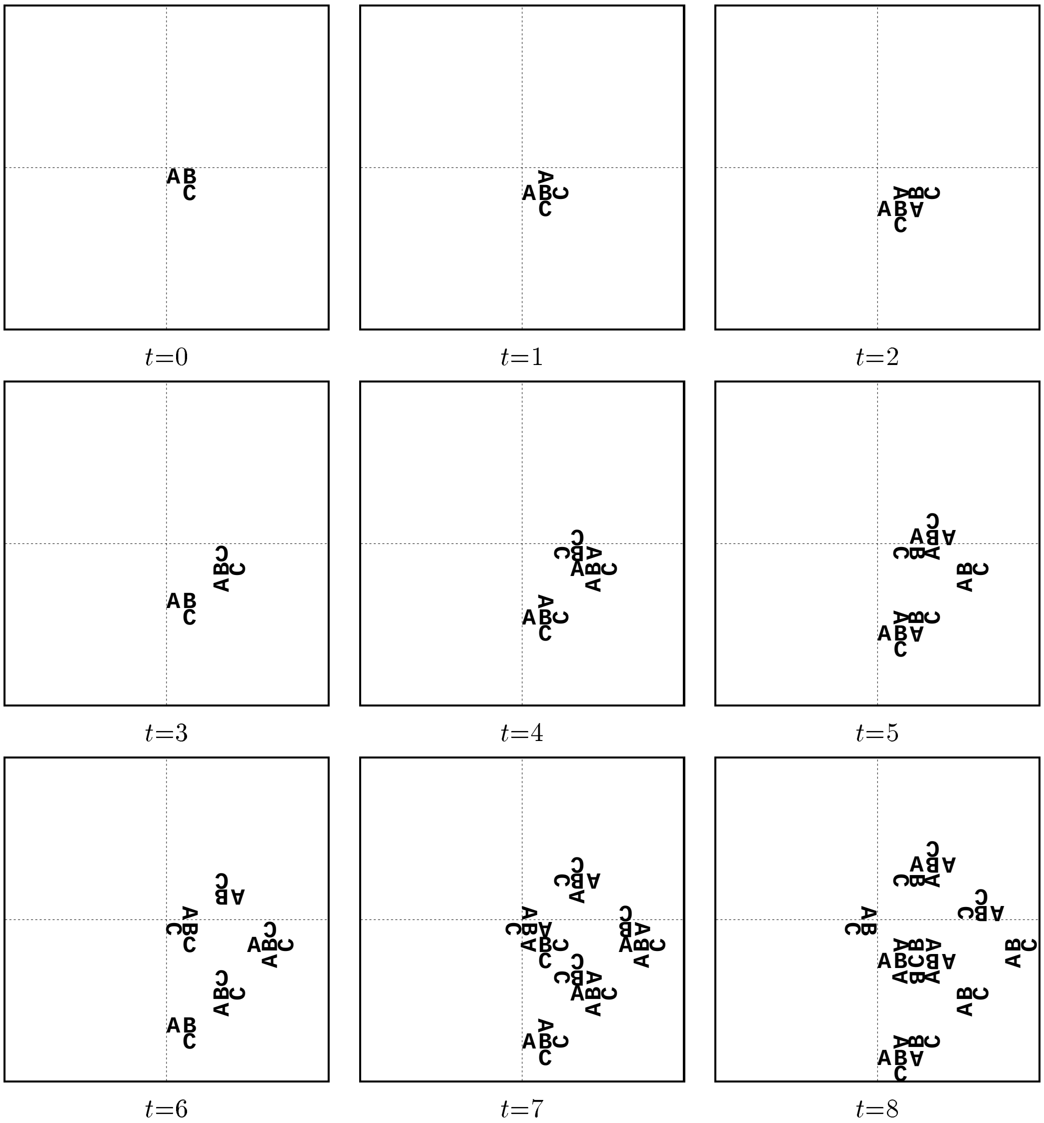


Figure 5.3: Self-replicating structure `UL3WC13V`$_1$. This structure exhibits a 5-step replication process, and begins producing the second replicant (t=4) while the first is still forming. Thus the first isolated replicant appears at t=5 and the second at t=8. The seed structure moves downward over time and those replicants which are rotated move in the other three principal directions.

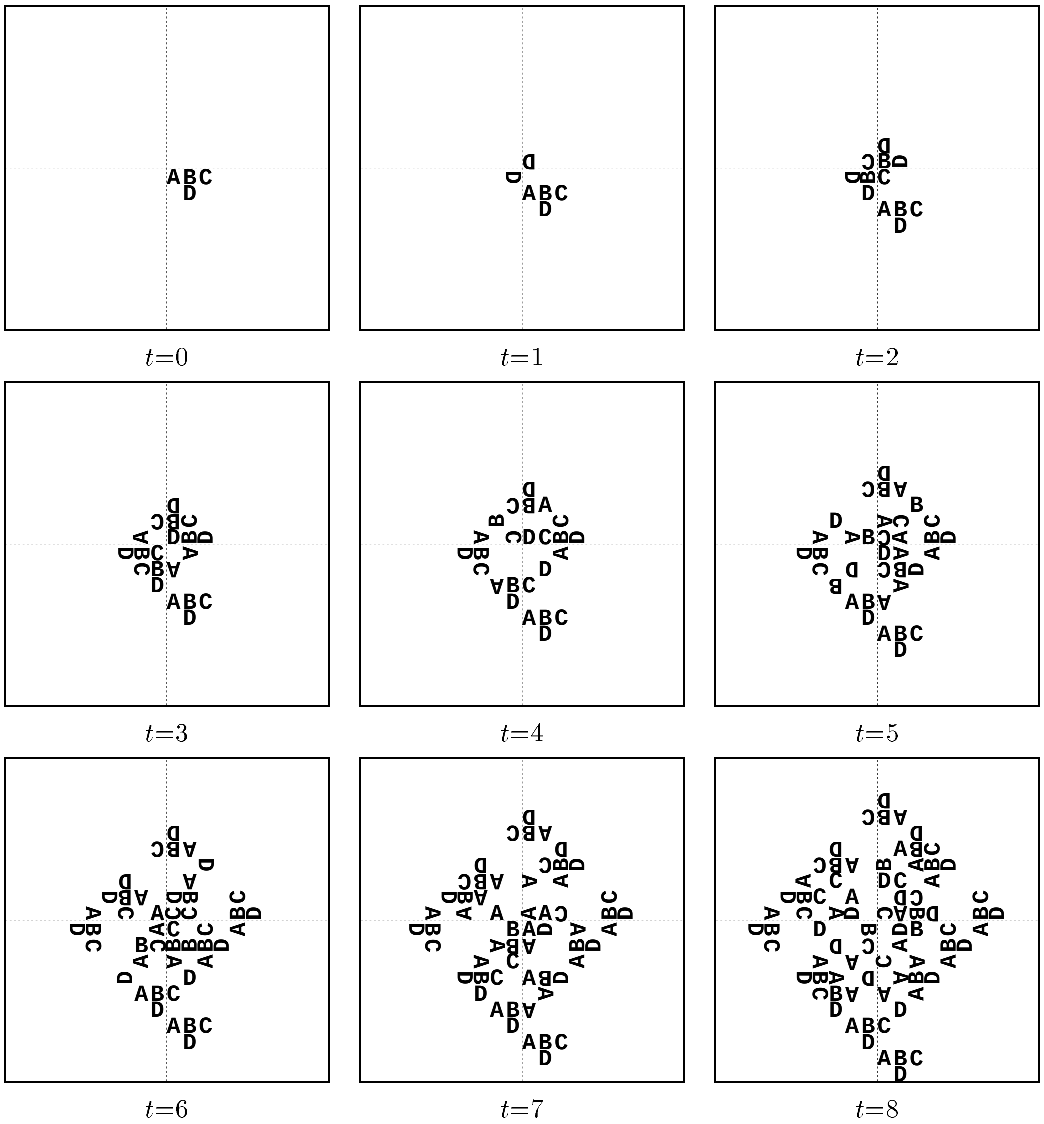


Figure 5.4: Self-replicating structure $\texttt{UL4WC17V}_2$. The 4-component seed structure moves downward over time. Three isolated replicants can be seen at t=5; subsequently, each moves away from the center.

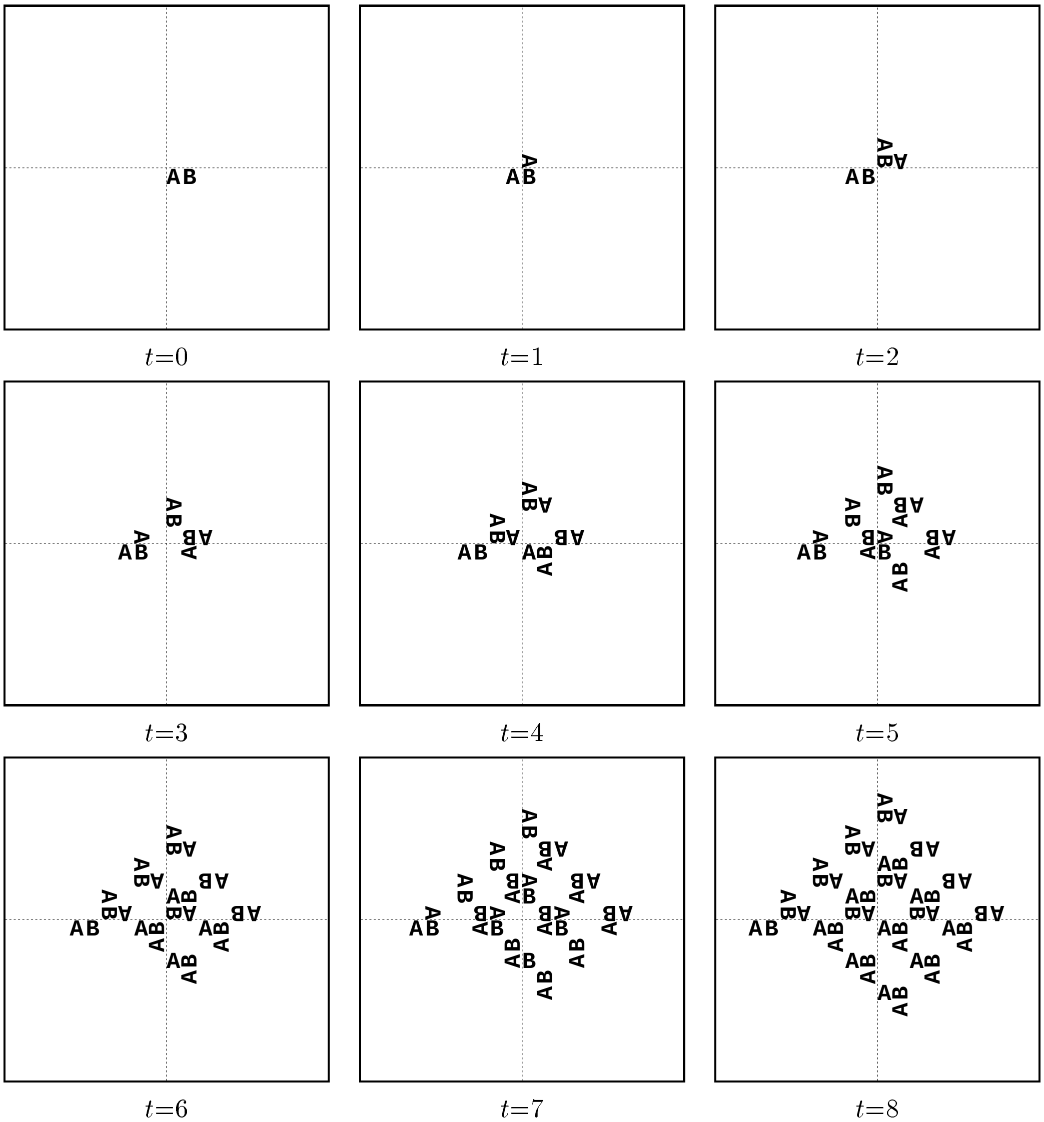


Figure 5.5: Self-replicating structure $\texttt{UL2W9V}_5$. A copy of the seed appears at t=2, which then becomes isolated at t=3, giving a replication cycle of 3 time steps. Replicants are rotated 90° clockwise, and because of this they collide in the center, forming clumps of four unused components as seen at t=8.

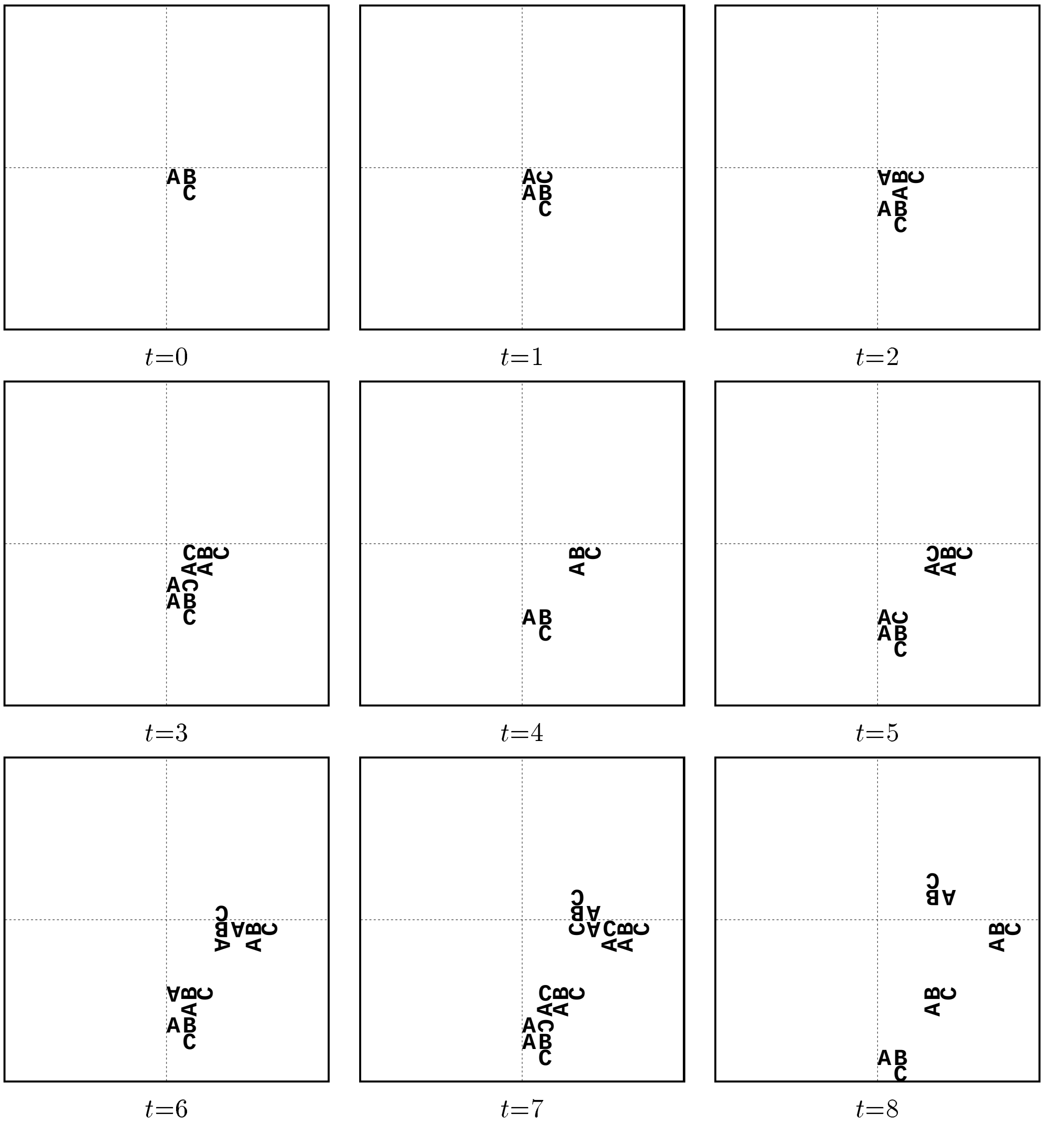


Figure 5.6: Self-replicating structure `UL3W13V`$_5$. The seed structure proceeds downward while producing an isolated replicant every four time-steps. Note that the first replicant is fully formed at t=2, yet not isolated. A unique behavior seen in this structure is the fact that there are no unused components during much of the colony formation (later, the colony collapses in on itself and collisions occur).

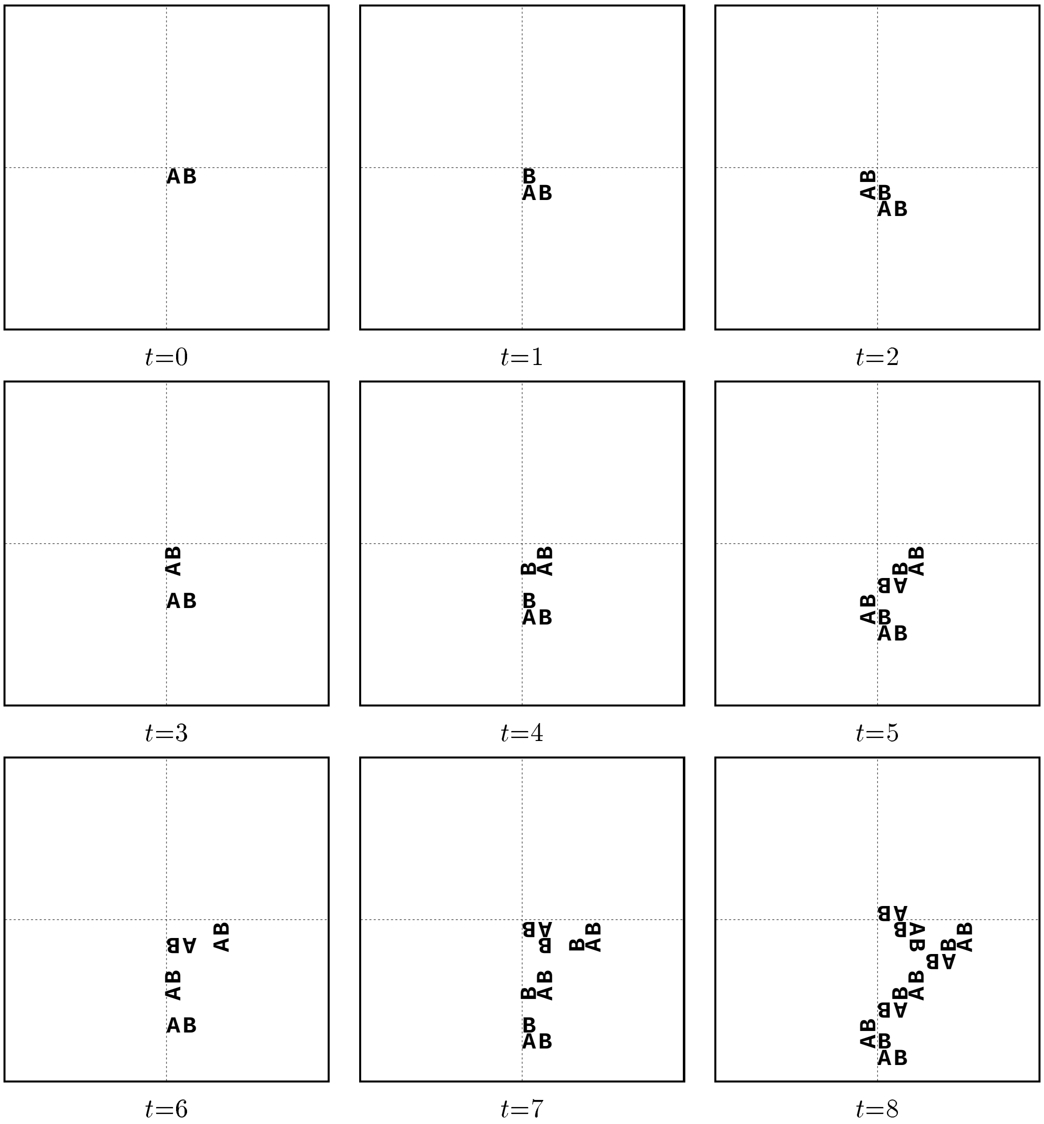


Figure 5.7: Self-replicating structure UL2EC9V$_5$. At t=1 the B component divides into two copies, and the A component does the same at t=2. Compared to other 2-component structures, this structure takes longer than most to produce large numbers of replicants. This is due to the fact that rotation of the replicants is such that the colony collapses in on itself rather quickly.

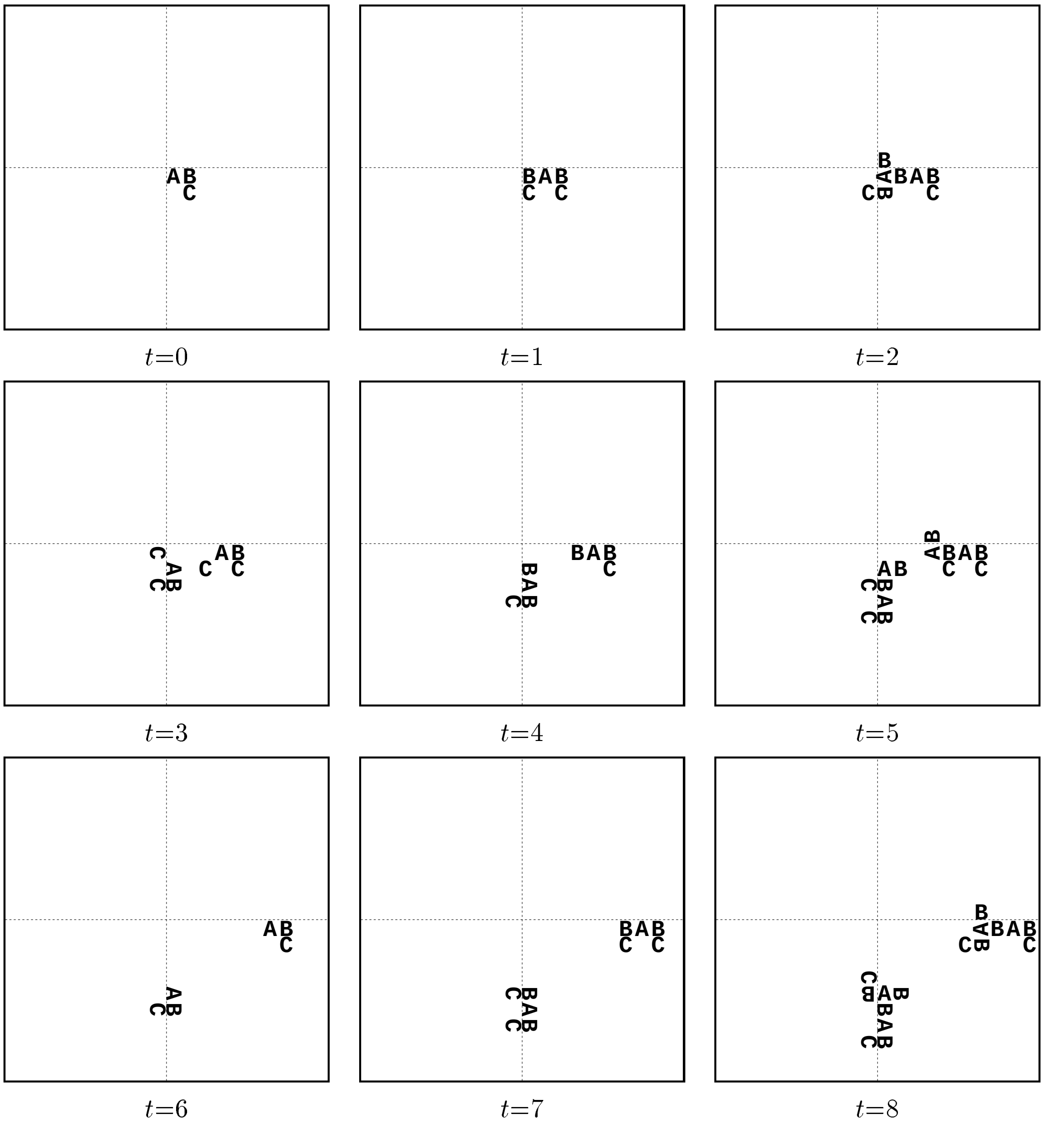


Figure 5.8: Self-replicating structure `UL3EC13V`$_7$. The seed structure moves to the left throughout while producing replicants that are rotated 90° clockwise. Two unused `C` components at t=3 disrupt the self-replication processes of the two structures seen there (by way of collisions). At t=6 there are no such disruptive components, and the two structures seen there continue unimpeded.

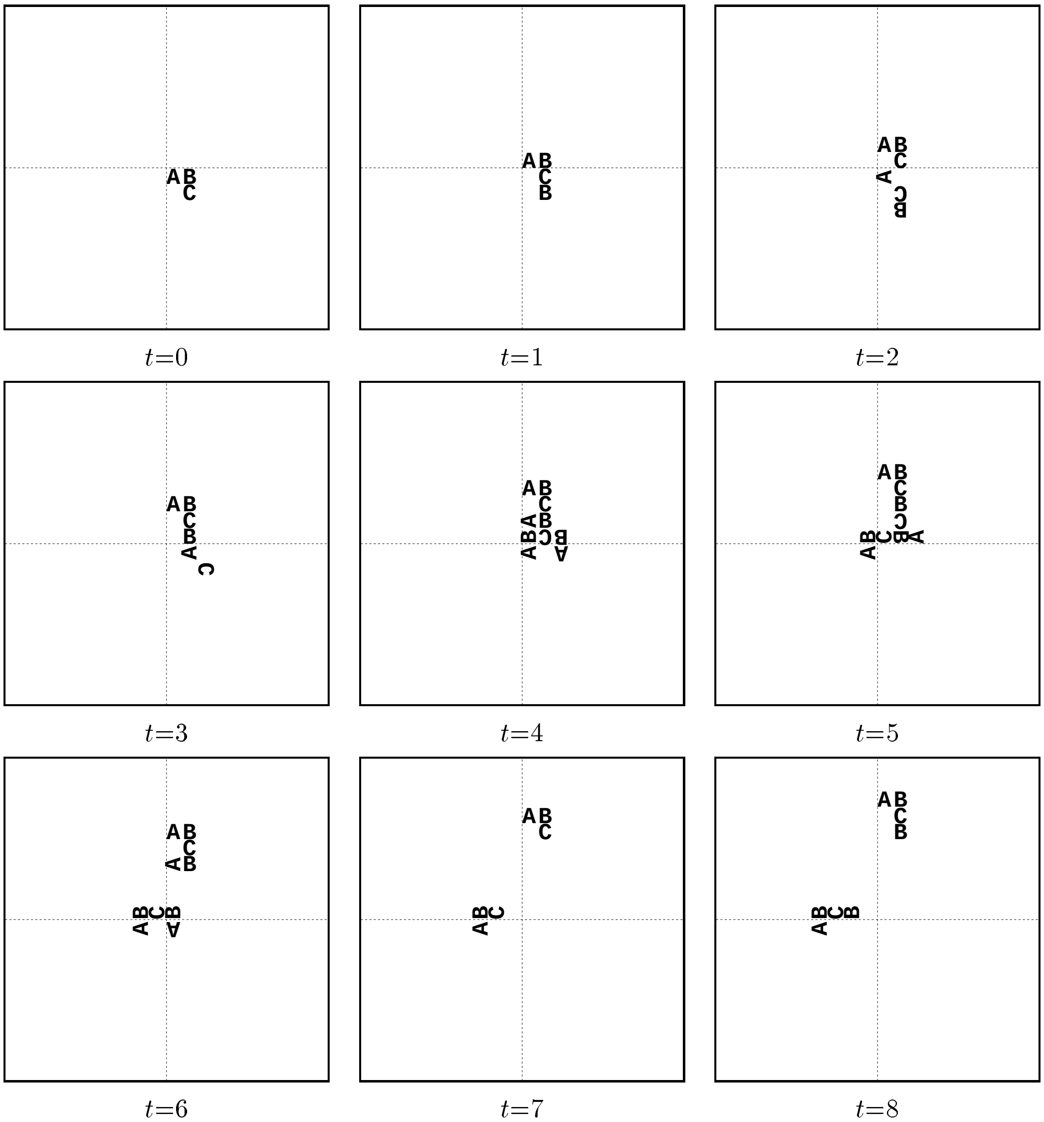


Figure 5.9: Self-replicating structure `UL3EC13V`$_{15}$. The seed structure produces the necessary components for the first replicant at t=2, however it takes until t=5 for these components to form an non-isolated copy. At t=7 the the first replicant is seen isolated.

5.2.3 Other Search Techniques

Experiments have shown that search techniques such as multiple-restart stochastic hillclimbing (MRSH), population-based incremental learning (PBIL) and simulated annealing (SA) are effective in searching large solution spaces using function optimization [Baluja95, Ginsberg93]. This section presents the results of applying two of these algorithms, MRSH and SA, to the task of automatic discovery of self-replicating structures in the EA cellular space model.

MRSH is a method of iterative optimization of static functions which has been successfully applied to standard problems solved by genetic algorithms [Baluja95]. The MRSH algorithm as applied to the task of discovering self-replicating structures is shown in Figure 5.10. Three variations of this algorithm were tried. In the first experiment, a list of rule changes attempted without improvement is maintained. These rule changes are not attempted again until a better solution is found. Recall from Chapter 3 that $|\delta|$ is the rule table size. If $10 \cdot |\delta|$ rules have been tried without improvement, a completely new rule table is randomly generated and the search is continued from there. The second experiment is the same as the first except restart (the process of randomly generating a completely new rule table) is forced 5 times during the search at equally spaced intervals. The third experiment is the same as the second except that if the rule table being tried is better than "or equal to" the best, it is adopted. To be consistent with the experiments using genetic algorithms the number of iterations was set to the population size multiplied by the number of generations ($200 \cdot 100 = 200,000$). One hundred experiments were run each with a different random number seed, for each of the variations of MRSH. One self-replicating structure, shown in Figure 5.12 on page 93, was discovered using the second variation of MRSH.

```
R ← RandomlyGenerateRuleTable
Best ← evaluate (R)
loop NumIterations
   N ← ChangeRandomRule(R)
   if (evaluate(R) > Best)
      Best ← evaluate(R)
      R ← N
end
```

Figure 5.10: Overview of the MRSH algorithm.

The simulated annealing algorithm is similar to MRSH except that at the beginning of the search, new rule tables are adopted almost randomly regardless of whether they are better. As the search proceeds the probability of accepting worse rule tables drops and the probability of accepting better rule tables rises. Simulated annealing derives its name from metal-casting techniques where molten metal is slowly cooled to produce a less brittle product. Likewise, in the simulated annealing algorithm the parameter T is slowly changed so that towards the beginning of the algorithm, the search can proceed in ways that allow it to break away from local maxima by taking steps towards rule tables with lower fitnesses. Figure 5.11 shows the algorithm for simulated annealing. To be consistent with the genetic algorithm experiments, 100 experiments were run using the following parameters: 200,000 iterations, $T_{\max} = 0.2$, $T_{\min} = 0.02$, r = temperature decay rate = 0.8, k =

time per temperature = 768 (rule table size, $|\delta|$). No self-replicating structures were found using these parameters in the simulated annealing algorithm.

1. (Restart) Set $T \leftarrow T_{\max}$. Select a rule table R at random and evaluate it.
2. (Stochastic hillclimb) Create a new rule table N by randomly changing one rule in the current rule table. Select the new rule table N with probability:

$$1.0 - (1.0 + \exp((\text{newfitness} - \text{fitness})/T)))$$

   Repeat this step k times.
3. (Anneal/Convergence Test) Set $T \leftarrow rT$. If $T \geq T_{\min}$, go to step 2. Otherwise go to step 1.

Figure 5.11: Overview of the simulated annealing algorithm.

For the parameters discussed above, these results give an indication that genetic algorithms outperform MRSH and SA for the task of discovering self-replicating structures. One difficulty however in comparing these algorithms is that each is defined by control parameters, and it is prohibitively expensive to thoroughly explore the parameters. Most of the work in this thesis concentrates on using genetic algorithms for the automatic discovery of self-replicating structures, since genetic algorithms have been shown to be particularly adept at finding sufficiently good solutions instead of a global optimum [Mitchell96].

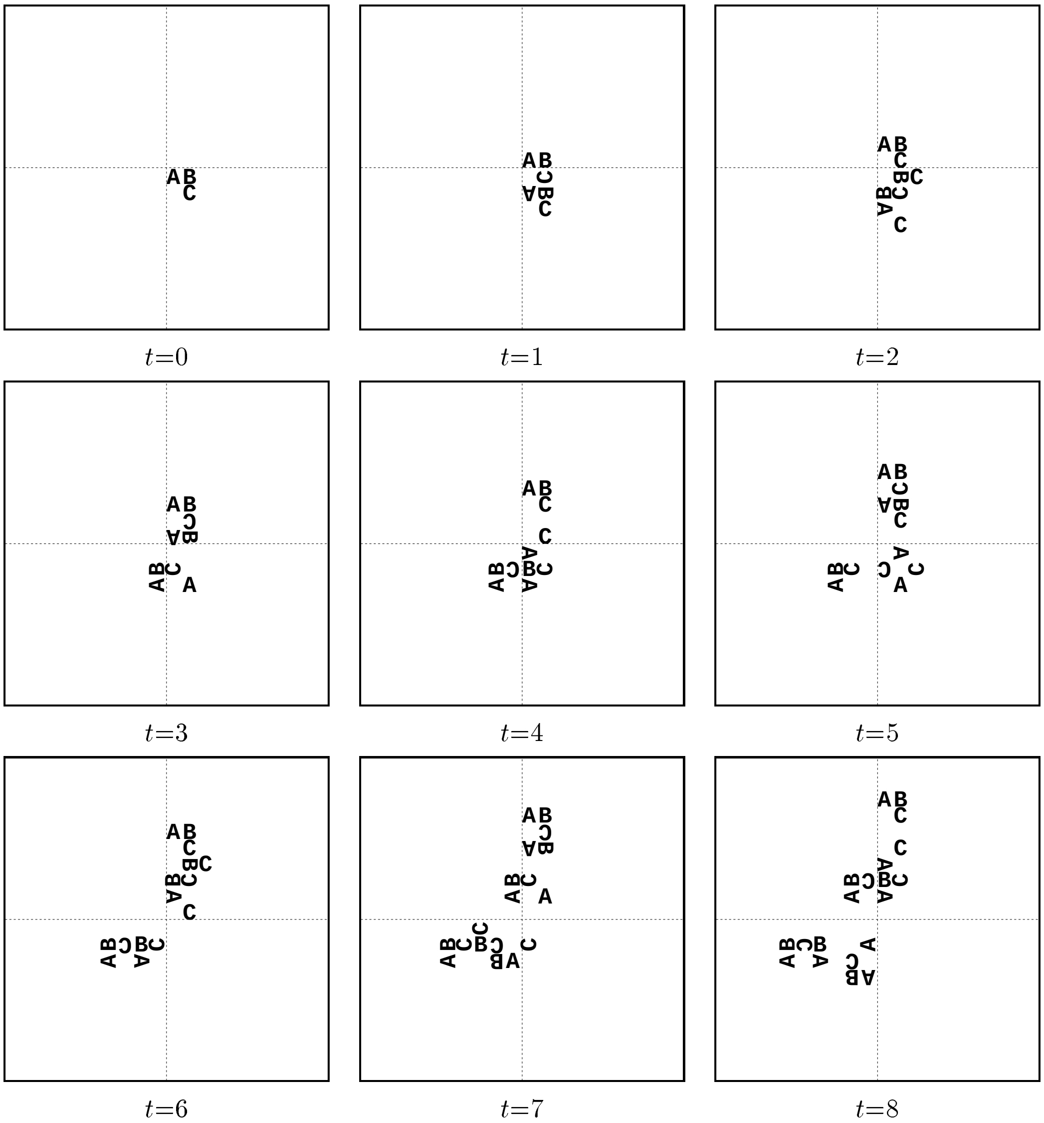


Figure 5.12: Sole self-replicating structure `UL3EC13V`$_{57}$ discovered by multiple restart stochastic hillclimbing (MRSH) algorithm. Original seed structure moves upwards while replicants appear rotated 90 degrees counterclockwise. Two isolated replicants can be seen at $t = 8$. The qualitative behavior of `UL3EC13V`$_{57}$ is quite similar to that of the structures discovered by the genetic algorithm.

5.2.4 Statistical Testing of Results

Sections 5.2.1 and 5.2.2 demonstrate for the first time that it is possible to *automatically* discover self-replicating structures in cellular space automata models. In this section a statistical significance test is presented regarding the yields obtained from the experiments conducted. First, a test to establish that the results from the genetic algorithm are statistically independent from other search techniques is described. Then we use the same test to show that statistically significant yields of self-replicating structures were found in comparison to random search.

In comparing the results from two search algorithms X and Y, it is desired to calculate the significance of the differences in the results. Using the tests described below, we can determine, at the 95% confidence level, when the difference in performance between the two algorithms is significant (the better-performing algorithm is significantly better), or that there is no significant difference between X and Y. Statistics are calculated from the performance data, which are organized into 2×2 tables arranged as shown in Table 5.5.

	# successes	*# failures*
X	a	b
Y	c	d

Table 5.5: 2×2 table for statistics calculation.

The well–known Chi-Square statistic can be used to check for effectiveness only when each cell of the table is greater than three. In other cases, we employ Fisher's Exact Test [Kanji93] with p representing the significance level, and a, b, c, and d are values shown in Table 5.5. The significance level is calculated as follows:

$$p = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{(a+b+c+d)!} \cdot \frac{1}{a!b!c!d!} \tag{5.1}$$

The statistical test is set up using the null hypothesis H_0 which states that X did not influence the results of the experiment. In other words, the number of successes produced by X could have come from either X or Y. The alternative hypothesis H_1 states that there is a statistically significant difference between X and Y. In comparing the genetic algorithm to the MRSH algorithm in Section 5.2.3 we have the following table:

	# successes	*# failures*
GA	8	92
MRSH	1	99

Using Equation 5.1, we calculate p to be 0.017. Since $0.017 < 0.05$ at the 95% significance level, we reject H_0 and conclude there is a statistically significant difference between applying GA versus MRSH.

One of the results of each experiment is the yield of discovered self-replicating structures. An important analytical measure of these results is the statistical significance of the yields obtained. In other words, comparing the yield found using the genetic algorithm in an experiment to the yield found by chance via random search. For every experiment that was run, comparable trials using random search was also tried. In each trial of random search, zero self-replicating structures were produced[2]. Thus we employ Fisher's Exact test (presented above) in the following manner. Let d represent the number of replicants discovered by the GA. Thus the 2×2 table can be written:

	# successes	*# failures*
GA	d	$100 - d$
Random Search	0	100

It is relatively easy to show that when $d = 4$ successes (4 self-replicating structures discovered in 100 trials), p=0.061, and with $d = 5$ successes, p=0.029. Thus a yield of 5 or more self-replicating structures is considered statistically significant at the 95% significance level. For the experimental results presented in Table 5.1 (page 79), it is seen that some yields are not statistically significant at the 95% significance level. For example, in the 3-component experiments, while the 8% yield from the EA model is statistically significant, the 3% yield from the CA_{SSI} model is not. Also, all of the 2-component experiments and none of the 4-component experiments gave statistically significant results at the 95% significance level.

5.2.5 Classification of Self-replication Processes

In this section we introduce a qualitative classification system for the self-replicating structures produced by the experiments described in this chapter. This system is thought to be widely applicable to other 2-D cellular space models using the von Neumann neighborhood and having square tessellations. Such a classification is useful since it draws attention to the many classes of self-replicating behavior that emerged in the experiments, and because it provides evidence that the fitness measures derived in Chapter 4 were not strongly biased towards a single, specific self-replication process. Prior to this research, it would have been difficult, if not impossible to identify a classification system mainly due to the fact that the number of manually-designed self-replicating structures reported in the literature totals less than 30.

The classes of behavior became apparent during observations of animated sequences of the self-replicating structures. Table 5.6 lists the names of the classes and an example structure of each class. The classes shown are divided into two sections: one for *processes* and one for *colonies*. Process classes are distinguished by process of self-replication that the structure exhibits. Colony classes refer to the shape of the formed colony. These classifications are not exclusive – certain structures can be classified into more than one class. The Process classes of behavior for self-replicating structures are described as follows:

- Trivial – characterized by simultaneous splitting of all components to form two distinct copies of the seed structure during the first step in the replication process. A more detailed discussion of this is found in Section 4.3.
- Prolific – a structure produces replicants every 2 time steps.

[2]This is not surprising. For example, assume that there are 10^6 rule tables that promote self-replication in a certain EA model with a search space size of 10^{2000}. Then the probability of finding such a rule table by random search is 10^{-1994} which can be approximated as zero.

- Mass-preserving – the individual components that comprise the parent structure and replicant remain active at each time step in the self-replication process.
- Complex – the self-replication process requires at least 2·(number of components comprising the structure) time steps to self-replicate.
- In-place – during the self-replication process, the structure remains in place, possibly rotating,
- High Density – numerous replicants are produced from the seed structure, but these replicants are unable to self-replicate themselves due to a high density (crowding) of components.

The Colony classes are concerned with the shape the colony forms and are described as follows:

- Linear – the colony forms along a line, expanding outwards in opposite directions.
- Rectangular – the colony forms a rectangular shape.
- Irregular – the colony does not form an identifiable geometric shape.

Process Classes	*Example*	*Page No.*
Trivial	$\texttt{UL3W13V}_{25}$	97
Prolific	$\texttt{UL2EC9V}_{3}$	98
Mass-preserving	$\texttt{UL3EC13V}_{7}$	99
(Non-Mass-preserving)	$\texttt{UL4WC17V}_{1}$	100
Complex	$\texttt{UL3EC13V}_{15}$	101
In-place	$\texttt{UL3EC13V}_{8}$	102
High Density	$\texttt{UL3WC13V}_{5}$	103
Colony Classes	*Example*	*Page No.*
Linear	$\texttt{UL3EC13V}_{21}$	104
Rectangular	$\texttt{UL2E13V}_{21}$	105
Irregular	$\texttt{UL3EC13V}_{5}$	106

Table 5.6: Classes of self-replicating structure behavior.

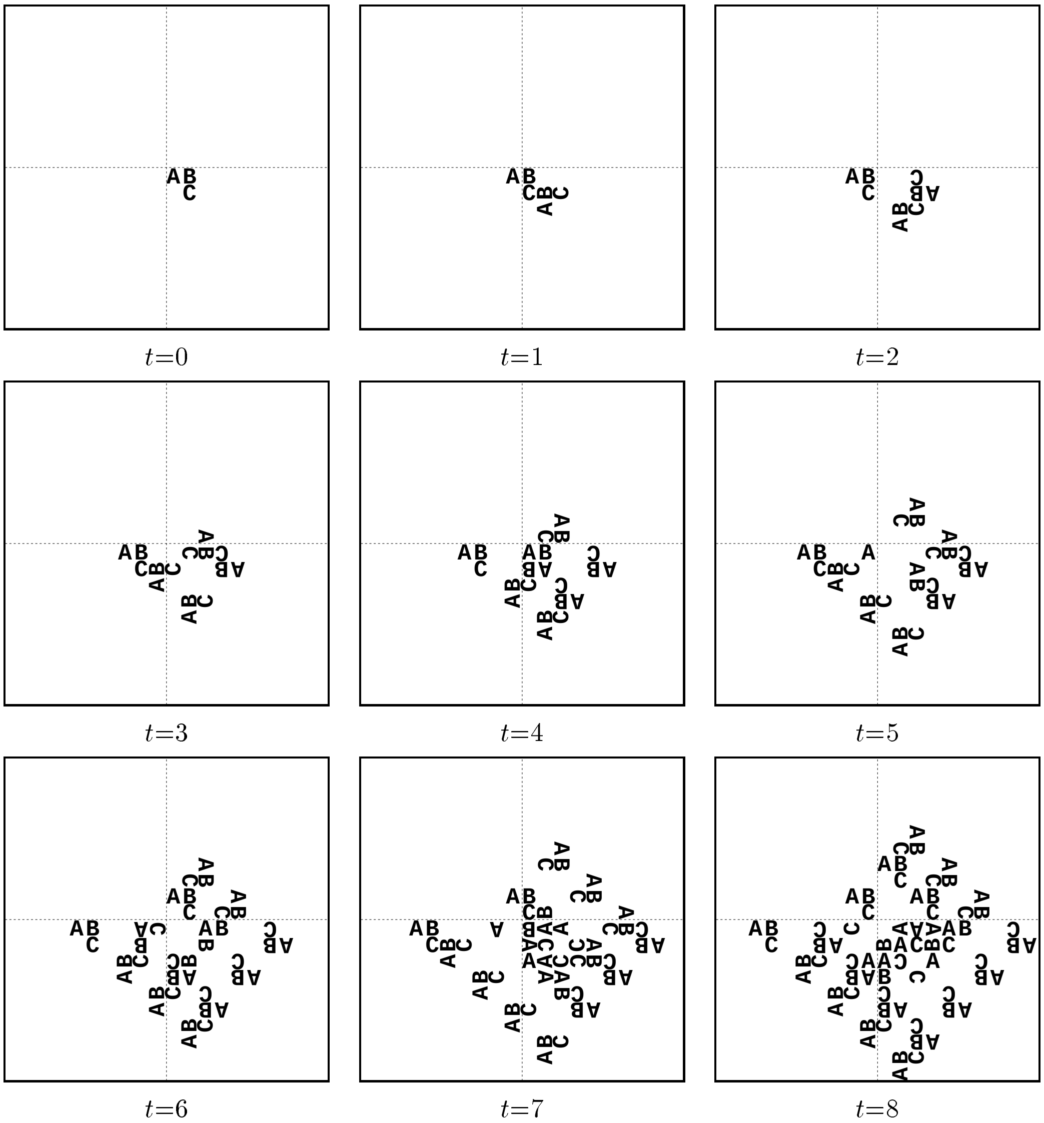


Figure 5.13: Example of "trivial" process class (structure $\texttt{UL3W13V}_{25}$). All components of the seed structure divide simultaneously to form a distinct replicant at t=1.

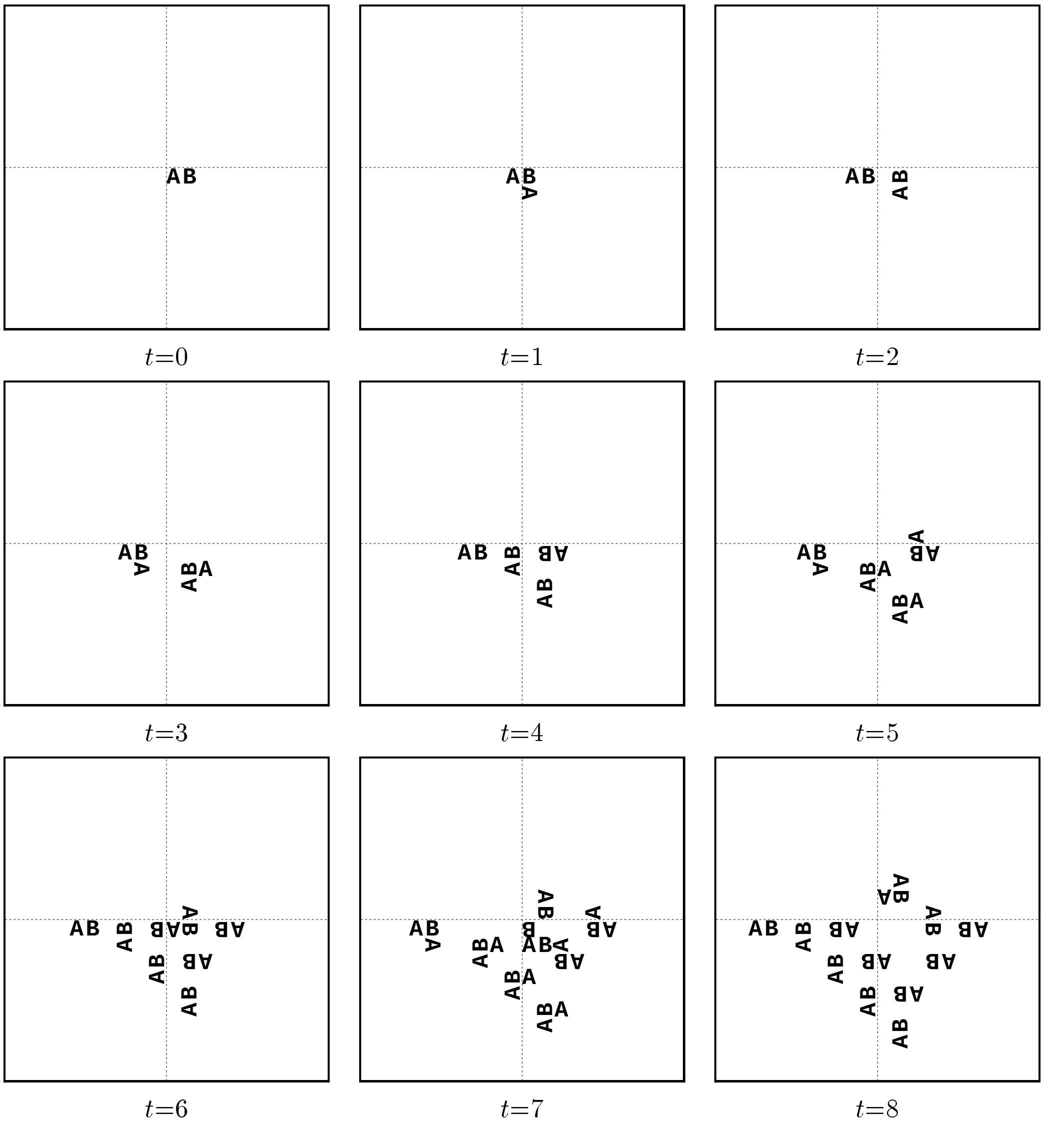


Figure 5.14: Example of "prolific" process class (structure $\texttt{UL2EC9V}_3$). Replicants are produced every other time-step.

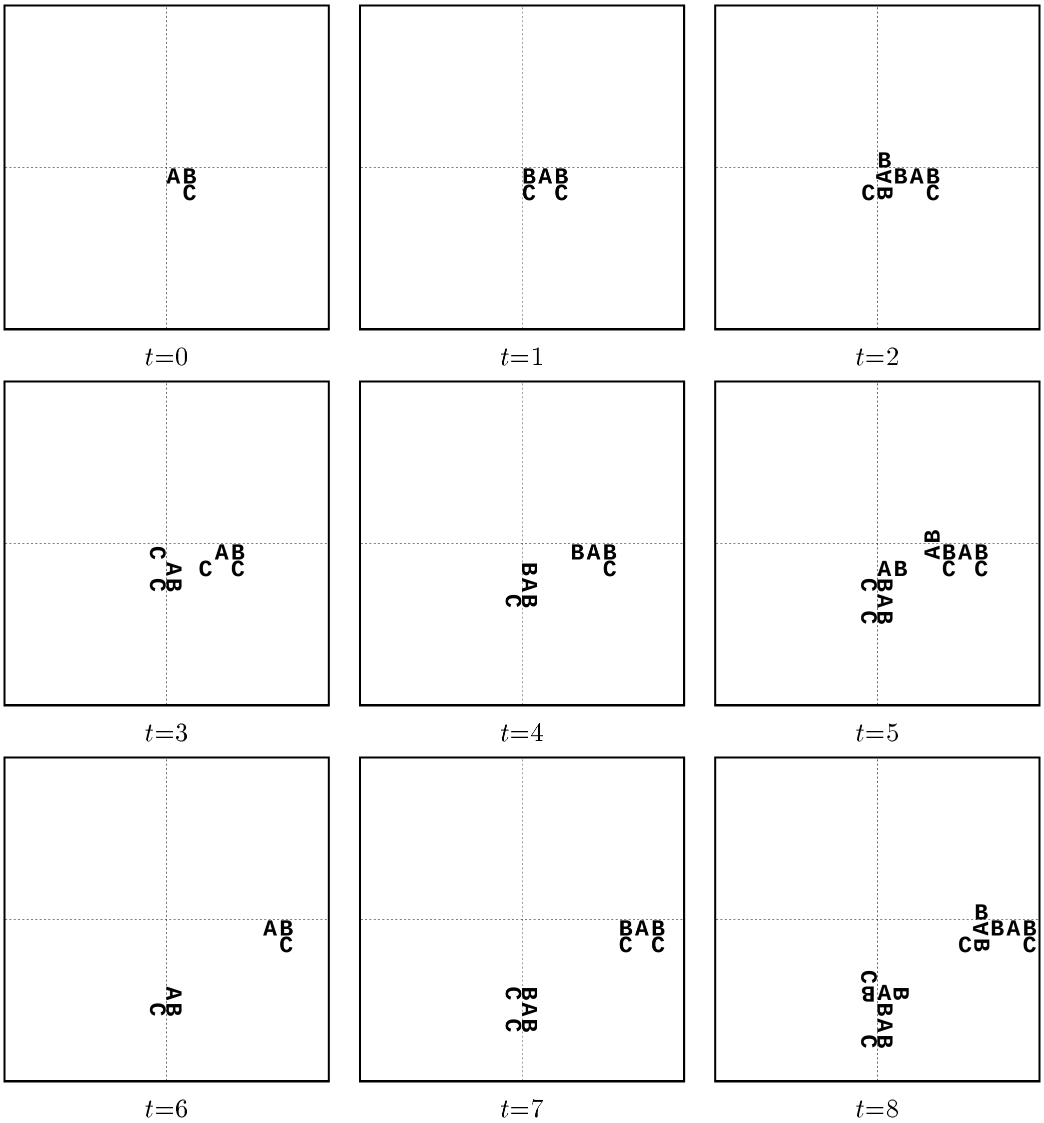


Figure 5.15: Example of the "mass-preserving" process class (structure $\texttt{UL3EC13V}_7$). It is seen that newly-created components persist at each time step during the replication process.

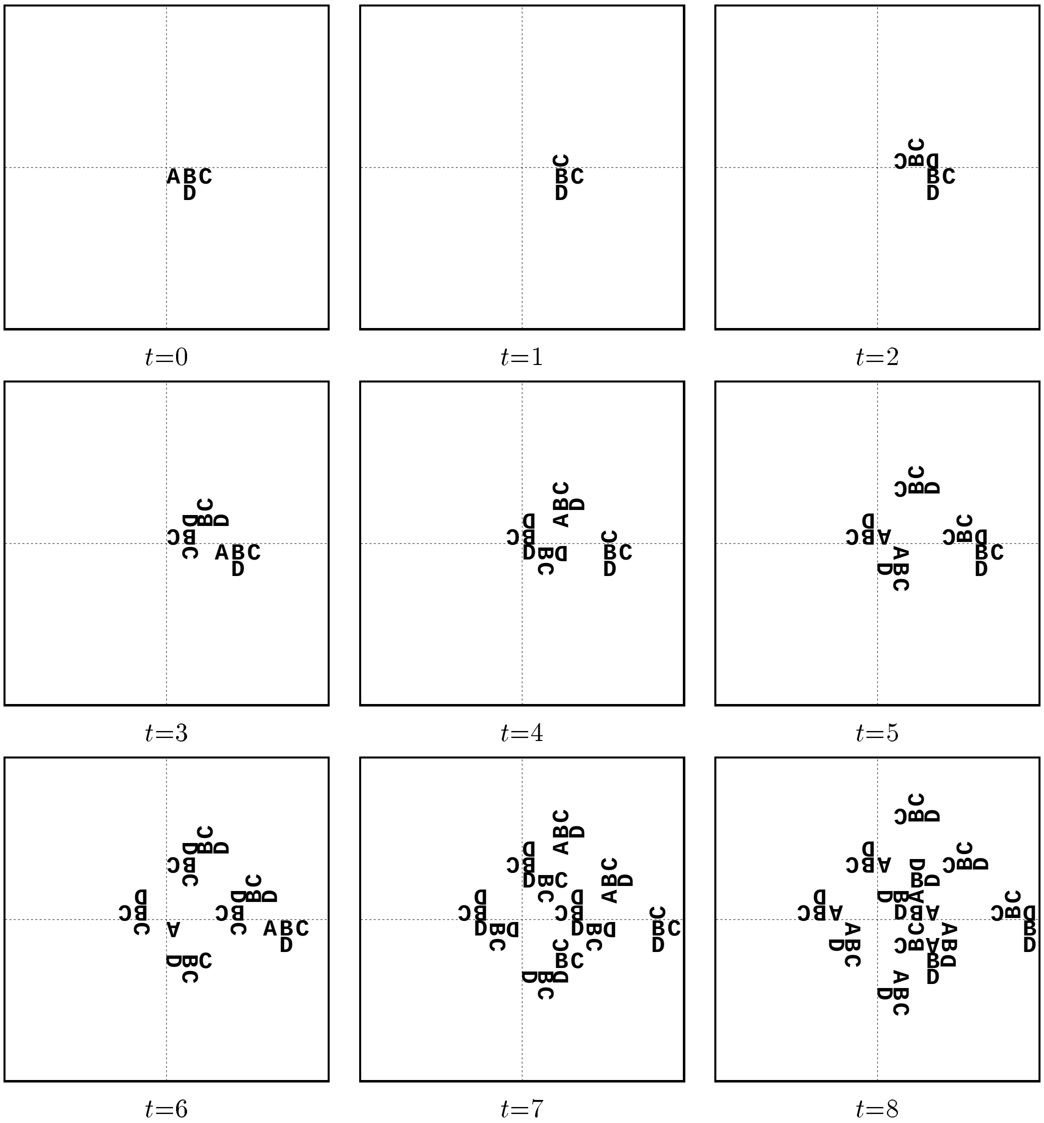


Figure 5.16: Example of "non-mass-preserving" process class (structure $\texttt{UL4WC17V}_1$). At t=2 and t=3 it is seen that the forming replicant lacks an `A` component.

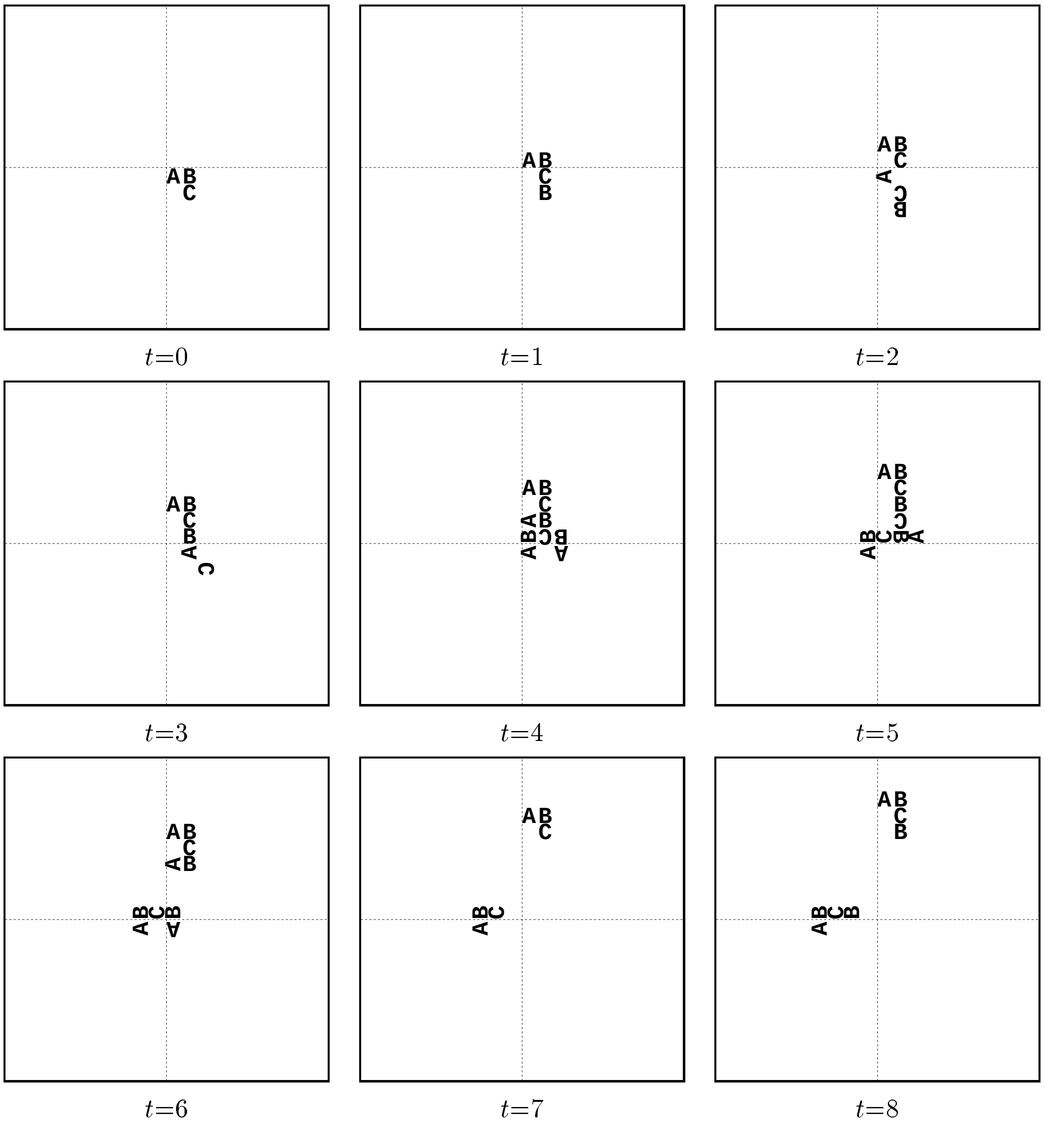


Figure 5.17: Example of the "complex" process class (structure $\texttt{UL3EC13V}_{15}$). The first isolated replicant appears at t=7 indicating a more complex replication process has taken place in comparison to those with shorter replication cycles.

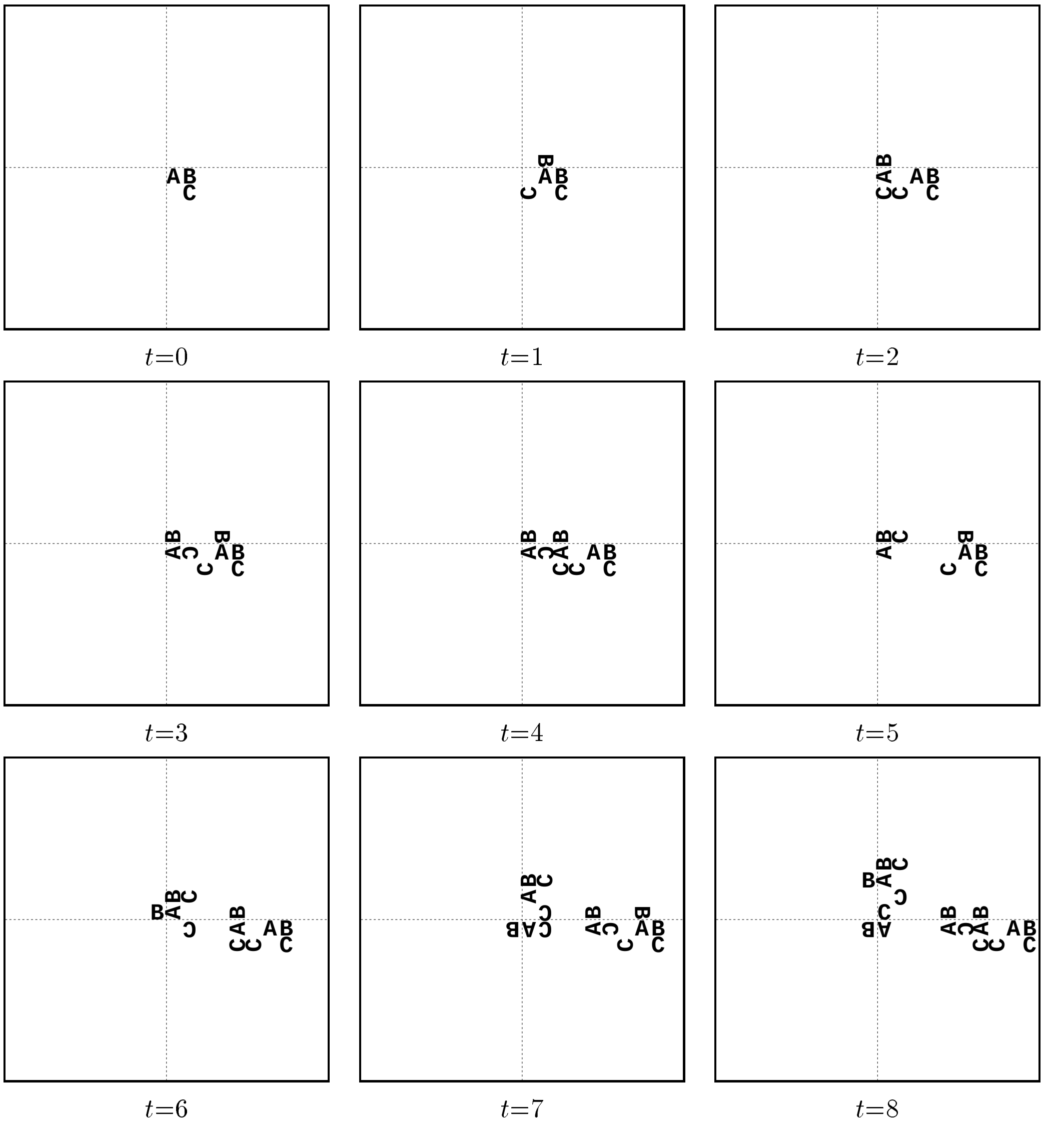


Figure 5.18: Example of "in-place" process class (structure `UL3EC13V`$_8$). The first replicant is constructed at the origin of the coordinate system until t=6 when it begins to move upwards.

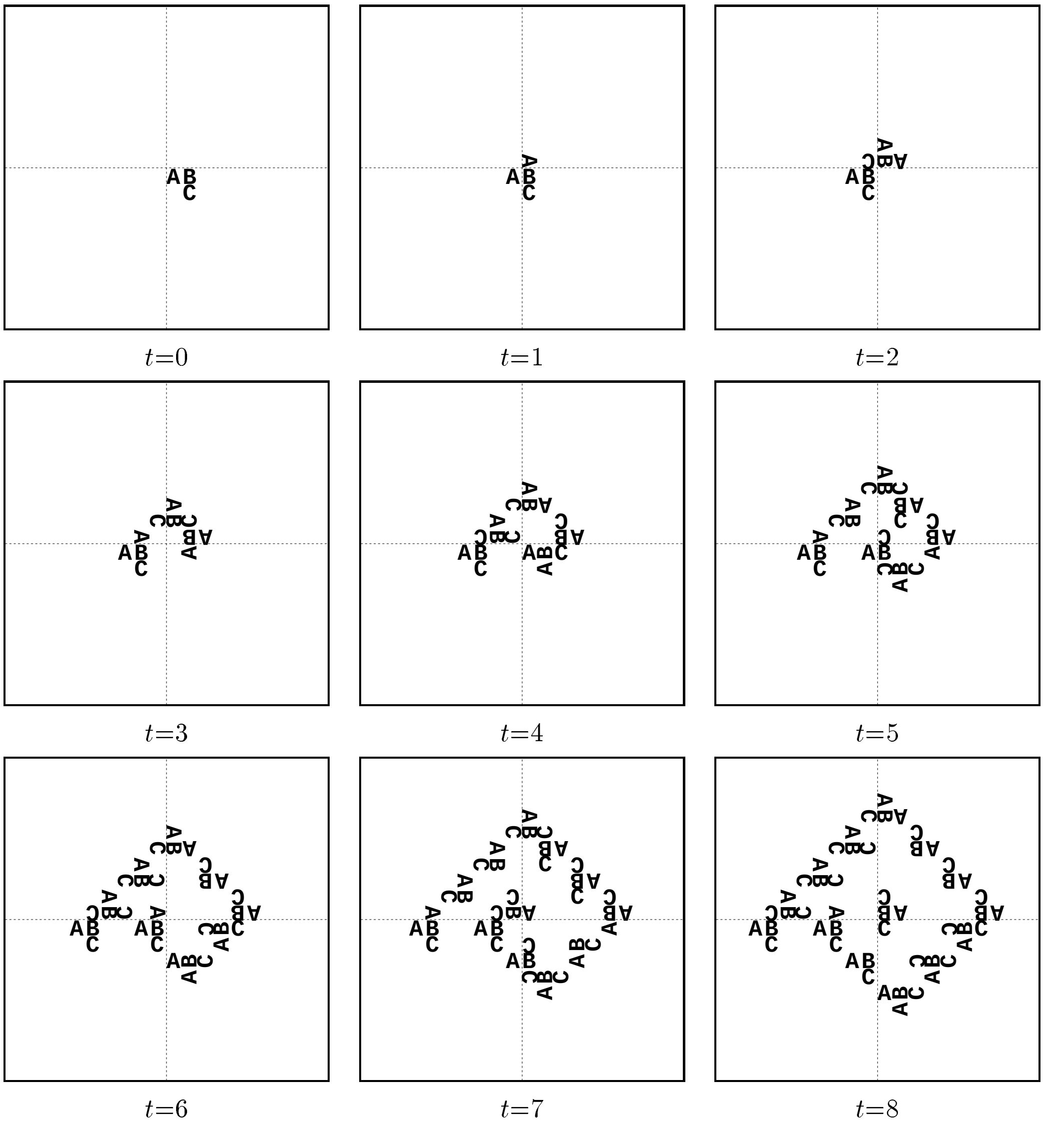


Figure 5.19: Example of "high-density" process class (structure $\mathtt{UL3WC13V}_5$). Starting at t=3, the seed structure produces a new replicant every other time-step. However, these replicants are only able to move and cannot further replicate due to crowding conditions.

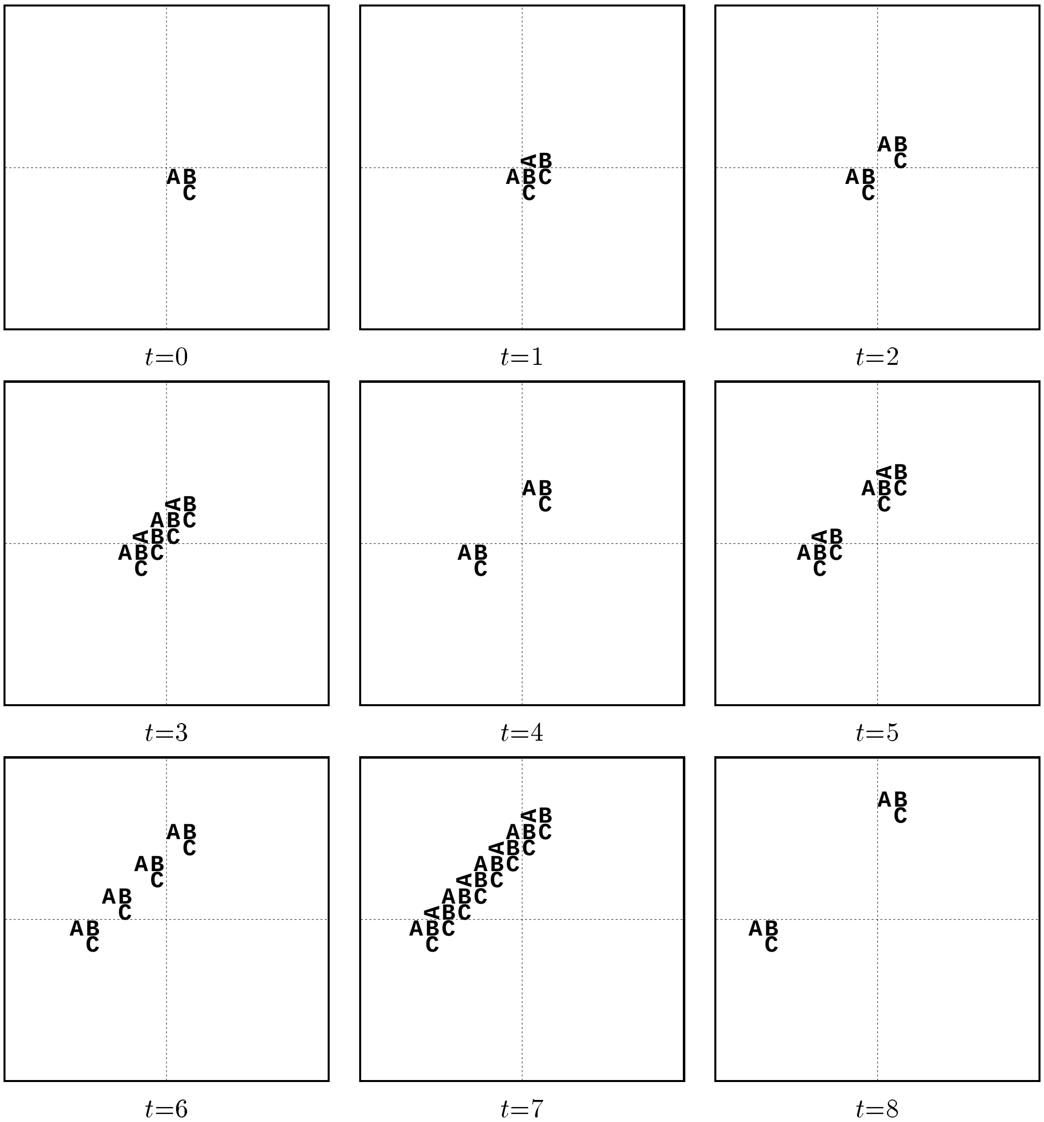


Figure 5.20: Example of "linear" colony class (structure $\texttt{UL3EC13V}_{21}$). The colony expands outward from its center, while the entire colony simultaneously moves towards the upper left.

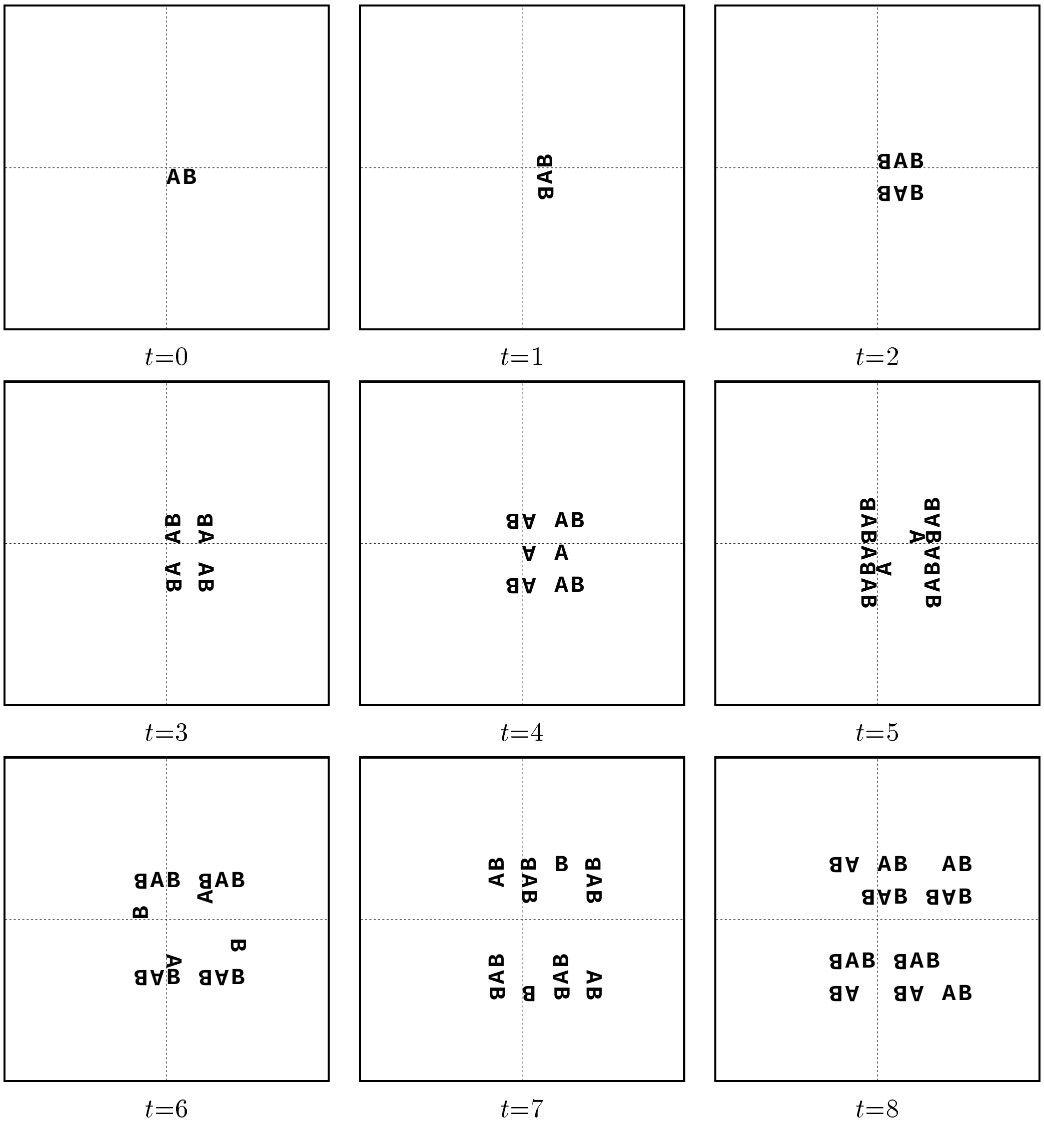


Figure 5.21: Example of "rectangular" colony class (structure $\texttt{UL2EC9V}_7$). The four replicants produced at t=3 define the rectangular shape of the colony, and subsequent time steps show the colony's expansion.

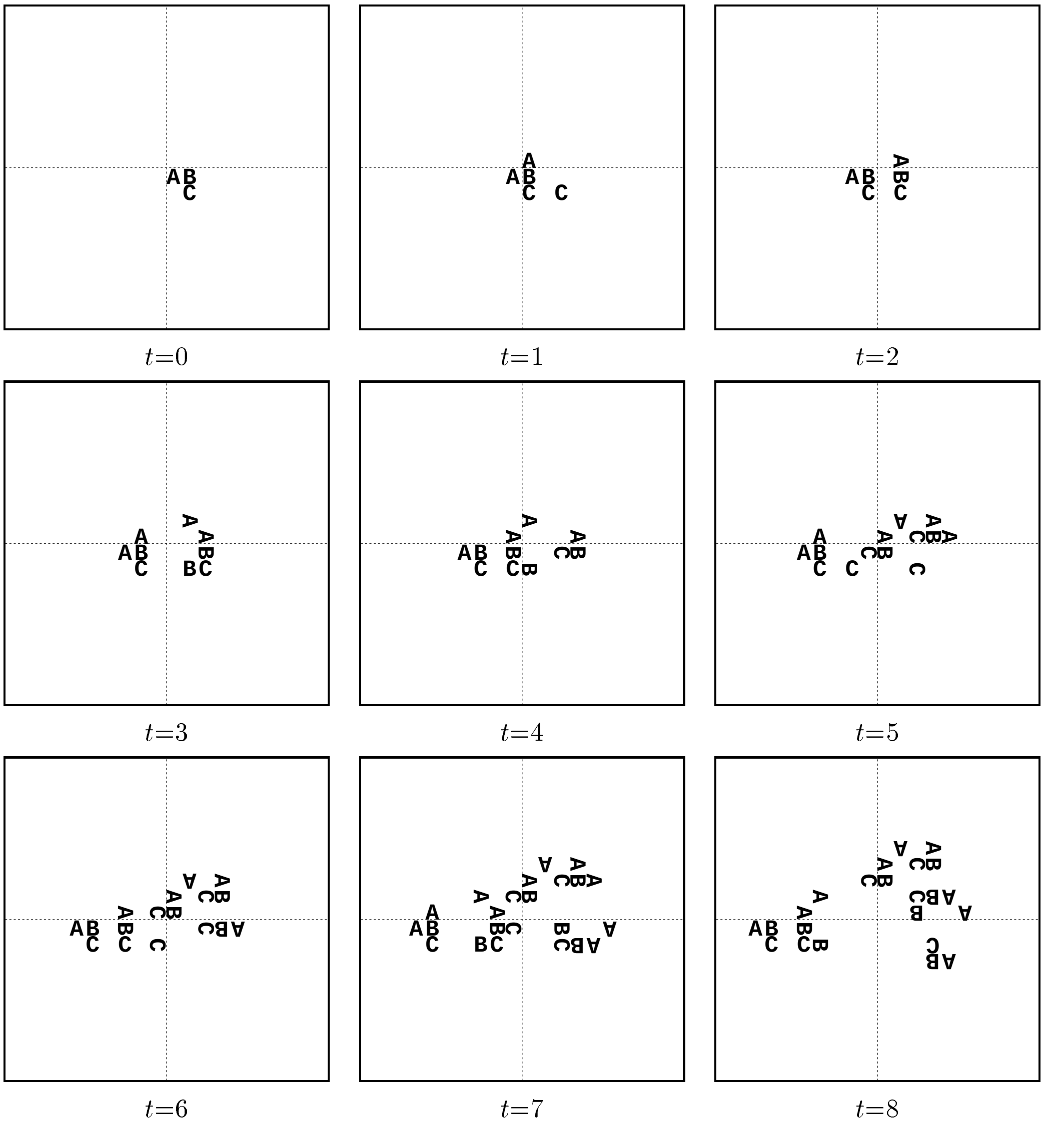


Figure 5.22: Example of the “irregular” colony class (structure `UL3EC13V`$_5$). The seed structure moves leftward and the first replicant appears at t=4. The colony shape produced is not easily classified as one particular geometric shape.

5.2.6 GA Performance

In the context of genetic algorithms, a *performance graph* [Davis91] is a plot of fitness versus generations. To gain a deeper understanding into the behavior of the genetic algorithm over time and specifically how the fitness function F behaves, GA performance graphs of the behavior of individual fitness measures are shown in Figures 5.23–5.28 (pages 108–111). The six GA runs chosen are identical with the exception of the stream of random numbers employed. Out of the 100 GA runs that comprise an experiment, these six were chosen since all of them resulted in the discovery of 3-component EA self-replicating structures of the form `UL3EC13V`. The overall fitness function that was used (for all GA runs) in the experiment was

$$F = 0.05 f_g + 0.75 f_p + 0.20 f_r \tag{5.2}$$

where, as described in Section 4.5.4.1, f_g is the fitness measure for growth, f_p is for relative positions, and f_r is for isolated replicants. The weights chosen in Equation 5.2 were arrived at by experimentation in this case[3]. A reasonable interpretation of Equation 5.2 is that the overall fitness value for a chromosome (i.e., rule table) should mainly come from the relative positions of components. Less important are the isolated replicants and growth of components. However, there is a deeper interpretation. What actually happens, as we shall see, is that the presumably insignificant growth measure plays a key role in getting the GA primed, the relative position maintains a steady increase in F, and the isolated replicant measure serves to lock-in a newly discovered self-replicating structure. These behaviors suggest that the three parts of the fitness function support each other in complex and unanticipated ways.

The GA performance graphs of Figures 5.23–5.28 plot values of F, f_g, f_p, and f_r (Equation 5.2) for the highest ranking chromosome of each generation (also called the "best-of-generation" chromosome). As discussed in Section 4.5.2, elitism is used whereby the best two chromosomes are copied directly from generation g to generation $g+1$. Thus plateaus can be seen on the GA performance graphs indicating that an elite chromosome went "unchallenged" for a certain period of time. Inspecting the six performance graphs for general trends, we make some general observations. The growth measure f_g is generally the most volatile, and this agrees with intuition: since it contributes the least in guiding F, large fluctuations are easily tolerated and have a lessened effect on F. The relative position measure f_p remains the highest contributor in most cases, which is not surprising since it has 75% weight in F, and thus the overall search, to some degree, is spent optimizing f_p. The isolated replicant measure f_r, being the hardest fitness measure to satisfy, generally stays effectively at zero for, in general, hundreds of generations until the other measures discover a rule table that promotes elements of a self-replicating process.

During roughly the first 50 generations, which we call epoch I, the growth measure increases rapidly, albeit sporadically, to help get the GA "boot-strapped." The growth measure is thought to be the easiest way of gaining fitness value since it is only concerned with quantities of components and not positioning. Thus, even at only 5% weight, it contributes to the early stages of the GA. It is hypothesized that such action seeds the population of rule tables in the GA with a large quantity of `DIV` (divide) actions, so that component production is encouraged. Between generations 50 and 500 (epoch II), the growth measure becomes less volatile, and a clear trend is seen in the ordering

[3]Other times the meta-level GA was used to adapt these weights.

of the curves:

$$f_p > f_g > f_r \tag{5.3}$$

Near the end of epoch II, it can be seen that many of the overall fitness values F are at their first plateau. This suggests that a strong chromosome has emerged that has good growth and positioning of components, yet it does not self-replicate. The last epoch (epoch III) occurs after generation 500 when the isolated replicant measure sharply increases indicating isolated replicants have appeared, and the seed structure may have exhibited self-replication.

We also note that our justification for choosing 2000 as the maximum number of generations to run is again supported by these curves. There is typically a sharp increase in the isolated replicant measure f_r upon discovery of a self-replicating structure. As seen from the performance graphs, such increases occur between generations 500 and 1500.

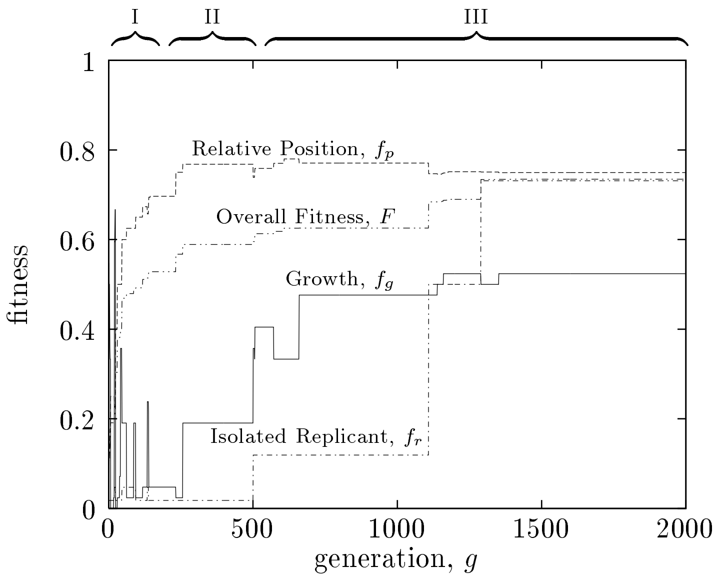


Figure 5.23: Individual fitness measure values for the best-of-generation chromosome during GA discovery of UL3EC13V$_{71}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

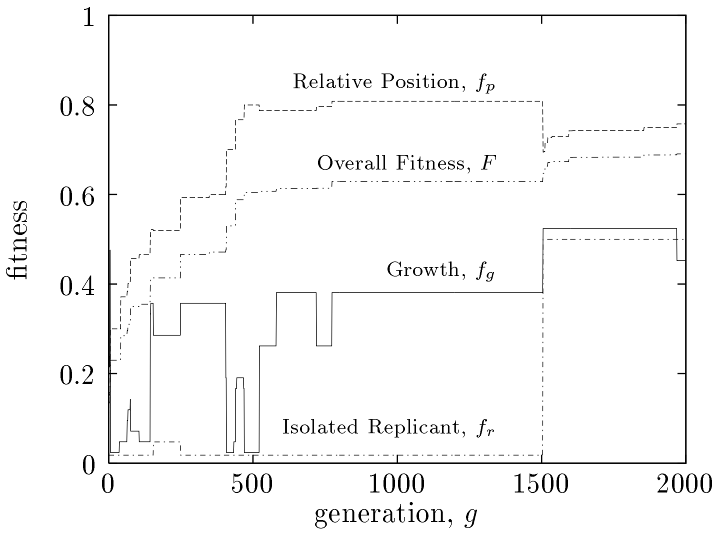


Figure 5.24: Individual fitness measure values for the best-of-generation chromosome during GA discovery of `UL3EC13V`$_{72}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

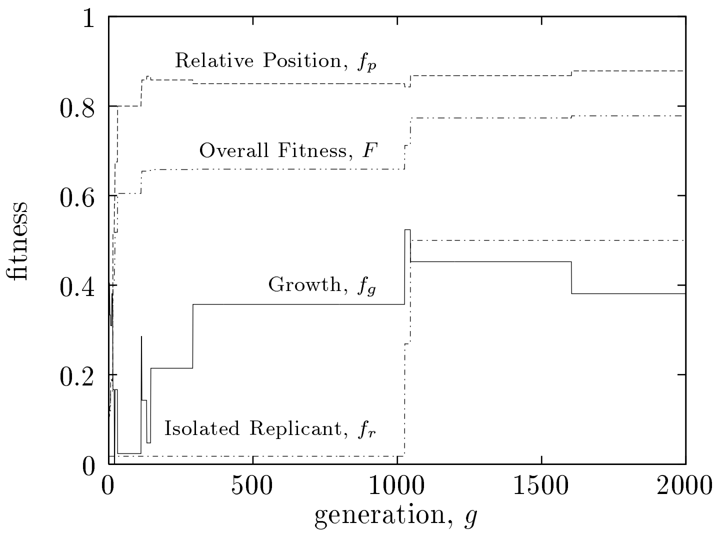


Figure 5.25: Individual fitness measure values for the best-of-generation chromosome during GA discovery of `UL3EC13V`$_{73}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

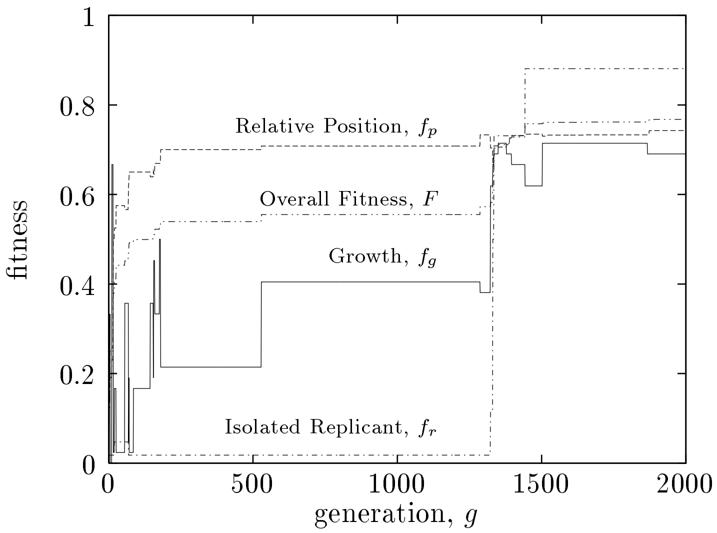


Figure 5.26: Individual fitness measure values for the best-of-generation chromosome during GA discovery of $\texttt{UL3EC13V}_{74}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

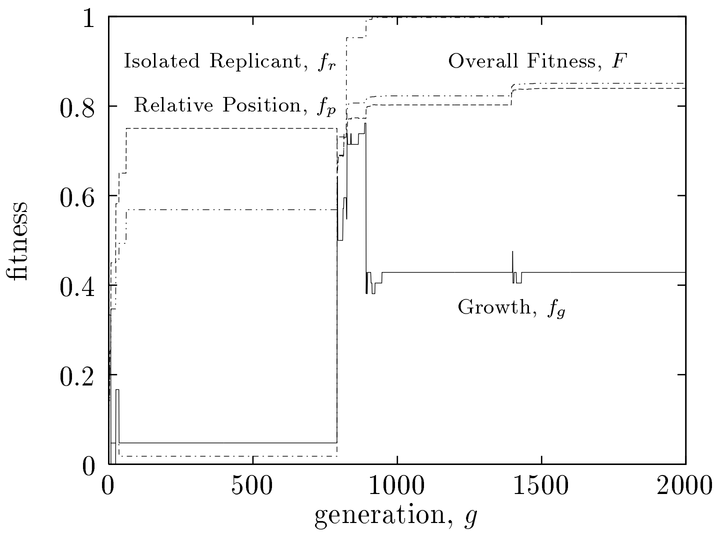


Figure 5.27: Individual fitness measure values for the best-of-generation chromosome during GA discovery of $\texttt{UL3EC13V}_{75}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

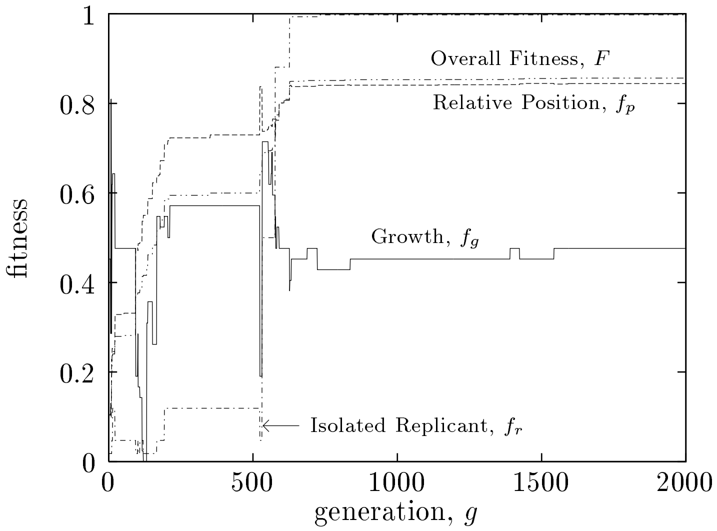


Figure 5.28: Individual fitness measure values for the best-of-generation chromosome during GA discovery of `UL3EC13V`$_{76}$. The overall fitness function is $F = 0.05f_g + 0.75f_p + 0.20f_r$.

Fitness Measure Interaction

A simple experiment was constructed to determine if each of the three fitness measures, by themselves, can promote development of a self-replicating structure. If none of the individual fitness measures are able to produce such a structure, this suggests that the measures are dependent on each other. The three experiments involve setting the weight vector $\mathbf{w} = (w_g, w_p, w_r)$ as follows:

$$\mathbf{w} = (1, 0, 0) \qquad \text{Experiment 1}$$

$$\mathbf{w} = (0, 1, 0) \qquad \text{Experiment 2}$$

$$\mathbf{w} = (0, 0, 1) \qquad \text{Experiment 3}$$

Experiments were conducted in the same manner as described in Section 5.1 using both EA and CA models with 3-component structures. The results were that zero self-replicating structures were discovered, suggesting that fitness measures interact and depend on each other to promote self-replicating behaviors.

5.3 Software System

The experimental method described in Section 5.1 was implemented in a software system designed by the author. The system was developed to be flexible, efficient, robust, and easy to use. Specific emphasis was placed on making the system general enough to allow a wide range of both CA and EA models to be simulated in a standalone manner as well as under a genetic algorithm. The main limitation in running genetic algorithm experiments is imposed by the resources available, since larger models require more memory and CPU time. For a typical present-day workstation having a RISC CPU and 64 megabytes of main memory, a genetic algorithm together with an EA model having $k = 17$ states will require approximately 15 hours to compute 200,000 fitness evaluations.

A block diagram showing the major components of the system is shown in Figure 5.29. The simulation engine is the core component of the system. It processes and stores the rule table input and iterates the cellular space over time. The statistics collection subprogram runs a single simulation using the simulation engine and collects relevant statistics at each time step. The visualization subprogram allows viewing of both CA and EA simulations. A simulation may be viewed in both forward and backward time directions (moving in the backward time direction simply displays previously generated images, and does not mean the cellular space can be iterated in this direction). In addition, graphics files may be saved at each time step and later printed out for hardcopy output (examples of which are the diagrams showing evolving structures in this thesis). The sources of input for the system are labelled "GA parameters" and "rule table and parameters". These input sources configure the system. Note that the output from the "Sequential Genetic Algorithm" subprogram is a rule table that is suitable for statistics collection and visualization.

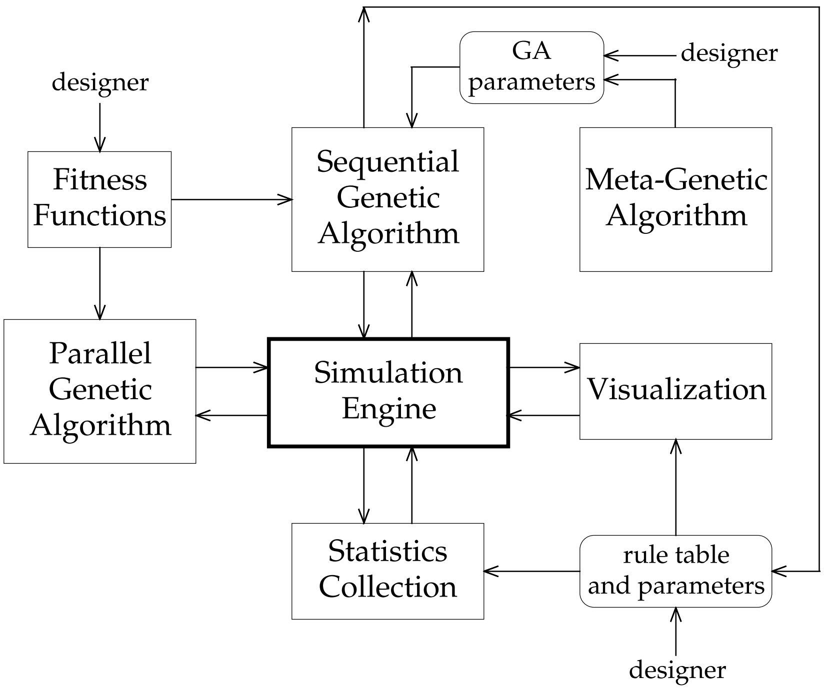


Figure 5.29: System block diagram shows the major components of the system in which experiments were performed. The simulation engine forms the core of the system since it is used by all components either directly or indirectly. Boxes indicate subprograms of the system, and ovals represent parameter sets that configure the system.

The genetic algorithm subprograms provide sequential and parallel implementations, and a meta-level GA for multiobjective optimization. The parallel genetic algorithm is depicted in Figure 5.30. In a technique called *semi-synchronous master slave* [Goldberg89], chromosomes are distributed to individual processing nodes via message-passing, EA/CA simulations are run locally, fitnesses calculated, and then fitnesses are sent back to the host node which maintains the populations. The name semi-synchronous is used since the host will asynchronously send chromosomes to the processing elements (PEs) during a single generation, however, it must wait (synchronize) until all PEs have finished before proceeding to the next generation. This form of parallelism is efficient as long as there is a low variance among simulation execution times, which is the case for these simulations.

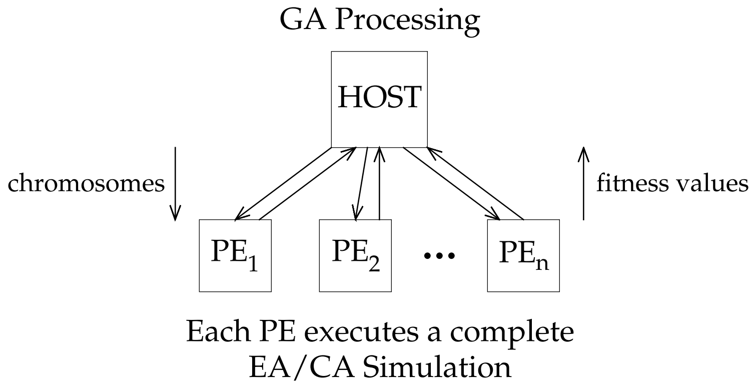


Figure 5.30: Semi-synchronous master/slave GA parallelism. Host processor executes the GA. In parallel, processing elements receive chromosomes, execute an EA or CA simulation, then send fitness values to the host.

Parallelism is also obtained when performing statistical trials, or experiments. As shown in Figure 5.31, each of the processing elements runs an entire genetic algorithm in parallel, and upon completion sends the highest-fitness chromosome to the host for storage. The host also scans for idle PEs and launches GAs as appropriate until the experiment is complete. The host operates asynchronously since it has no dependencies to wait for, and thus it does not need to synchronize at any point during the experiment.

The third form of parallel processing implemented in the software system is the system for executing the meta-level GA. As shown in Figure 5.32, the parallelism is similar to that of Figure 5.31. In this case, however, the host processor is running a separate (smaller and less computationally intensive) GA to optimize fitness measure weights of Equation 4.16. The PEs each execute a complete primary (i.e., rule discovery) GA and send fitness values to the host.

The system was implemented in the C++ programming language and is comprised of over 10,000 lines of source code. It was developed on Sun workstations and has successfully run on other computer systems including DEC Alpha, RS/6000, and PCs running UNIX. The parallel versions supported run on the following supercomputers: DEC Alpha processor farm clusters, Thinking Machines Connection Machine 5, and the IBM SP2. A set of UNIX shell scripts is also part of the system, and it allows for load balancing on processor farm clusters.

The system has been released into the public domain so that other researchers may use this system as a research tool. This and other details concerning the simulation system may be found in Appendix B.

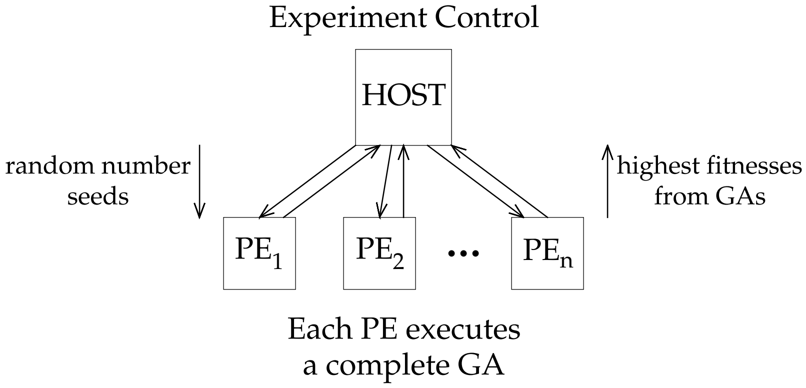


Figure 5.31: Asynchronous master/slave parallelism for running an experiment. Host processor oversees experiment by asynchronously starting GAs when idle PEs are seen. Each processing element executes, in parallel, a complete GA, then sends the highest-fitness chromosome found to the host for storage.

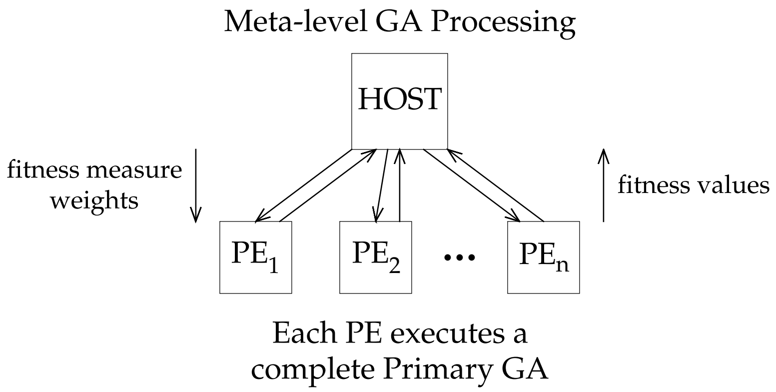


Figure 5.32: Parallelism in meta-level GA. Host processor executes meta-level GA and distributes fitness measure weights to PEs. Each PE executes, in parallel, a complete GA, then sends the overall fitness to the host.

Chapter 6

Conclusions and Future Work

The research presented in this dissertation focuses on the automatic design and analysis of self-replicating structures in cellular space automata models. In conclusion, we summarize the main contributions of this work and discuss open problems and areas where further study would be beneficial.

6.1 Conclusions

The research results in this dissertation contain several important contributions towards the theory of self-replicating automata that began with the work of John von Neumann. Automatic creation of self-replicating structures in cellular space models was shown to be possible and methods for improving the efficiency of the discovery process were presented. These include the use of component-sensitive input and use of the effector automata model introduced in Chapter 3. Central to the rule discovery process was the design of effective fitness measures to promote self-replicating behaviors. Results presented showed that these fitness measures:

- did not impose a strong bias towards a particular process of self-replication as evidenced by the large variety of structures found,
- are not specific to any one cellular space model,
- are computationally feasible,
- and resulted in a statistically significant quantity of discovered self-replicating structures.

Building on the success of automatically discovering self-replicating structures, an analysis of a large collection of such structures was undertaken, a task that was never before possible due to the lack of specimens to study. Representative samples of these structures were presented and analyzed both quantitatively and qualitatively.

The self-replicating structures presented in this dissertation compare favorably in terms of simplicity with those generated manually in the past [Reggia93]. However, more interesting is that these replicating structures differed in unexpected ways from those developed in previous automata models. For example, they all were moving during replication, and all generated debris (unused extra components). In some simulations, the replicant was not the initial seed structure but a larger structure built from it. Such unanticipated results suggest that genetic algorithms can be powerful tools for exploring the space of possible self-replicating structures. Furthermore, if the basic physical processes can be identified and represented effectively, such an approach might even be modified and applied to discover new self-replicating molecular structures [Hong92].

6.2 Future Work

The paradigm of component-sensitive input introduced in this thesis is a general technique that could be further exploited in other cellular space models, especially those models that have proven too computationally burdensome to simulate in the past. The effector automata model provides many of the same advantages of the standard cellular automata model, yet with more physical realism, smaller rule tables and search spaces, and more complex automata. As in CA applications, a vast range of potential behaviors are possible with EA models. Future work on automatic discovery of self-replicating structures that would be useful includes:

Investigation of Minimal Structure Size An open problem related to this research concerns the question of minimal self-replicating structure size, in terms of the rule table size. The results of this dissertation provide intuition into solving such a problem since two and three component structures were used which had small numbers of rules used in the replication process.

Use of the Moore Neighborhood The studies in this thesis concentrated on automata using the von Neumann neighborhood. Using the larger Moore neighborhood (Figure 2.1) would presumably allow more complex self-replicating structures to develop due to greater interaction among components.

Investigation of Other Automata Models In addition to the cellular automata and effector automata models studied in this thesis, other cellular space models such as stochastic automata and inhomogeneous cellular automata [Hartman86] can be used with the rule discovery techniques presented herein. Also, properties of the models researched could be varied in the following ways: hexagonal space tessellation, other cell contention policies and varying sets of actions (for EA models).

Increased Complexity of Seed Structures The seed structures used were limited to a maximum of four components due to computational limitations. As more powerful computers become available, it would be of interest to discover rule tables for larger structures, some having unique components, and others having repeated components. One would hypothesize that a GA would more easily discover rule tables for seed structures having unique components because in the case of repeated components, a specific repeated component has to be trained to facilitate self-replication in many more situations. Also of interest are cases in which not all of the component types are represented in the seed structure.

Self-organization of Seed Structure Another avenue of pursuit is to begin with a random configuration of components distributed throughout the cellular space. Through the use of effective fitness functions, it might be possible to encourage the self-organization of seed structures which have the ability to self-replicate. Such an experiment could be constructed so that if self-replication does not emerge within a given timeframe, the space becomes completely quiescent.

Co-evolution of the Seed Structure Co-evolving the seed structure and the rule table simultaneously might yield interesting results. Although such an approach was briefly tried, with more powerful computers such an approach would be worthwhile.

Refinement of Fitness Measures Assigning partial fitness to nascent self-replicating structures is a non-trivial problem, and it would be difficult, if not impossible to define an optimal fitness function. However, further refinement of the fitness measures could result in techniques that yield even larger quantities of self-replicating structures.

Biochemical Simulation The EA model and research software used in this thesis could be adapted to simulate, at a low level, basic biochemical nucleotide interactions. The goal of such an experiment could be to see what underlying cellular space rules are needed to promote template-directed replication. For example, EA components named `A`, `C`, `G`, and `T` could be used to represent the four nucleotide bases of DNA.

Appendix A

Calculation of Circular Permutations

Certain calculations of search space sizes presented in the main text rely on advanced combinatorics. Circular permutations are needed in calculations involving isotropic neighborhood patterns. Here we present below the main points from this theory. For a full treatment, see a text on combinatorial theory such as [Hall67, pg. 12].

In order to derive an expression for the number of circular permutations, the *Möbius function* from number theory, denoted $\mu(n)$, is defined as follows. First, we note that every positive integer $n > 1$ has a unique factorization as a product of prime powers

$$n = p_1^{i_1} p_2^{i_2} \cdots p_r^{i_r} \tag{A.1}$$

where each p is a unique prime and each i is a positive integer. $\mu(n)$ is then defined as:

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if any } i_k > 1 \text{ in Eq. A.1} \\ (-1)^r & \text{if } i_1 = i_2 = \cdots = i_r = 1 \text{ in Eq. A.1} \end{cases} \tag{A.2}$$

The number of circular permutations of length n, using k distinct symbols is denoted ${}_k\mathrm{CP}_n$, and is calculated by summing the function $M(n)$ over $d|n$ (the notation $d|n$ (d,n positive integers) represents each integer in $\{1, \ldots, d\}$ that evenly divides n for $d \leq n$). $M(n)$ is the number of circular permutations of period n, and is expressed as

$$M(n) = \frac{1}{n} \sum_{d|n} \mu(d) k^{\frac{n}{d}} \tag{A.3}$$

Thus the total number of distinct circular permutations of length n is

$${}_k\mathrm{CP}_n = \sum_{d|n} M(d) \tag{A.4}$$

As an example, consider UL06W8V, a 2-D cellular automata model having 4 strongly symmetric cell states $\{\bullet, \#, \mathrm{L}, \mathrm{O}\}$, and 1 weakly symmetric symbol {A} which can be rotated in any of 4 directions {A, ⊳, ∀, ⊲}. Using the von Neumann neighborhood, we have $n = 4$, and $k = 8$ cell states. The calculation of circular permutations begins by calculating values for M in Equation A.3

as follows:

$$
\begin{aligned}
M(n) &= \tfrac{1}{n}\textstyle\sum_{d|n}\mu(d)k^{\frac{n}{d}} \\
\\
M(1) &= \tfrac{1}{1}[(1)8^{\frac{1}{1}}] \\
&= 8 \\
\\
M(2) &= \tfrac{1}{2}[(1)8^{\frac{2}{1}} + (-1)8^{\frac{2}{2}}] \\
&= 28 \\
\\
M(4) &= \tfrac{1}{4}[(1)8^{\frac{4}{1}} + (-1)8^{\frac{4}{2}}] \\
&= 1008
\end{aligned}
$$

and then substituting into Equation A.4:

$$
\begin{aligned}
{}_8\mathrm{CP}_4 &= M(1) + M(2) + M(4) \\
&= 1008 + 28 + 8 \\
&= 1044
\end{aligned}
$$

Thus, there are 1044 state transitions for each of the four strongly symmetric components, giving a total of 4176. For the weakly symmetric component, there are $8^4 = 4096$ transition rules. Summing these we get 8272 total transition rules.

For reference purposes, Table A.1 compiles values of ${}_k\mathrm{CP}_4$ for small k. The function k^4 represents the number of length four permutations of k unlike objects, when each may be repeated any number of times, and is tabulated for use when comparing isotropic to non-isotropic cellular space models.

k	k^4	${}_k\mathrm{CP}_4$	k	k^4	${}_k\mathrm{CP}_4$
1	1	1	11	14641	3696
2	16	6	12	20736	5226
3	81	24	13	28561	7189
4	256	70	14	38416	9660
5	625	165	15	50625	12720
6	1296	336	16	65536	16456
7	2401	616	17	83521	20961
8	4096	1044	18	104976	26334
9	6561	1665	19	130321	32680
10	10000	2530	20	160000	40110

Table A.1: Tabulation of permutation values for n=d=4.

Appendix B

Software System Details

B.1 Introduction

The software environment consists of 12 programs which are divided into two groups of six: one for simulating the EA model, and the other for the CA model. For each model there is an simulation engine, an X-window viewer, and a genetic algorithm. Additionally, there are two versions of each program: one in which the input paradigm is component-sensitive input (CSI) and the other is for state-sensitive input (SSI). The names of all the programs and their associated properties are summarized in Table B.1 below.

	CA		EA	
	SSI	CSI	SSI	CSI
Simulation engine	`ca1`	`ca2`	`ea1`	`ea2`
X-Windows viewer	`xca1`	`xca2`	`xea1`	`xea2`
Genetic algorithm	`caga1`	`caga2`	`eaga1`	`eaga2`

Table B.1: Software system program names.

The software has been tested and successfully runs on five computing platforms: Sun SPARC workstations running SunOS/Solaris, DEC Alpha workstations running Digital UNIX, IBM RS/6000 workstations running AIX, and IBM PC compatibles running Linux and BSD/OS UNIX. All of these computers need to have C and C++ compilers, lex, and X-Windows installed. In addition parallel versions of the genetic algorithm will run on Thinking Machines CM5, DEC Alpha farm clusters, and IBM SP2 systems.

B.2 Installing the System

Installation of the software involves two steps: unpackaging the software and building the programs. The distribution file will be called `ea.tar.gz` or `ea.tgz`. To unpackage this file, enter the following commands:

```
gunzip ea.tar.gz  or  gunzip ea.tgz
tar -xvf ea.tar
```

Building the programs is accomplished by using the `make` utility. For example, the create the `xea2` program, one enters the command `make xea2`. The `make` system is configured by default for Sun workstations. To install the programs on non-Sun workstations, users should read the `README` and `Makefile` files for assistance.

Once installed, each program is configured for a particular run by using *config* files. Config files all have names that end in "`.cfg`". A number of demonstration config files are included, and are useful for testing that the system is working properly. For example, one can enter the command,

```
xea2 -c srs3a.cfg
```

which will allow the user to view an EA simulation based on the configuration information in the `srs3a.cfg` file. The "`-c`" option specifies that a config file should be loaded.

B.3 Using the Viewer

The viewer programs allow the user to observe cellular and effector automata models over time. The user is presented with a square space, 80 cells long on each side. Figure B.1 shows the viewer with the major areas outlined. After loading the config file, the user may begin by advancing the simulation one time step at a time by pressing the forward button `->`. A fast-forward button `-->` permits viewing of a rapid succession of configurations, and a stop button `Stop` halts this process. The reverse button `<-` allows the user to iterate backwards in time (previously displayed images are shown – the cellular does not actually iterate in reverse). Figure B.2 shows the simulation controls as they appear at the top of the window.

In the center of the viewer a square, called a frame, is displayed for two purposes: to provide a frame a reference (with respect to moving automata); and for selecting the area of the screen to save to a file. The following list demonstrates the capabilities of the viewer program.

- To change the number of cells in the frame, click on any button in the top row of the Preferences dialog box (Figure B.3).
- To toggle the display of the frame, click on the `View` menu and select "Toggle frame."
- To toggle the display of the grid, click on the `View` menu and select "Toggle grid."
- To generate PostScript output of a single frame, click on the `File` menu and select "Save 1 frame as PostScript."
- To generate PostScript output of a series of frames, click on the `File` menu and select "Save n frames as PostScript."
- To change the PostScript font size click on any button in the bottom row of the Preferences dialog box (Figure B.3).
- To generate a bitmap image of a single frame, click on the `File` menu and select "Save 1 frame as bitmap."

Shortcuts

For convenience, single keystroke commands can be entered to accomplish certain functions. These are summarized in Table B.3. A window containing similar information is available by choosing `Shortcuts...` from the `File` menu.

Key	Command
n	load next config file
r	reload current config file
space	forward 1 time-step
b	backward 1 time-step
q	quit program

Table B.2: Commands using one keystroke.

B.4 Using the Simulation Engine

The simulation programs give the user a convenient way to quickly collect statistics regarding CA or EA rulesets. The statistics are written to an output file which the user can examine. The filename given to the output file is the same as the config file, except "stats" is prepended to the name.

After setting up the appropriate config file, the simulation engine can be invoked, for example, as follows:

```
ea2 -n 15 -c ea2.cfg
```

where `-n` denotes the number of time steps to simulate, and `-c` denotes the config file. The statistics file is a text file containing data regarding the run. This data are arranged as tables where rows correspond to time–steps. Also in this file are the computed values for all the available fitness functions.

B.5 Using the Genetic Algorithm

The genetic algorithm programs allow the user to experiment with having a GA search for high-performing CA or EA rule tables. GA-related parameters are stored in a config file which is read by the program. An example config file is shown in Figure B.6. The parameters shown in Figure B.6 are, for the most part, self-explanatory. The format of the file must adhere to what is seen in this example, with the exception that multiple space characters are equivalent to one. There must not be any blank lines in this file, and the ordering of lines must remain as shown. The fitness function section selects which functions should be used by the GA. There are currently about twenty such functions and users may write their own (in C/C++) which can be added to the source code file `fitness.cc`. Adding new fitness functions requires knowledge of the source code, and more detailed instructions regarding this can be found in the documentation that accompanies the software.

As an example of running the GA, one could issue the command:

```
eaga2 -c eaga2.cfg
```

which will run a GA on the CSI version of the EA model. During a GA run, the GA will save the best-of-generation (also called "best-yet") rule table in a filename that begins with `by` and which includes a time-stamp. This file may then be used as input to run a standalone simulation.

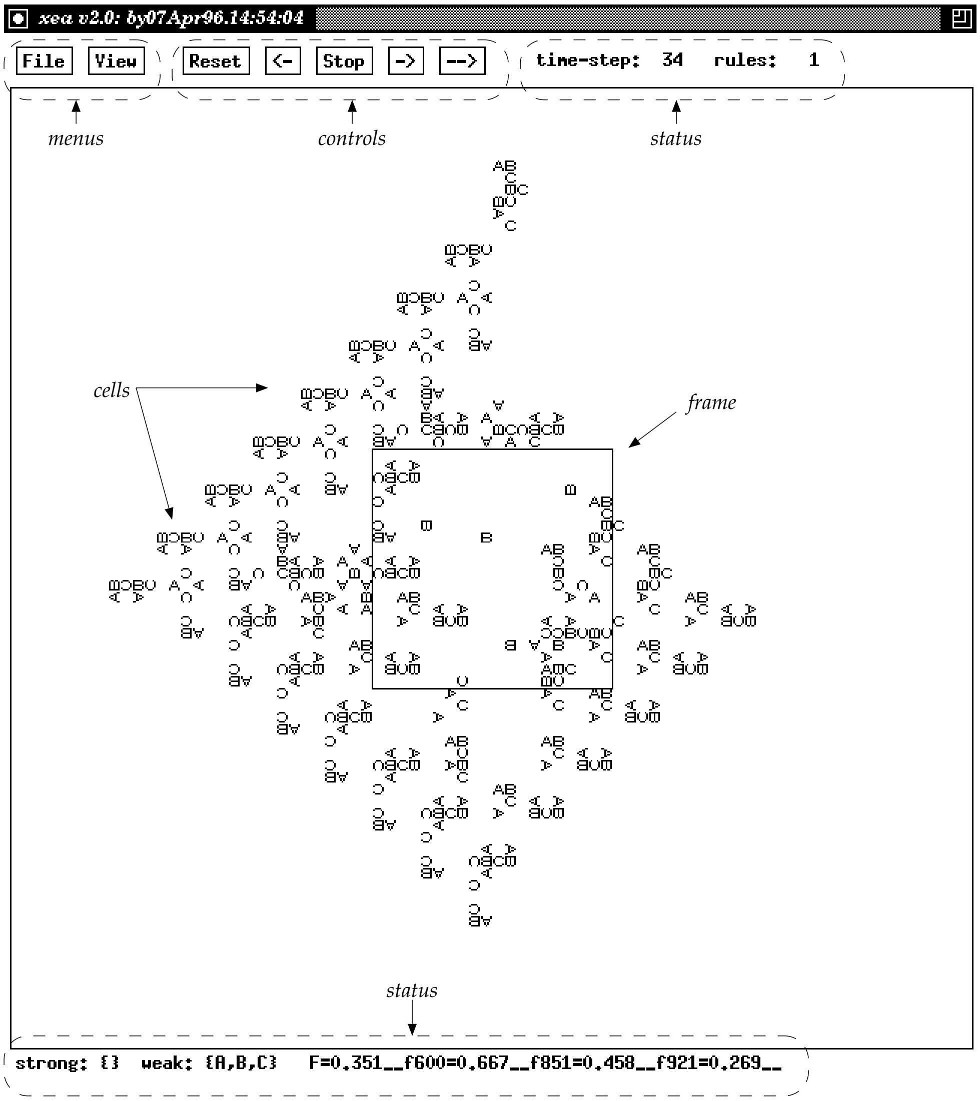


Figure B.1: `xea` viewing program showing main areas. Dashed ovals indicate menus, control buttons, and status information areas. Italicized text is also overlayed to show the frame and cells.

Figure B.2: Simulation controls located at top of main window.

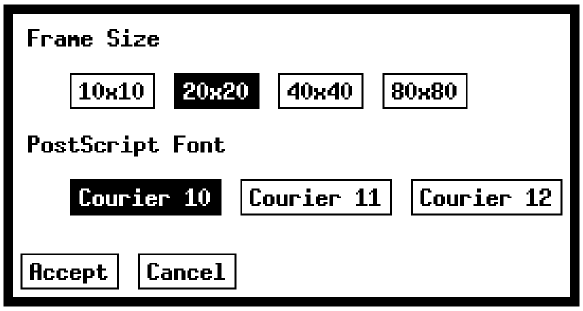


Figure B.3: Preferences dialog box.

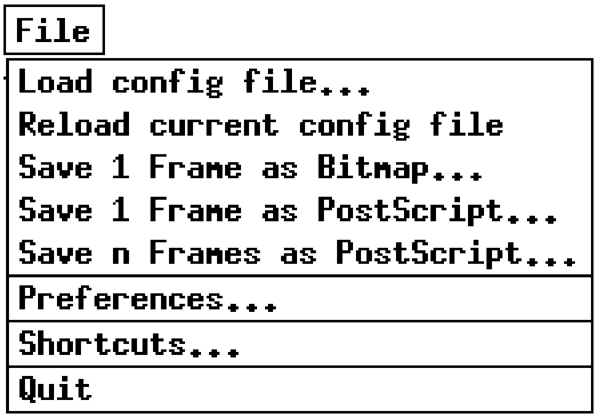


Figure B.4: The `File` pull-down menu.

Figure B.5: The `View` pull-down menu.

```
population_size                       = 100
max_number_of_generations             = 2000
stop_if_constant_for_X_generations    = 2000
crossover_probability                 = 0.8
mutation_probability                  = 0.1
seed                                  = 6030
number_of_simulator_iterations        = 10
stats_collection_begin                = 1
stats_collection_end                  = 10
stop_if_fitness_above                 = 0.99
neighbor_orientation = insensitive
fitness_functions = 3
  ff = 600 wt = 0.3
  ff = 853 wt = 0.6
  ff = 910 wt = 0.1
automata =  3 types:
  A  is directed
  B  is directed
  C  is directed
IC =  3 components:
  A (40, 40) at  0
  B (40, 41) at  0
  C (41, 41) at  0
```

Figure B.6: Example config file for use with genetic algorithm.

Appendix C

Discovered Self-replicating Structures

A small archive of the discovered self-replicating structures appears on the following pages.

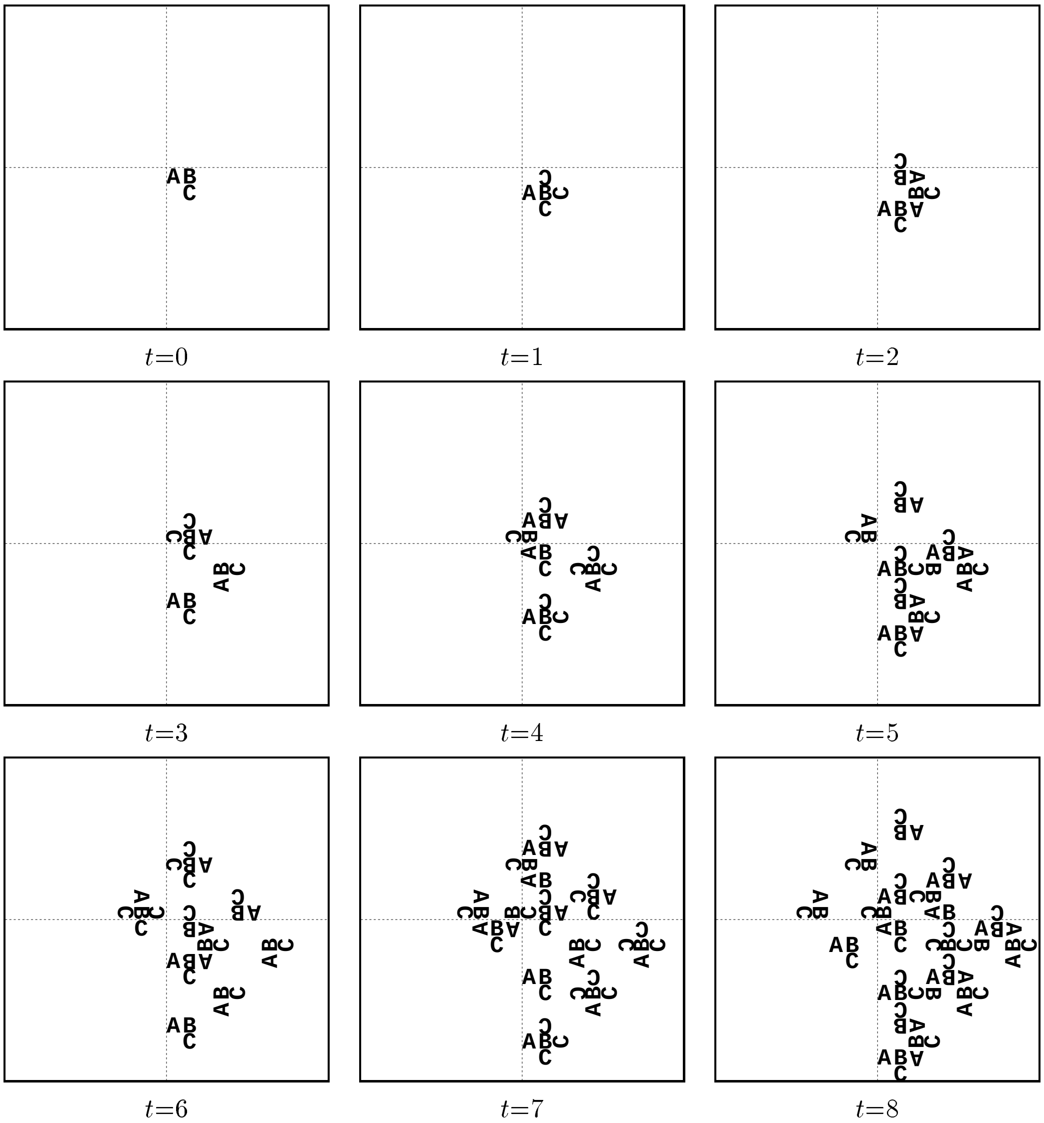


Figure C.1: Self-replicating structure $\texttt{UL3WC13V}_8$.

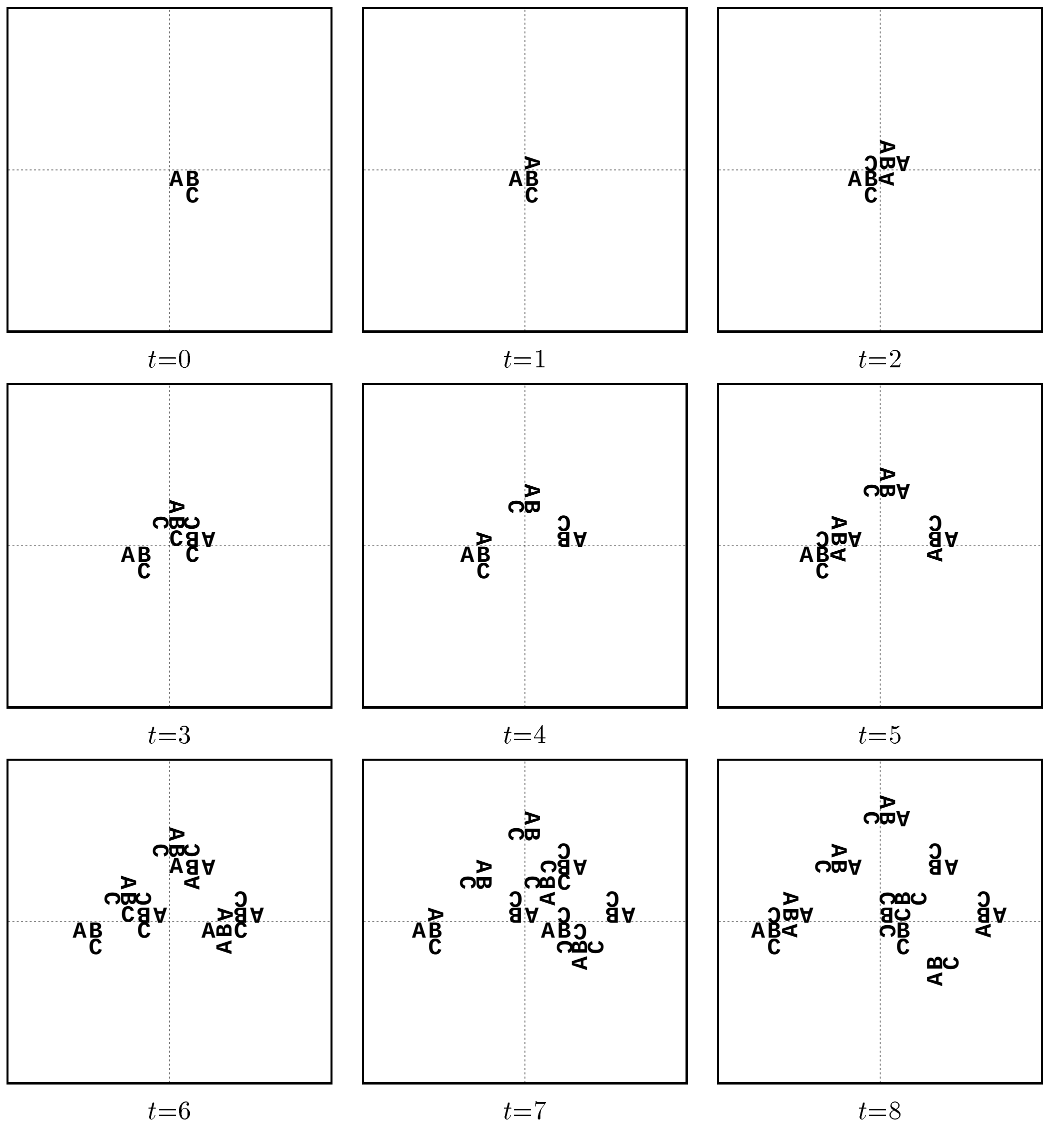


Figure C.2: Self-replicating structure $\mathtt{UL3WC13V}_{13}$.

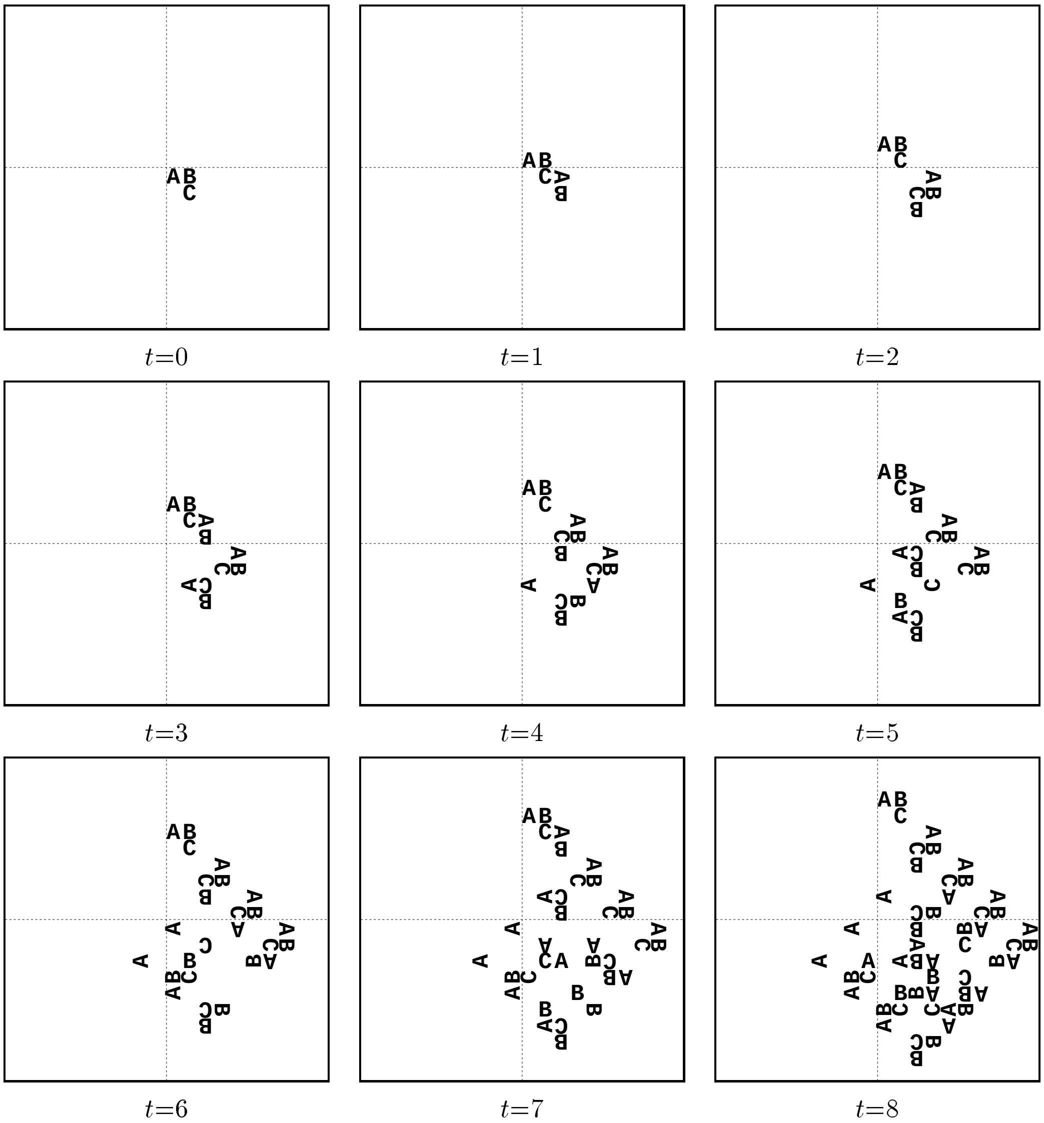


Figure C.3: Self-replicating structure $\mathtt{UL3W13V}_6$.

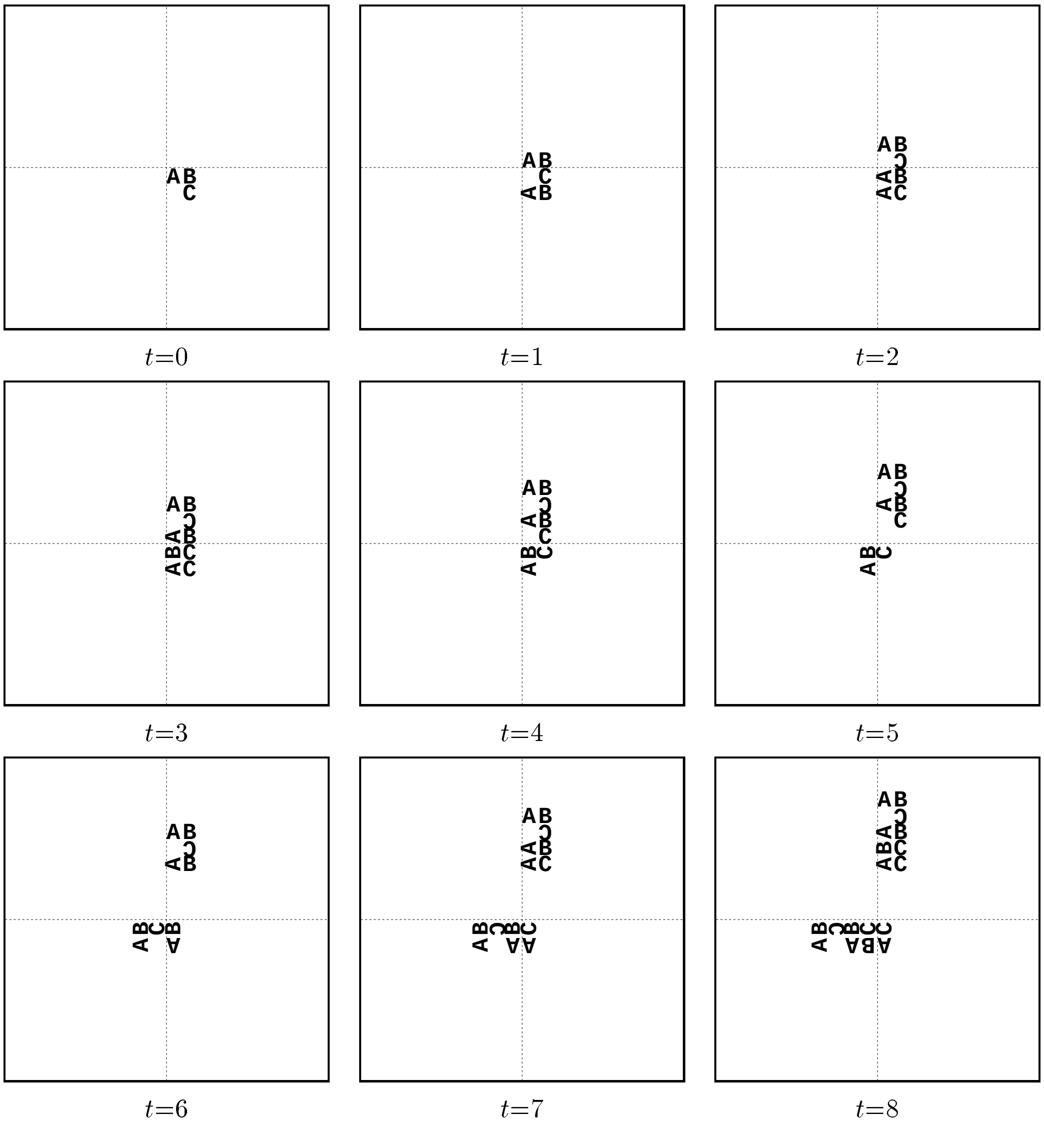


Figure C.4: Self-replicating structure $\mathtt{UL3EC13V}_6$.

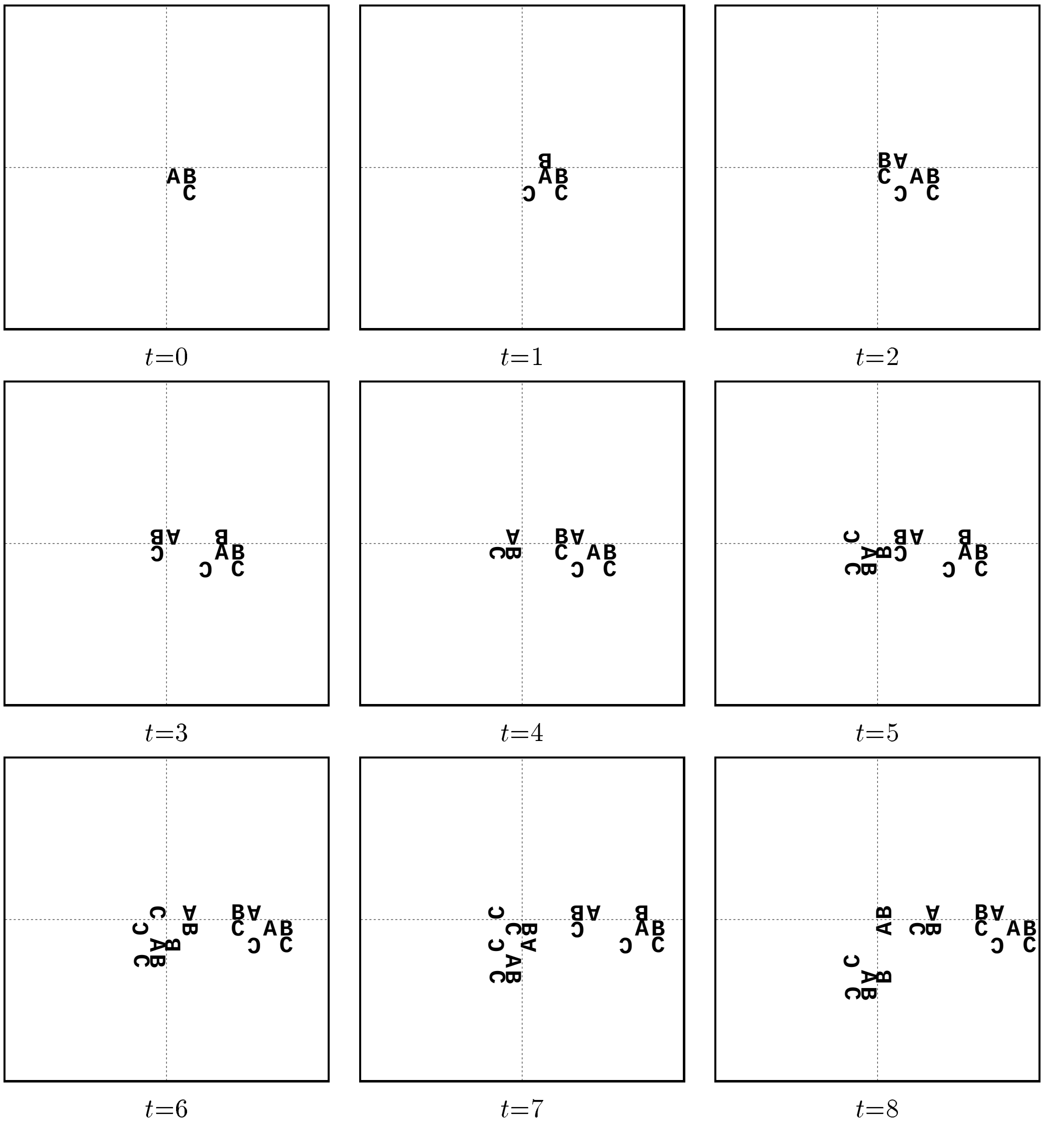


Figure C.5: Self-replicating structure $\texttt{UL3EC13V}_{10}$.

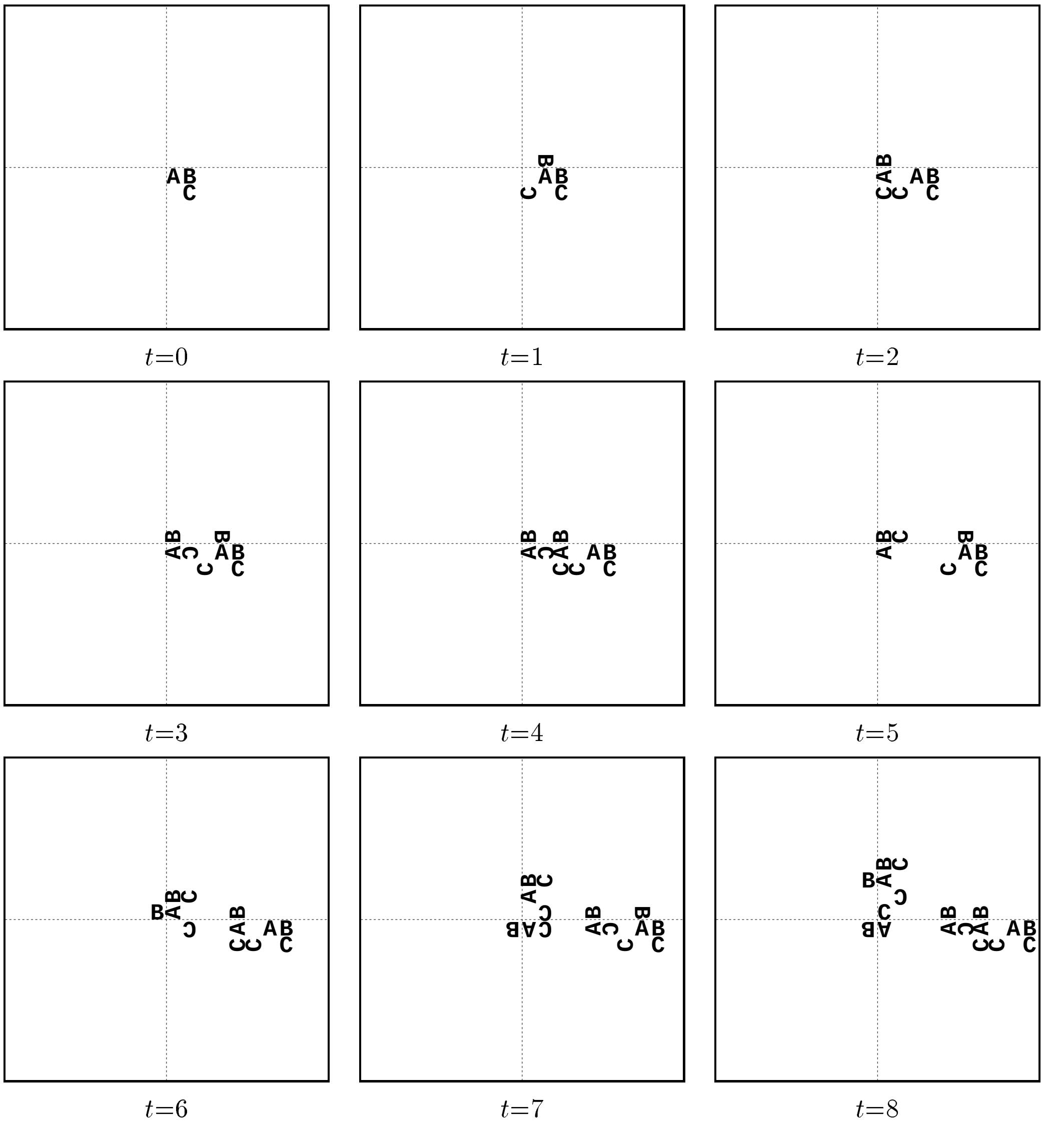


Figure C.6: Self-replicating structure $\texttt{UL3EC13V}_{12}$.

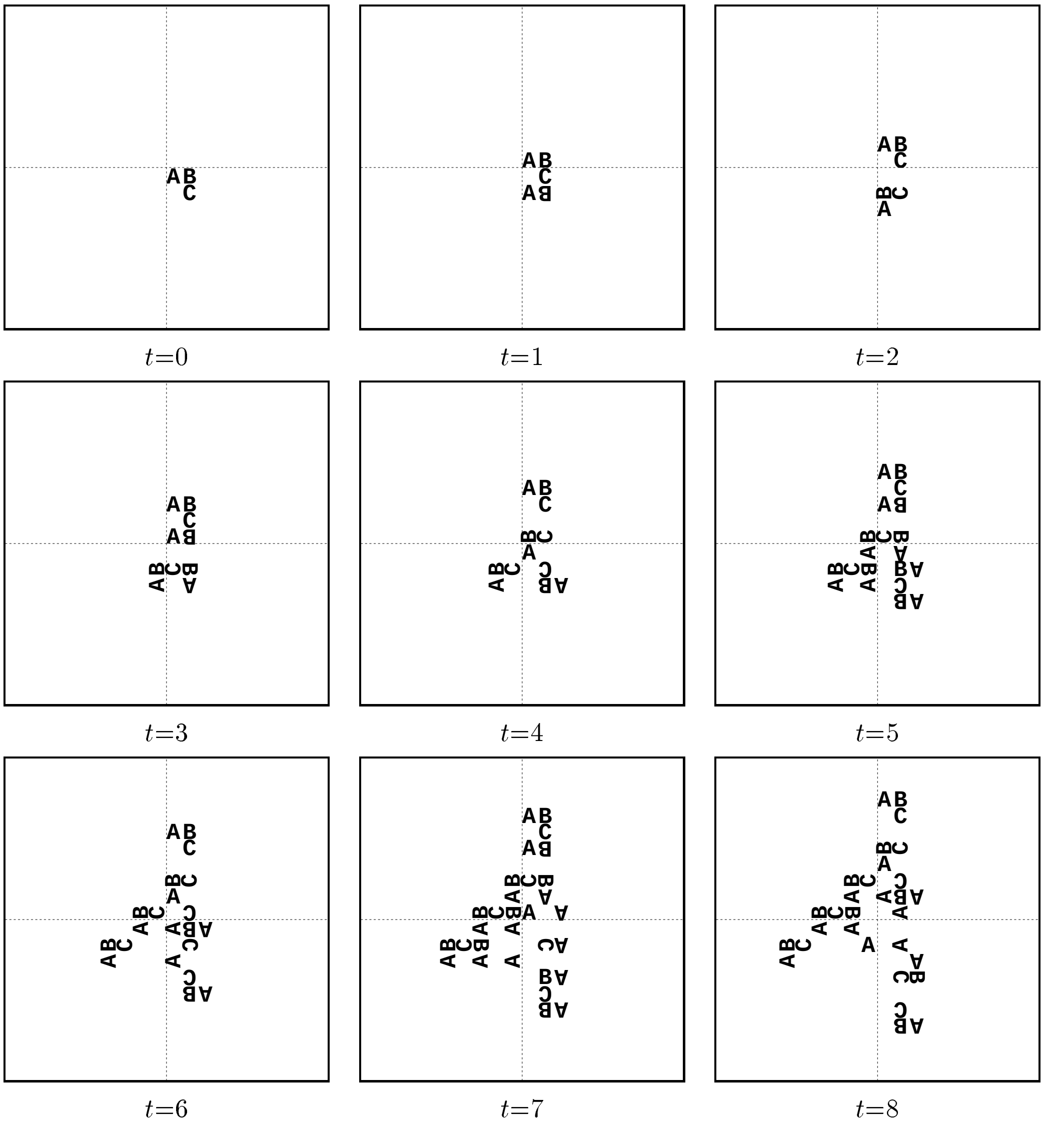


Figure C.7: Self-replicating structure $\texttt{UL3EC13V}_{14}$.

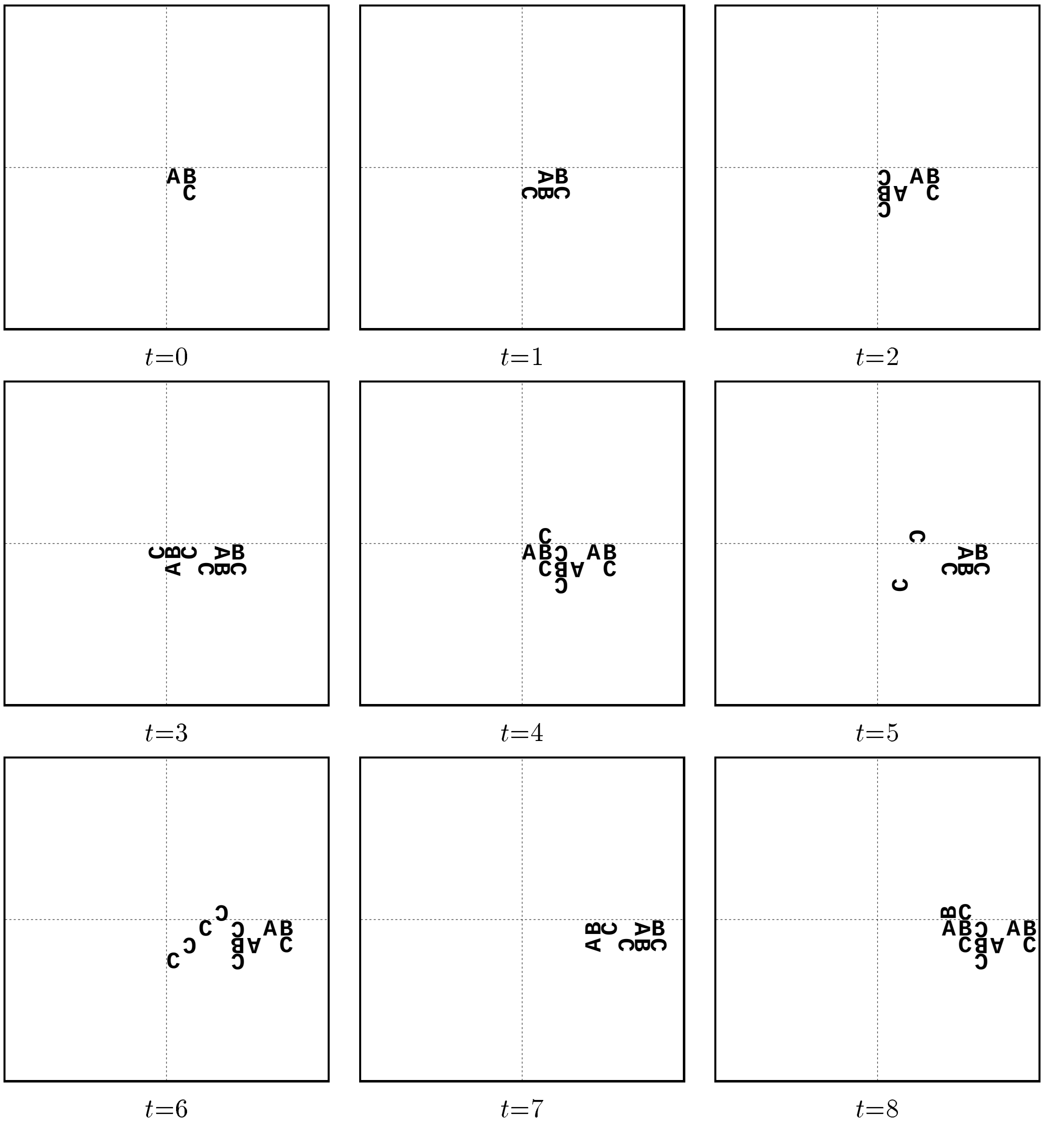


Figure C.8: Self-replicating structure $\texttt{UL3EC13V}_{16}$.

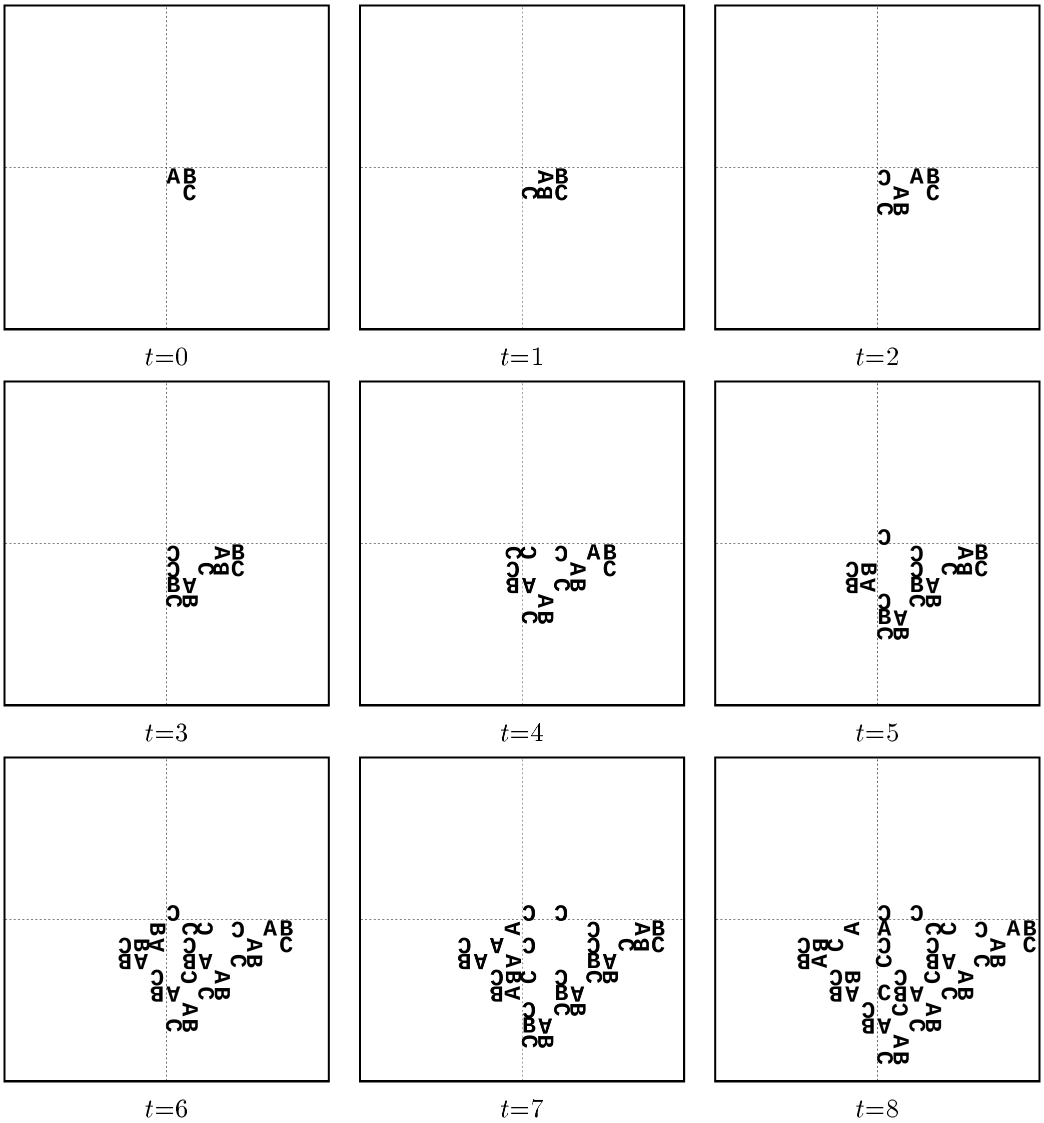


Figure C.9: Self-replicating structure `UL3EC13V`$_{20}$.

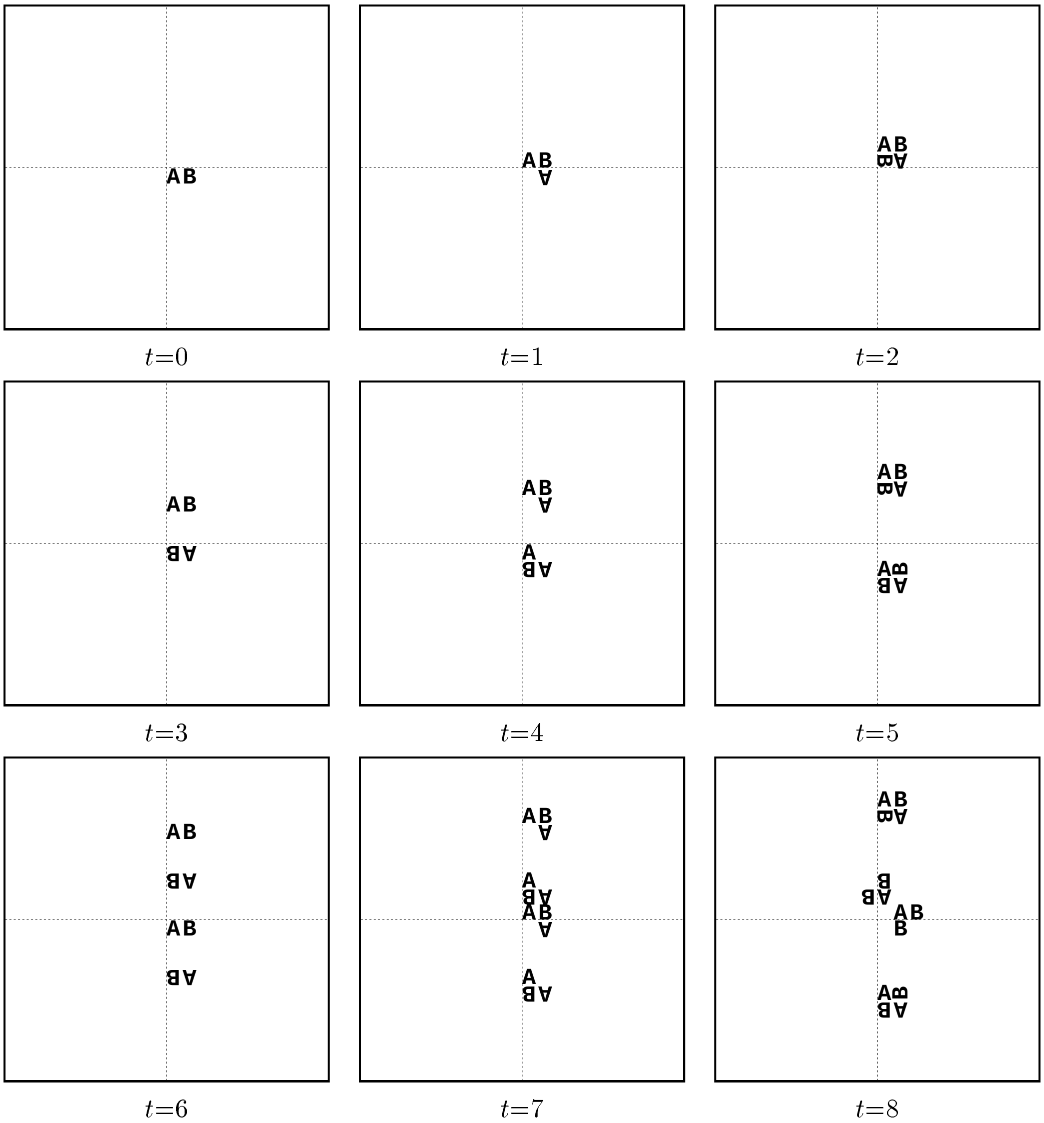


Figure C.10: Self-replicating structure $\mathtt{UL2EC9V}_4$.

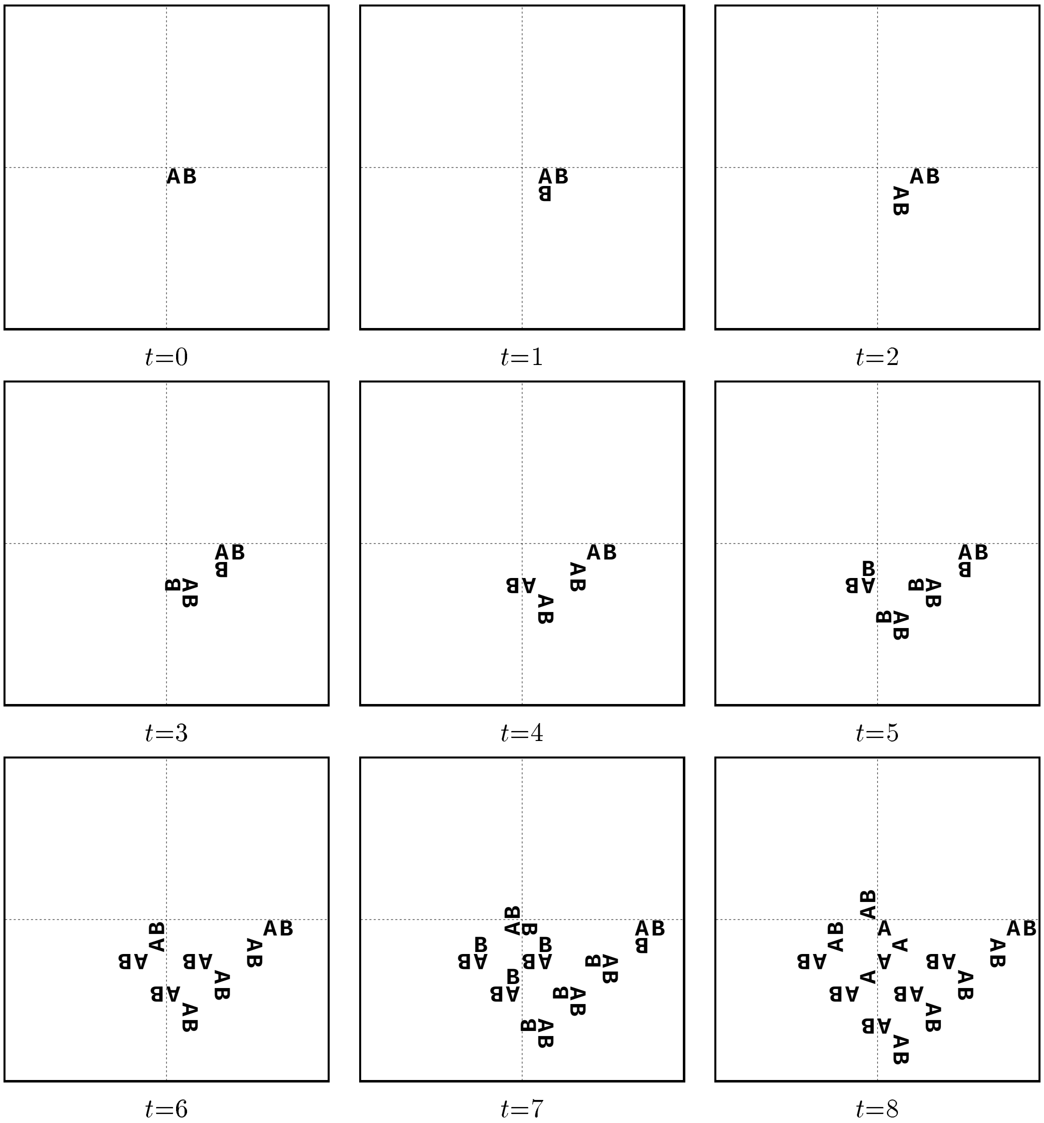


Figure C.11: Self-replicating structure $\texttt{UL2EC9V}_6$.

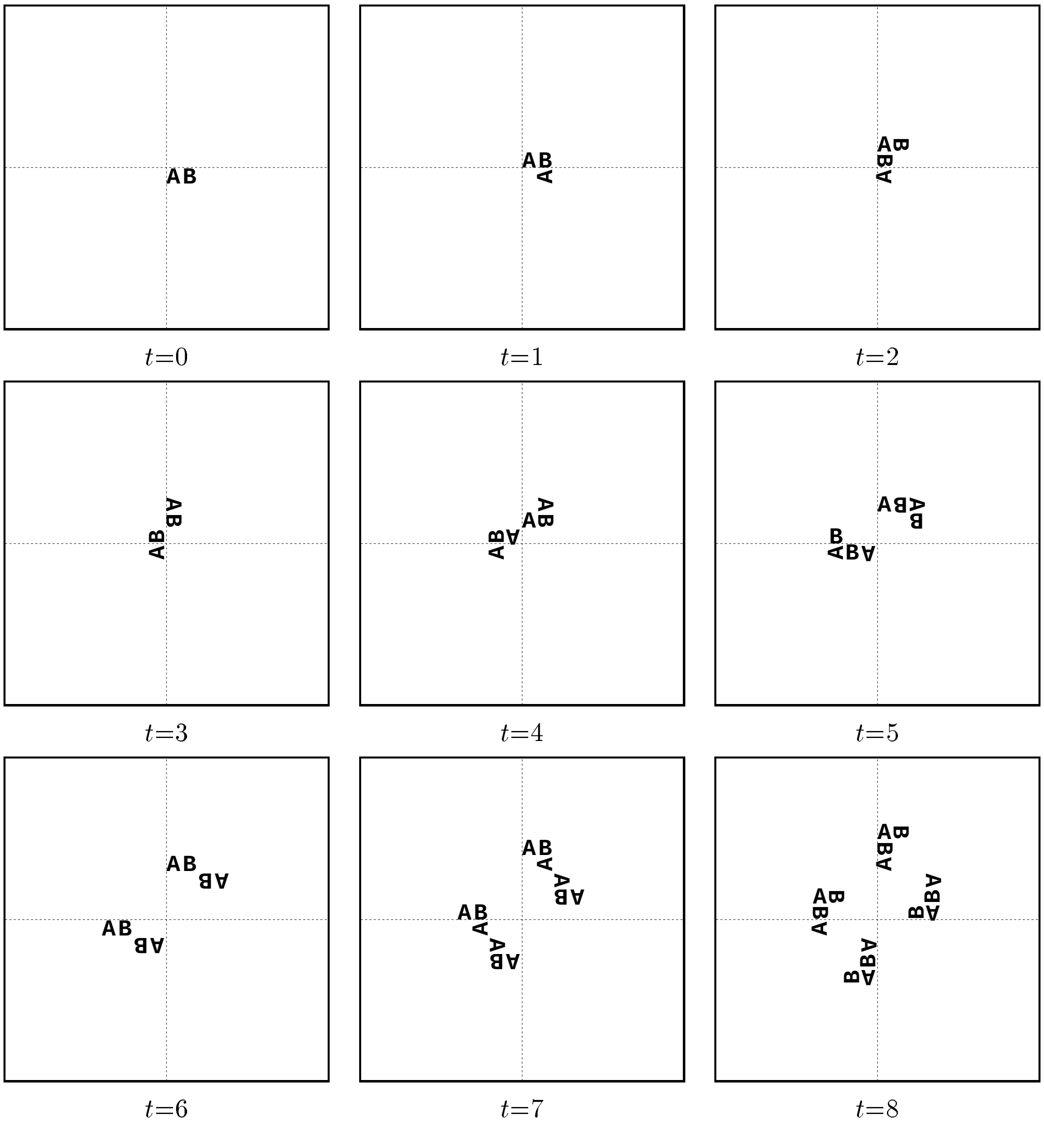


Figure C.12: Self-replicating structure $\texttt{UL2EC9V}_9$.

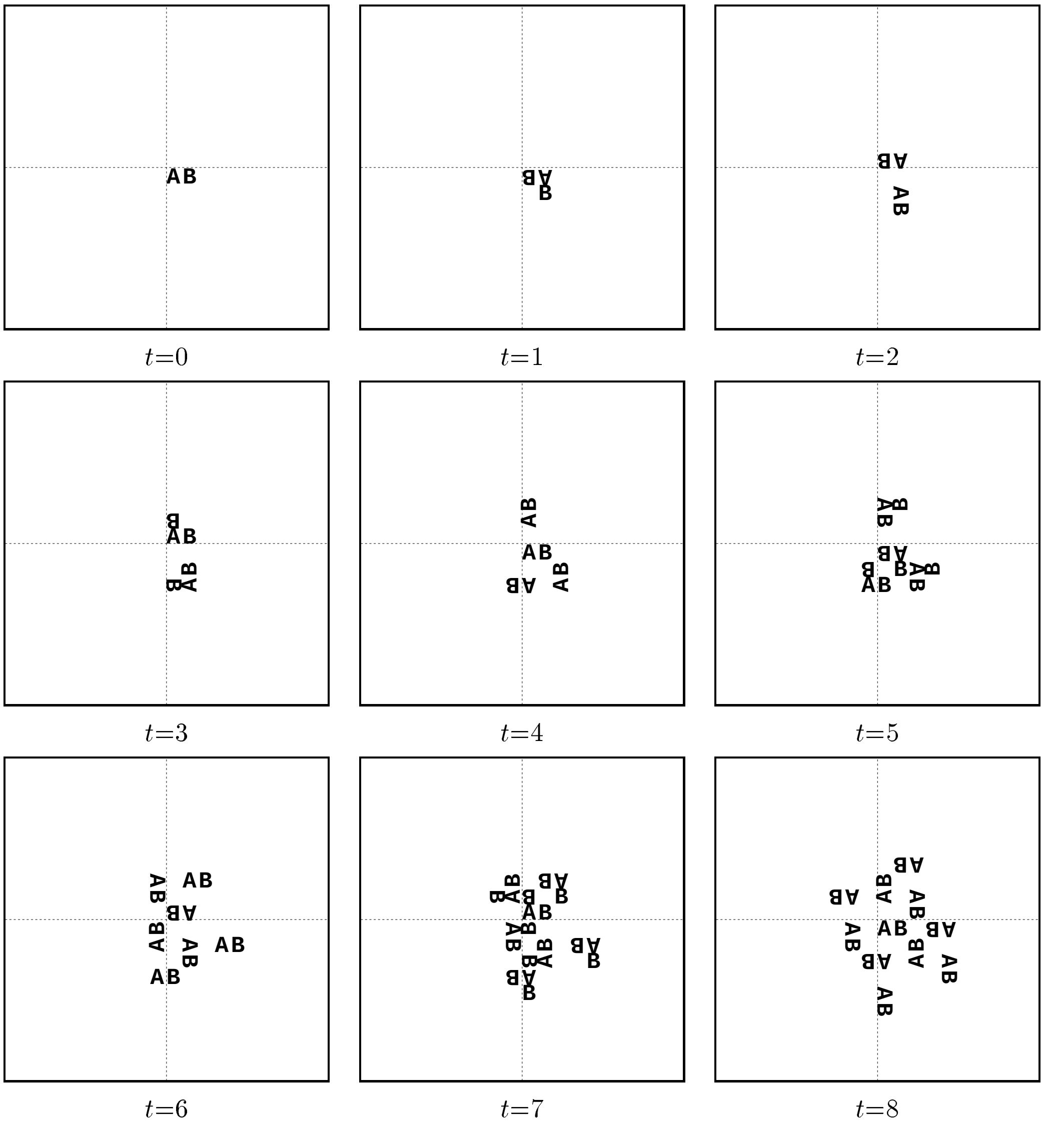


Figure C.13: Self-replicating structure $\mathtt{UL2EC9V}_{10}$.

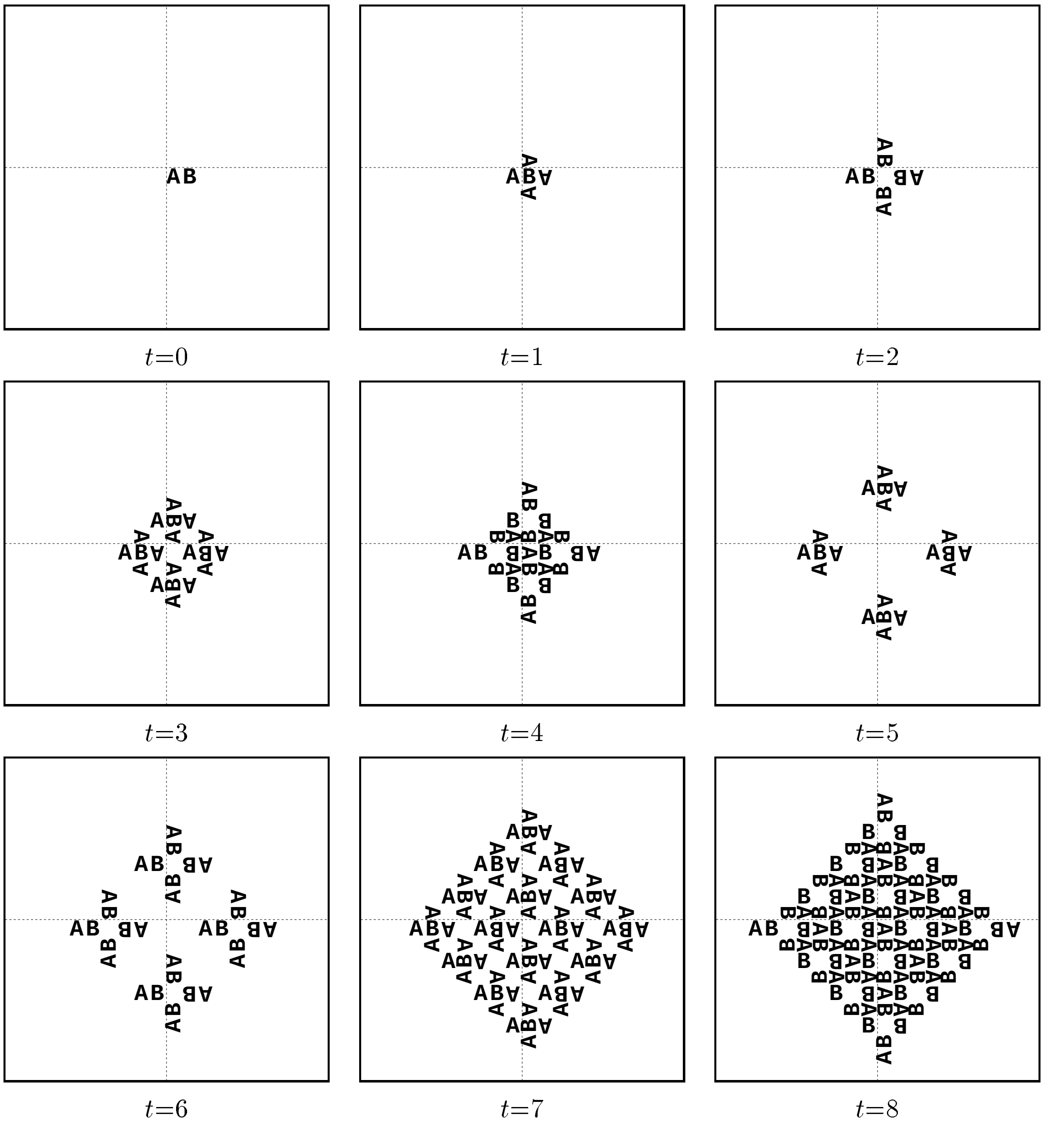


Figure C.14: Self-replicating structure $\texttt{UL2WC9V}_2$.

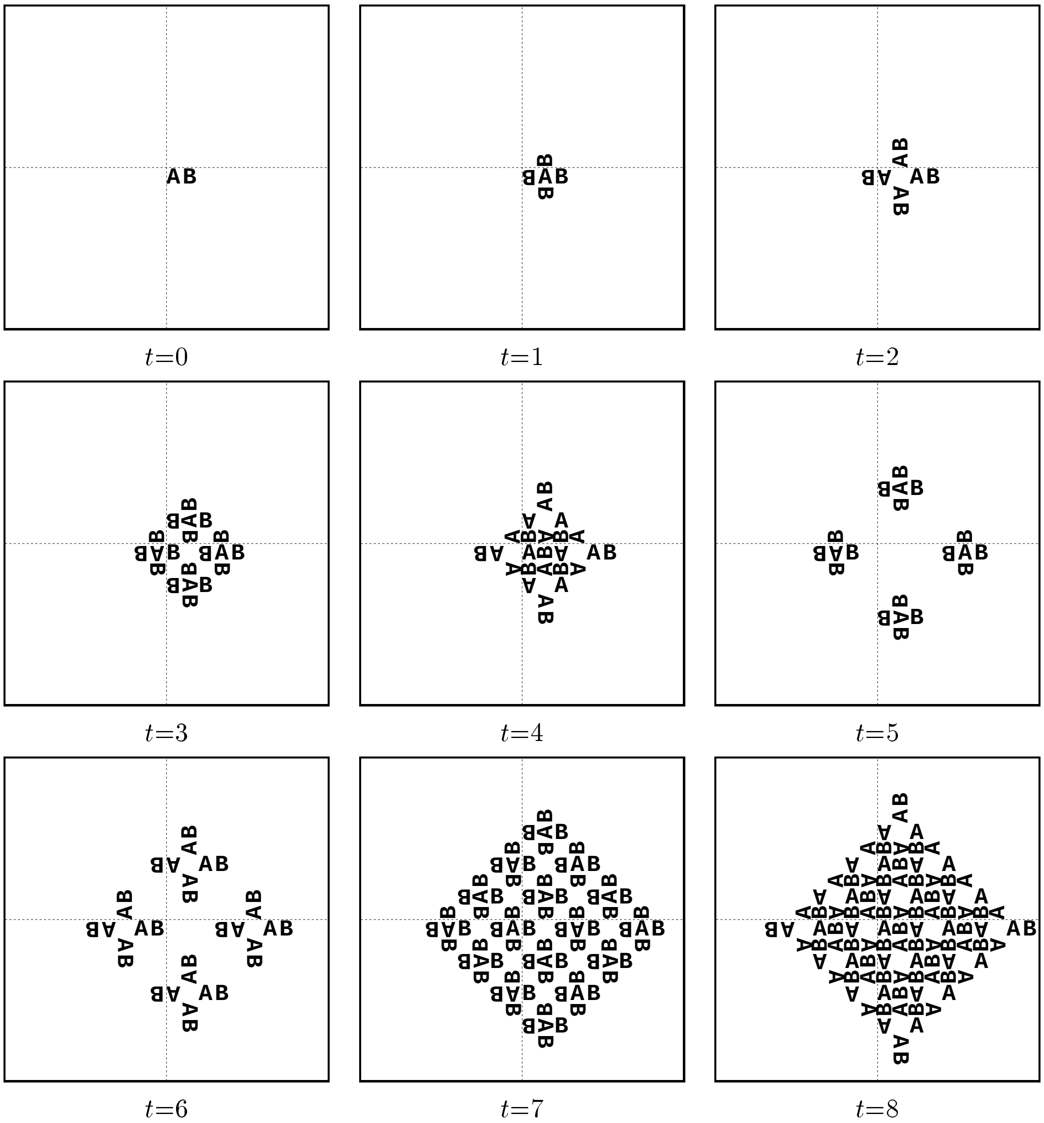


Figure C.15: Self-replicating structure UL2WC9V$_3$.

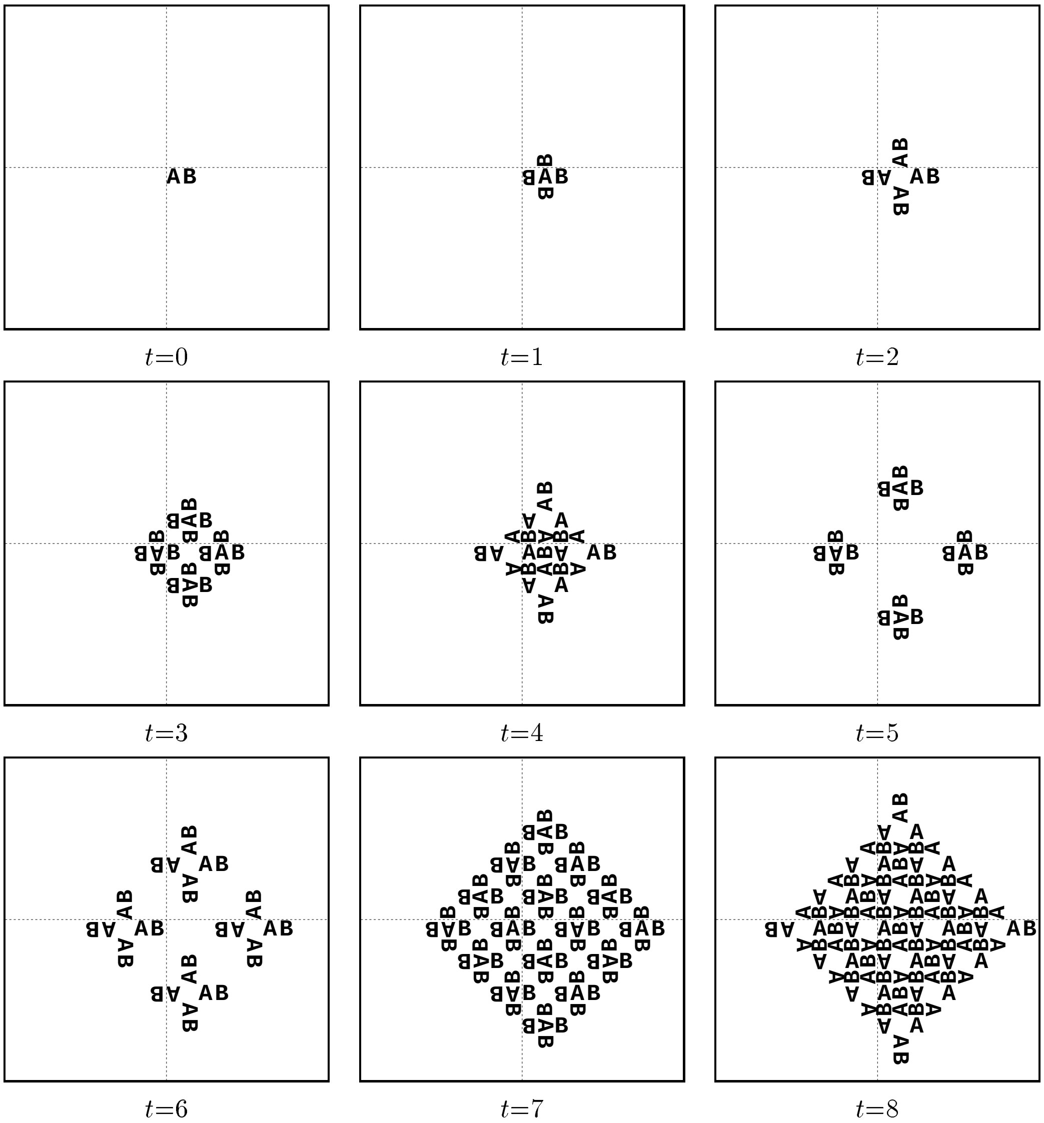
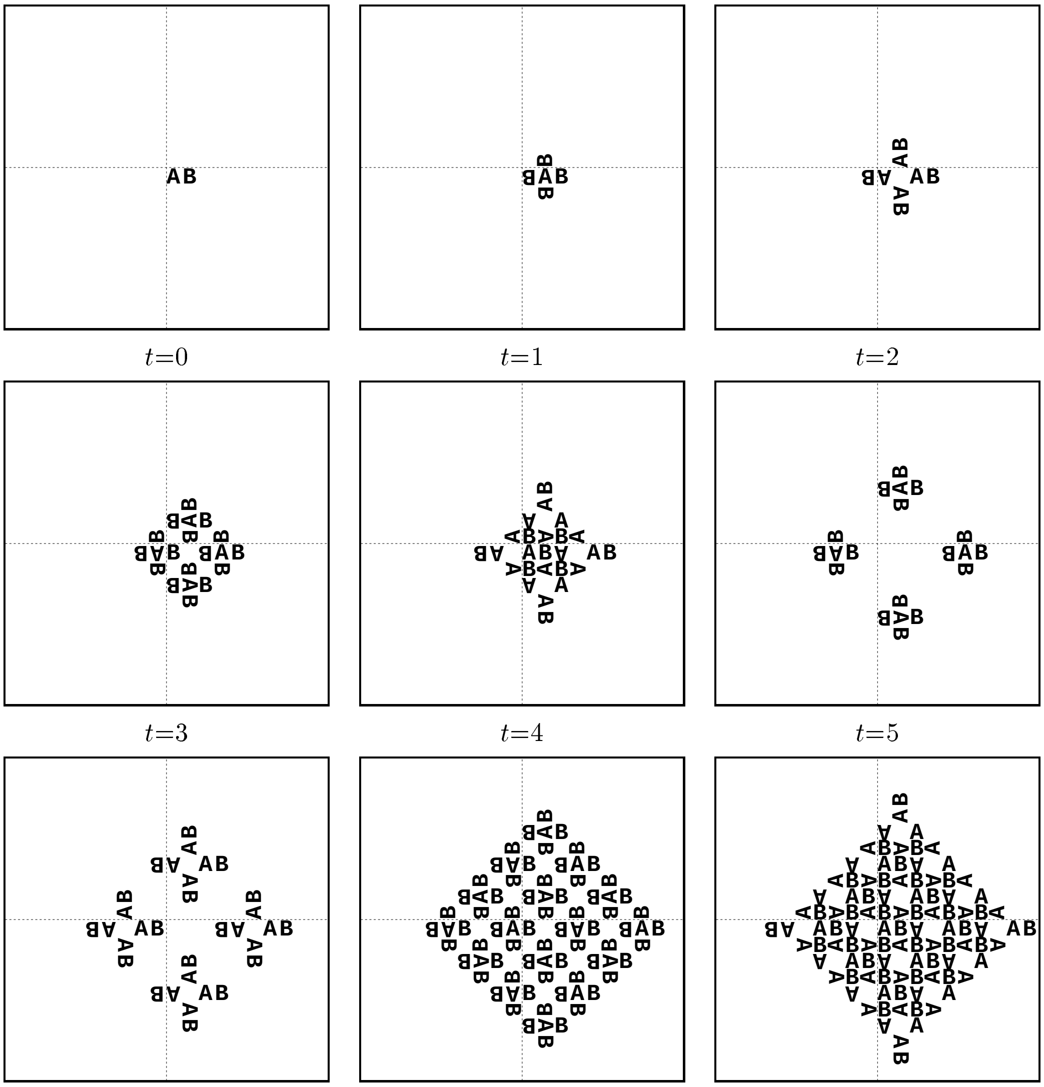


Figure C.16: Self-replicating structure $\mathtt{UL2WC9V}_8$.



t=0 t=1 t=2

t=3 t=4 t=5

t=6 t=7 t=8

Figure C.17: Self-replicating structure $\texttt{UL2WC9V}_9$.

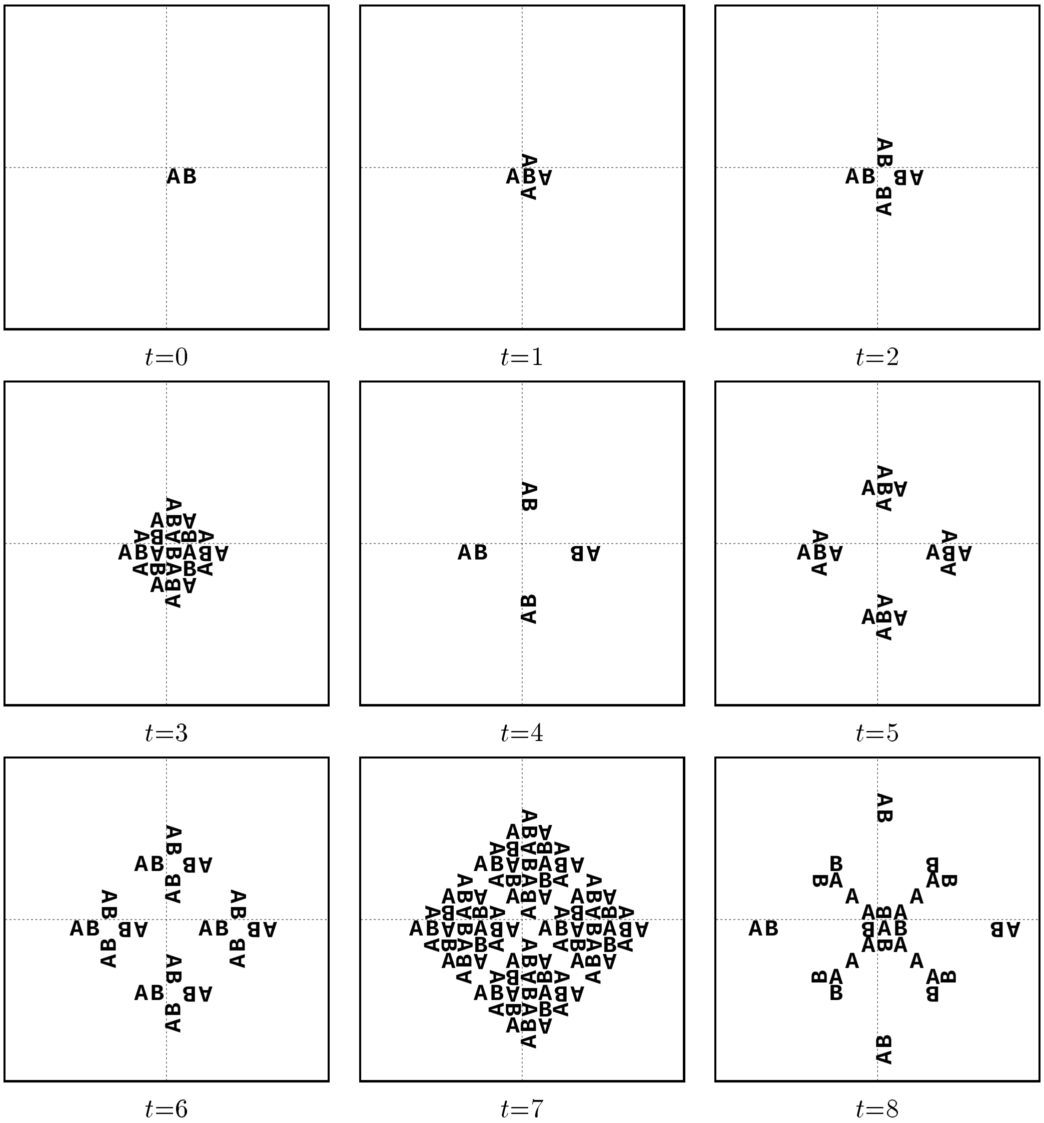


Figure C.18: Self-replicating structure $\texttt{UL2WC9V}_{10}$.

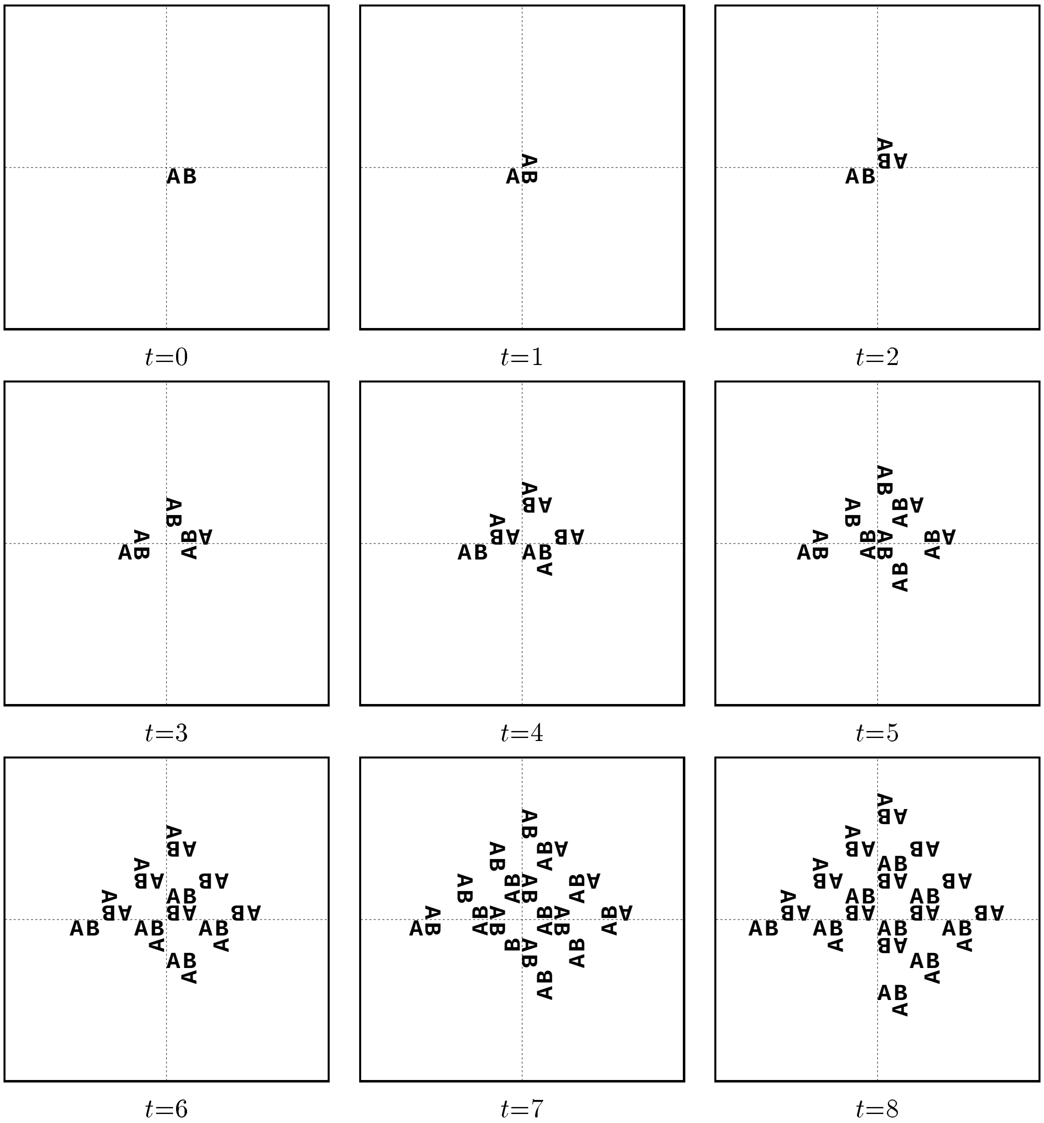


Figure C.19: Self-replicating structure UL2W9V$_4$.

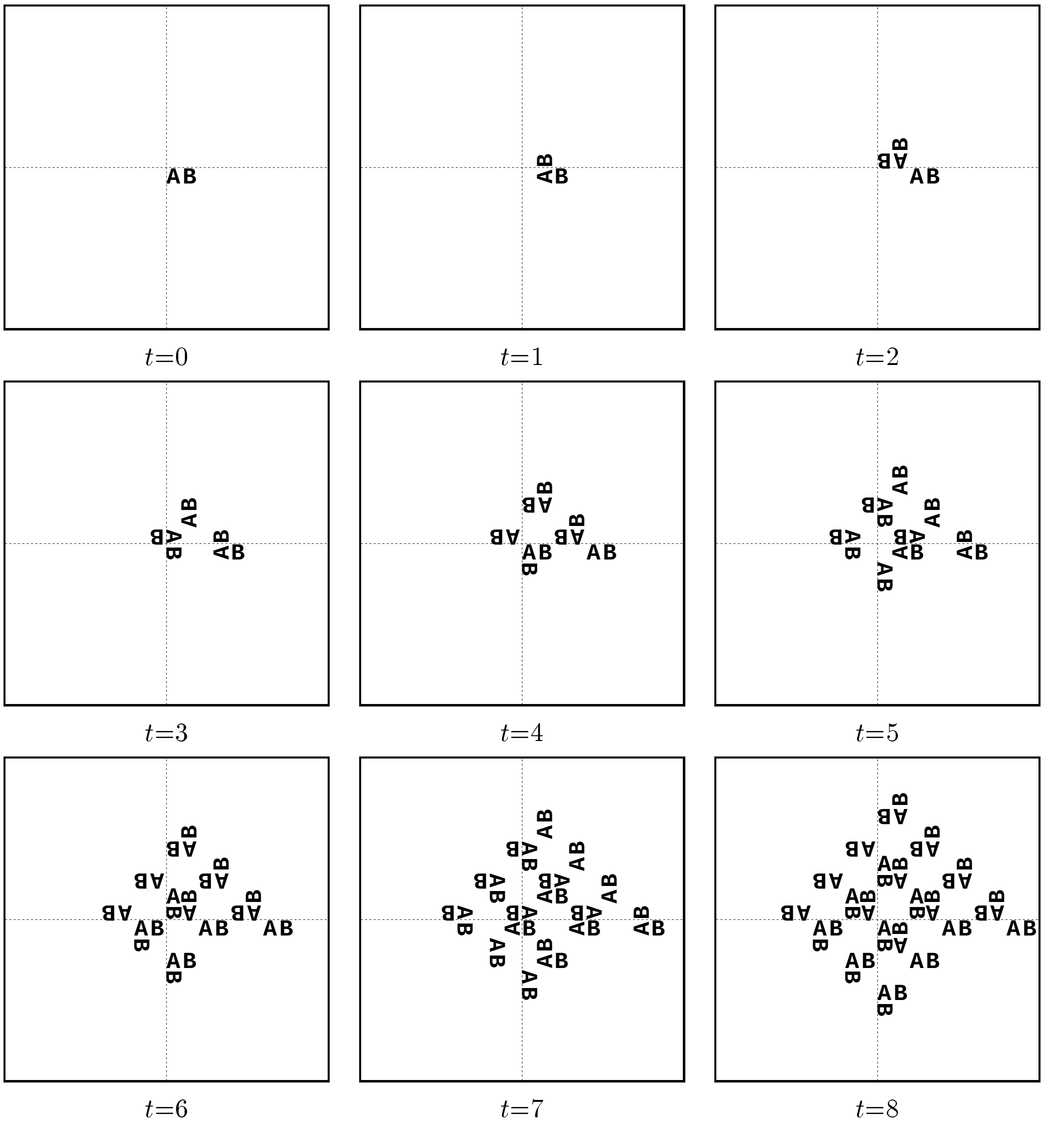


Figure C.20: Self-replicating structure $\mathtt{UL2W9V}_6$.

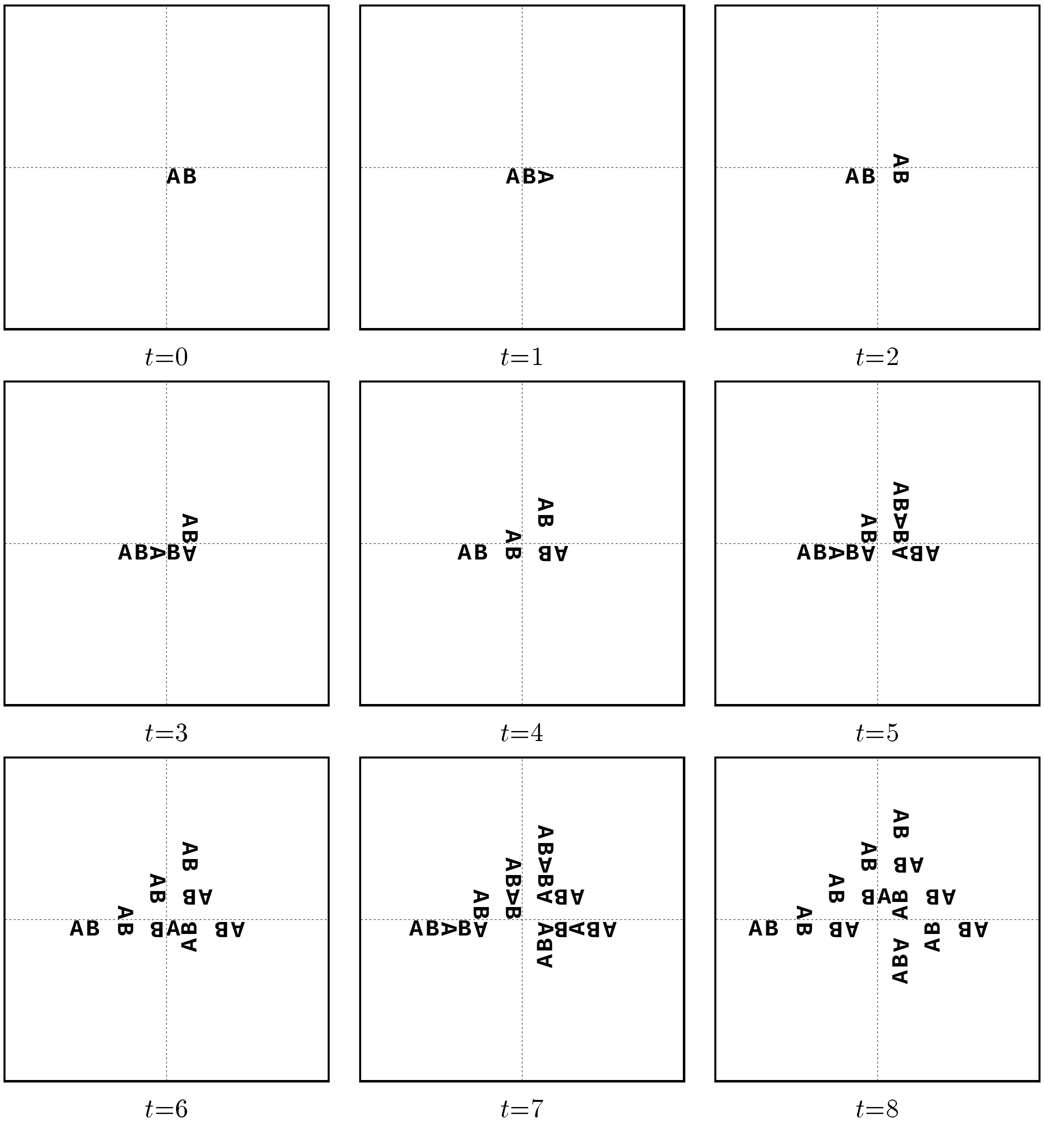


Figure C.21: Self-replicating structure $\mathtt{UL2W9V}_7$.

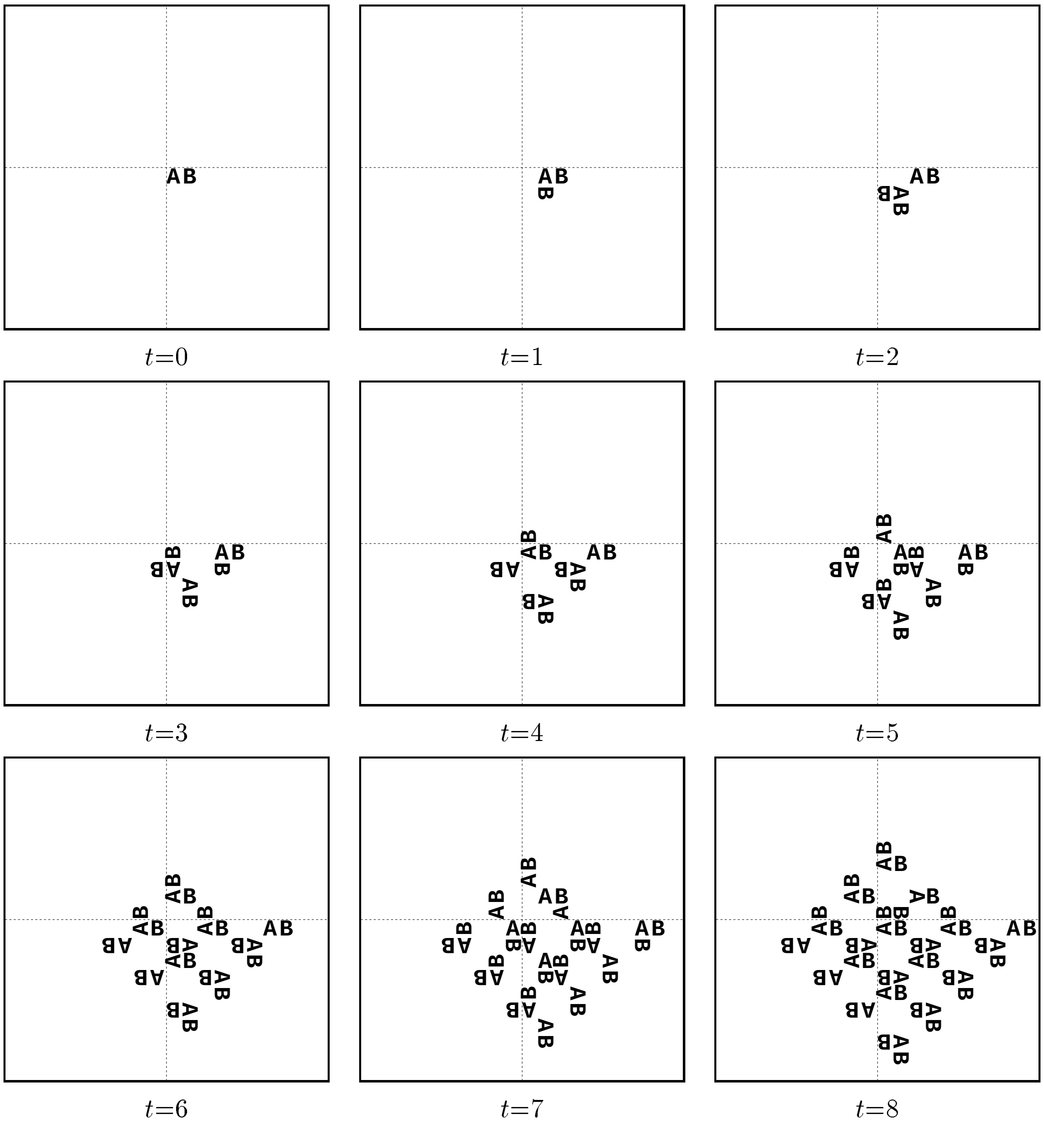


Figure C.22: Self-replicating structure $\texttt{UL2W9V}_9$.

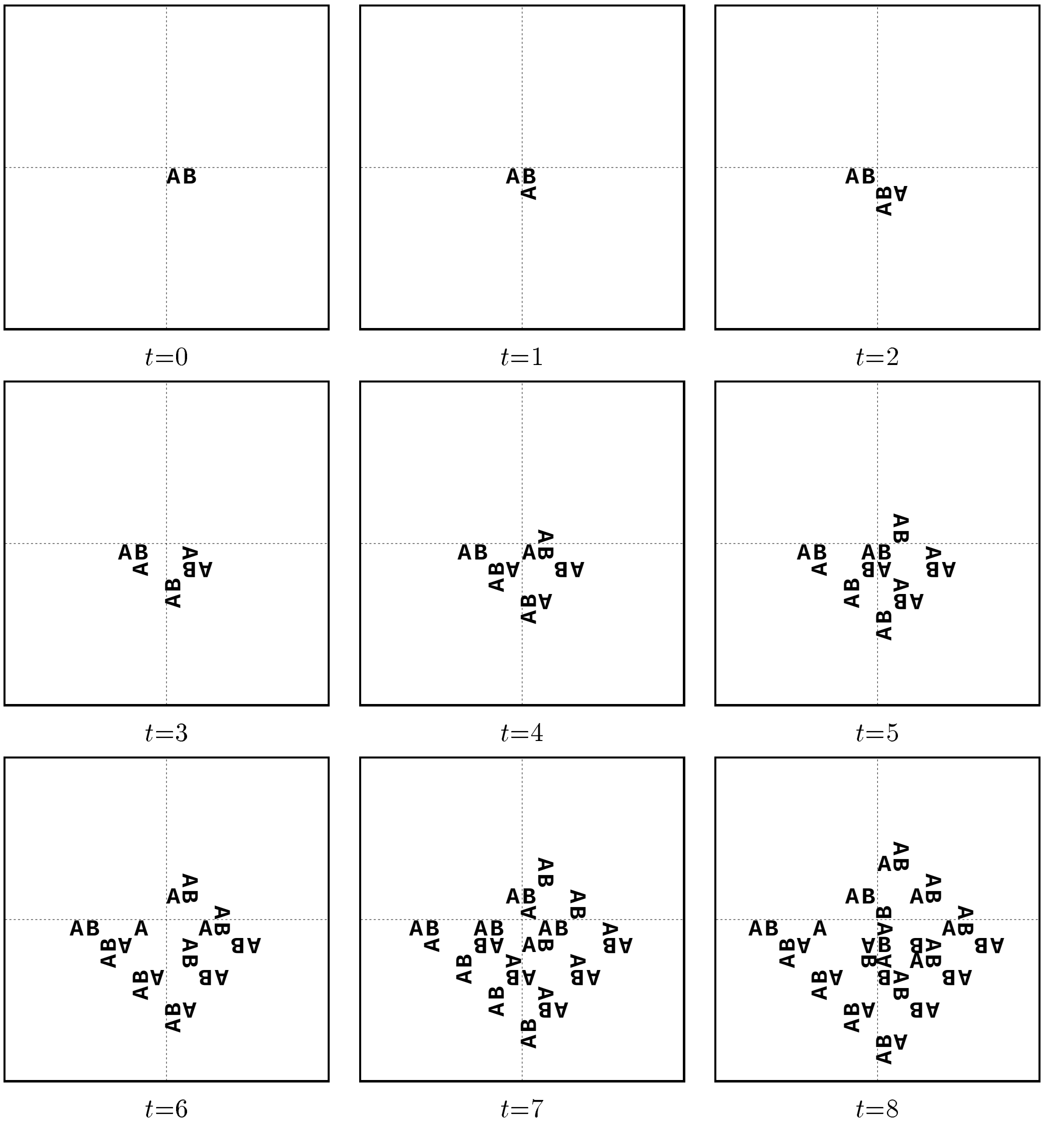


Figure C.23: Self-replicating structure $\mathtt{UL2W9V}_{10}$.

References

[Andre96] D. Andre, F. Bennett, and J. Koza, "Evolution of Intricate Long-distance Communication Signals in Cellular Automata using Genetic Programming." In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, 1996.

[Arbib66] M. A. Arbib, "Simple Self-Reproducing Universal Automata," *Information and Control*, Vol. 9, pp. 177–189, 1966.

[Arbib67] M. A. Arbib, "Automata Theory and Development: Part I," *Journal of Theoretical Biology*, Vol. 14, pp. 131–156, 1967.

[Arbib69a] M. A. Arbib, *Theories of Abstract Automata,* Prentice-Hall, Englewood Cliffs, NJ, 1969.

[Arbib69b] M. A. Arbib, "Self-Reproducing Automata: Some Implications for Theoretical Biology." In *Towards a Theoretical Biology*, Vol. 2, C. H. Waddington (ed), University of Edinburgh Press, Edinburgh, pp. 204–226, 1969.

[Baluja95] S. Baluja, "An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics," Carnegie Mellon School of Computer Science Technical Report CMU-CS-95-193, (Pittsburgh, PA, 15213).

[Bennett85] C. Bennett and R. Landauer, "The Fundamental Physical Limits of Computation," *Scientific American*, Vol. 253, July, pp. 48–56, 1985.

[Booker90] L. Booker, D. Goldberg, and J. Holland, "Classifier Systems and Genetic Algorithms." In *Readings in Machine Learning*, J. Shavlik, T. Dietterich (eds.), Morgan Kaufmann, San Mateo, CA, pp. 404–427, 1990.

[Brooks94] R. A. Brooks and P. Maes (eds), *Artificial Life IV*, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, 1994.

[Burks70] A. Burks, *Essays on Cellular Automata*, Univ. of Illinois Press, 1970.

[Burks74] A. Burks, "Cellular Automata and Natural Systems," *Cybernetics and Bionics*, Oldenbourg, Munich, 1974.

[Burks84] C. Burks and J. D. Farmer, "Towards Modeling DNA Sequences as Automata," *Physica D*, **10**, North-Holland, pp. 157–167, 1984.

[Byl89] J. Byl, "Self-Reproduction in Small Cellular Automata," *Physica D*, **34**, North-Holland, pp. 295–299, 1989.

[Conrad72] M. Conrad, "Information Processing in Molecular Systems," *Currents in Modern Biology*, **5**, North-Holland, pp. 1–14, 1972.

[Chou94] H. Chou, J. Reggia, R. Navarro-González, and J. Wu, "An Extended Cellular Space Method for Simulating Autocatalytic Oligonucleotides," *Computers Chem.*, Vol. 18, No. 1, pp. 33–43, 1994.

[Codd68] E. F. Codd, *Cellular Automata*, Academic Press, 1968.

[Crutchfield95] J. P. Crutchfield and M. Mitchell, "The Evolution of Emergent Computation," *Proceedings of the National Academy of Sciences*, Vol. 92, No. 23, p. 10742, November, 1995.

[Dandekar92] T. Dandekar and P. Argos, "Potential of Genetic Algorithms in Protein Folding and Protein Engineering Simulations," *Protein Engineering*, **5**, 7, pp. 637–645, 1992.

[Davis91] L. D. Davis (ed), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[Davidor91] Y. Davidor, "Genetic Algorithms and Robotics," *Robotics and Automated Systems*, World Scientific, Singapore, 1991.

[De Jong75] K. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor, 1975.

[De Jong87] K. De Jong, "On Using Genetic Algorithms to Search Program Spaces," *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 210–216, 1987.

[Drexler89] K. E. Drexler, "Biological and Nanomechanical Systems: Contrasts in Evolutionary Capacity." In [Langton88], pp. 501–519, 1989.

[Farmer91] J. D. Farmer, "Artificial Life: The Coming Evolution." In [Langton91a], pp. 815–840, 1991.

[Freitas82] R. Freitas and W. Gilbreath (eds), *Advanced Automation for Space Missions*, NASA Conference Publication 2255, National Technical Information Service, Springfield, VA, 1982.

[Frisch86] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas Automata for the Navier-Stokes Equation," *Physical Review Letters*, **56**, pp.1505, 1986.

[Gardner70] M. Gardner, "The Fantastic Combinations of John Conway's New Solitaire Game Life," *Scientific American*, **223**:4, pp. 120–123, 1970

[Ginsberg93] M. Ginsberg, *Essentials of Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1993.

[Goel89] N. S. Goel and R. L. Thompson, "Movable Finite Automata (MFA): A New Tool for Computer Modeling of Living Systems." In [Langton88], pp. 317–340, 1989.

[Goldberg89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass, 1989.

[Grefenstette86] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 1, pp. 122–128, 1986.

[Hall67] M. Hall, *Combinatorial Theory*, Blaisdell Publishing, Waltham, Mass., 1967.

[Harp91] S. A. Harp and T. Samad, "Genetic Synthesis of Neural Network Architecture." In [Davis91], pp. 202–221, 1991.

[Hartman86] H. Hartman and G. Vishniac, "Inhomogeneous Cellular Automata." In *Disordered Systems and Biological Organization*, E. Bienenstock (ed.), Springer-Verlag, Heidelberg, pp. 53–57, 1986.

[Hogeweg88] P. Hogeweg, "Cellular Automata as a Paradigm for Ecological Modeling," *Applied Mathematics and Computation*, **27**, pp. 81–100, 1988.

[Holland75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975.

[Holland76] J. H. Holland, "Studies of the Spontaneous Emergence of Self-Replicating Systems Using Cellular Automata and Formal Grammars." In *Automata, Languages, Development,* A. Lindenmayer and G. Rozenberg (eds), North-Holland, pp. 385–404, 1976.

[Holland80] J. H. Holland, "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases," *International Journal of Policy Analysis and Information Systems*, Vol. 4, pp. 217–240, 1980.

[Holland92] J. H. Holland, "Genetic Algorithms," *Scientific American*, July, pp. 66–72, 1992.

[Hong92] J.-I. Hong, Q. Feng, V. Rotello, and J. Rebek, Jr., "Competition, Cooperation, and Mutation: Improving a Synthetic Replicator by Light Irradiation," *Science*, **255**, pp. 848–850, 1992.

[Hopcroft79] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass, 1979.

[Jacobson58] H. Jacobson, "On Models of Self-Reproduction," *American Scientist*, **46**, pp. 255–284, 1958.

[Jefferson91] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang, "Evolution as a Theme in Artificial Life: The Genesys/Tracker System." In [Langton91a], pp. 549–578, 1991.

[Kanji93] G. Kanji, *100 Statistical Tests*, Sage Publications, London, 1993.

[Kephart94] J. O. Kephart, "A Biologically Inspired Immune System for Computers." In [Brooks94], pp. 130–139, 1994.

[Koza92] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[Koza94] J. Koza, "Artificial Life: Spontaneous Emergence of Self-Replicating and Evolutionary Self-Improving Computer Programs." In [Langton94], pp. 225–262, 1994.

[Laing76] R. Laing, "Automaton Inspection," *J. Computer and System Sciences*, **13**, pp. 172–183, 1976.

[Langley87] P. Langley, H. Simon, and G. Bradshaw, "Heuristics for Empirical Discovery." In *Computational Models of Learning*, L. Bolc (ed), Springer-Verlag, 1987.

[Langton84] C. G. Langton, "Self-Reproduction in Cellular Automata," *Physica D*, **10**, pp. 135–144, 1984.

[Langton86] C. G. Langton, "Studying Artificial Life with Cellular Automata," *Physica D*, **22**, pp. 120–149, 1986.

[Langton88] C. G. Langton (ed), *Artificial Life*, Santa Fe Institute Studies in the Sciences of Complexity, Vol. VI, Addison-Wesley, 1988.

[Langton90] C. G. Langton, "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation," *Physica D*, **42**, pp. 12–37, 1990.

[Langton91a] C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (eds), *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Vol. X, Addison-Wesley, 1991.

[Langton91b] C. G. Langton, "Life at the Edge of Chaos." In [Langton91a], pp. 41–91, 1991.

[Langton94] C. G. Langton (ed), *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVII, Addison-Wesley, 1994.

[Lenat77] D. Lenat, "The Ubiquity of Discovery," *Artificial Intelligence*, **9**, pp. 257–285, 1977.

[Lohn95] J. Lohn and J. Reggia, "Discovery of Self-Replicating Structures using a Genetic Algorithm," *1995 IEEE International Conference on Evolutionary Computing*, Perth, pp. 678–683, 1995.

[Lugowski89] M. Lugowski, "Computational Metabolism: Towards Biological Geometries for Computing." In [Langton88], pp. 341–368, 1989.

[McMullin92a] B. McMullin, "The Holland α-Universes Revisited." In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life,* F. Varela and P. Bourgine (eds), MIT Press, pp. 317–326, 1992.

[McMullin92b] B. McMullin, *Artificial Knowledge: An Evolutionary Approach,* Ph.D. Thesis, Department of Computer Science, The National University of Ireland, University College Dublin, 1992.

[Mange94] D. Mange and A. Stauffer, "Introduction to Embryonics: Towards New Self-repairing and Self-Reproducing Hardware Based on Biological-like Properties," Artificial Life and Virtual Reality, John Wiley, 1994.

[Mange95] D. Mange, S. Durand, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, and C. Piguet, "A New Self-Reproducing Automaton Based on a Multi-Cellular Organization." Dept. of Comp. Sci. Tech. Report, Swiss Federal Institute of Technology (Lausanne, Switzerland CH 1015), 1995.

[Merkle94] R. C. Merkle, "Self-replicating Systems and Low Cost Manufacturing." In *The Ultimate Limits of Fabrication and Measurement*, M.E. Welland, J.K. Gimzewski, (eds), Kluwer, Dordrecht, pp. 25–32, 1994.

[Miller74] S. L. Miller and L. E. Orgel, *The Origins of Life on Earth*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[Mitchell93] M. Mitchell, P. T. Hraber, and J. P. Crutchfield, "Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations," *Complex Systems*, **7** (2), pp. 89–130, 1993.

[Mitchell94] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, "Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments," *Physica D*, **75**, pp. 361–391, 1994.

[Mitchell96] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, 1996.

[Moore62] E. F. Moore, "Machine Models of Self-Reproduction," *Proc. Fourteenth Symp. on Applied Mathematics*, pp. 17–33, 1962.

[Myhill64] J. Myhill, "The Abstract Theory of Self-Reproduction," *Views on General Systems Theory, Proc. Second Systems Symp. at Case Inst. of Technology*, M. Mesarovic (ed), Wiley, pp. 106–118, 1964.

[Orgel92] L. E. Orgel, "Molecular Replication," *Nature*, 358, pp. 203–209, 1992.

[Packard88] N. H. Packard, "Adaptation Toward the Edge of Chaos." In J.A.S. Kelso, A.J. Mandell, and M.F.Shlesinger (eds), *Dynamic Patterns in Complex Systems*, pp. 293–301, World Scientific, Singapore, 1988.

[Parsons93] R. Parsons, S. Forrest, and C. Burks, "Genetic Algorithms for DNA Sequence Assembly," *Proc. First Intl. Conf. on Intelligent Systems for Molecular Biology*, pp. 310–318, 1993.

[Penrose58] L. Penrose, "Mechanics of Self-Reproduction," *Ann. Human Genetics*, 23, pp. 59–72, 1958.

[Perdang93] J. M. Perdang and A. Lejune (eds), *Cellular Automata: Prospects in Astrophysical Applications*, World Scientific, Singapore, 1993.

[Preston84] K. Preston and M. Duff, *Modern Cellular Automata*, Plenum, New York, 1984.

[Ray92] T. Ray, "Evolution, Ecology and Optimization of Digital Organisms," *Santa Fe Institute Working Paper 92-08-042*, 1992.

[Richards90] F. C. Richards, T. P. Meyer, and N. H. Packard, "Extracting Cellular Automaton Rules Directly from Experimental Data," *Physica D*, **45**, pp. 189–202, 1990.

[Reggia93] J. A. Reggia, S. Armentrout, H. H. Chou, and Y. Peng, "Simple Systems That Exhibit Self-Directed Replication," *Science*, 259, pp. 1282–1288, February, 1993.

[Shahookar90] K. Shahookar and P. Maxumder, "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization," *IEEE Transactions on Computer-Aided Design*, **9** (5), pp. 500–511, 1990.

[Stephenson92] I. Stephenson and R. Taylor, "Creatures: A Simulation Environment for Autonomous Behavior," Technical Report ASEG.92.16, University of York (York, England, Y01 5DD), 1992.

[Taub61] A. H. Taub, *John von Neumann: Collected Works. Volume V: Design of Computer, Theory of Automata and Numerical Analysis*, Oxford, Pergamon Press, 1961.

[Thatcher70] J. W. Thatcher, "Universality in the von Neumann Cellular Model." In [Burks70], pp. 132–186, 1970.

[Toffoli87] T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, 1987.

[Varela74] F. G. Varela, H. R. Maturana, and R. Uribe, "Autopoiesis: The Organization of Living Systems, its Characterization and a Model," *BioSystems*, 5, pp. 187–196, 1974.

[Vitányi73] P. M. B. Vitányi, "Sexually Reproducing Cellular Automata," *Mathematical Biosciences*, **18**, pp. 23–54, 1973.

[von Neumann51] J. von Neumann, "The General and Logical Theory of Automata." In [Taub61], pp. 288–328, 1951.

[von Neumann66] J. von Neumann, *Theory of Self-Reproducing Automata*, A. Burks (ed), University of Illinois Press, 1966.

[Watson87] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, *Molecular Biology of the Gene*, 4th ed., Benjamin/Cummings, Menlo Park, 1987.

[Wilson89] S. W. Wilson, "The Genetic Algorithm and Simulated Evolution," in [Langton88], 157–166, 1989.

[Wolfram84] S. Wolfram, "Universality and complexity in cellular automata," *Physica D*, **10**, pp. 1–35, 1984.

[Wolfram86] S. Wolfram, *Theory and Applications of Cellular Automata*, World Scientific, Singapore, 1986.

[Wolfram94] S. Wolfram, *Cellular Automata and Complexity*, Addison-Wesley, Addison-Wesley, Reading, Mass, 1994.