ELSEVIER

Physica D 110 (1997) 252–276

PHYSICA D

Emergence of self-replicating structures in a cellular automata space

Hui-Hsien Chou [a,1], James A. Reggia [b,*]

[a] The Institute for Genomic Research, 9712 Medical Center Drive, Rockville, MD 20850, USA
[b] Department of Computer Science and Institute for Advanced Computer Studies, A.V. Williams Building, University of Maryland, College Park, MD 20742, USA

Received 10 April 1997; received in revised form 25 May 1997; accepted 30 May 1997
Communicated by A.M. Albano

Abstract

Past cellular automata models of self-replication have always been initialized with an original copy of the structure that will replicate, and have been based on a transition function that only works for a single, specific structure. This article demonstrates for the first time that it is possible to create cellular automata models in which a self-replicating structure emerges from an initial state having a random density and distribution of individual components. These emergent self-replicating structures employ a fairly general rule set that can support the replication of structures of different sizes and their growth from smaller to larger ones. This rule set also allows "random" interactions of self-replicating structures with each other and with other structures within the cellular automata space. Systematic simulations show that emergence and growth of replicants occurs often and is essentially independent of the cellular space size, initial random pattern of components, and initial density of components, over a broad range of these parameters. The number of replicants and the total number of components they incorporate generally approach quasi-stable values with time.

Keywords: Self-replication; Self-organization; Cellular automata

1. Introduction

Attempts to create artificial self-replicating structures or "machines" have been motivated by the desire to understand the fundamental information processing principles involved in self-replication [1]. Understanding these principles could contribute to a better understanding of biological replication, shed light on the origins of life, and support atomic-scale manufacturing (nanotechnology) [6]. While a variety of approaches have been taken, the use of cellular automata models has been among the most prominent.

Cellular automata can be viewed as dynamical systems in which both time and space are discrete. John von Neumann first conceived of using cellular automata to study the logical origanization of self-replicating structures [18]. In his and subsequent two-dimensional cellular automata models space is divided into cells, each of which

* Corresponding author. Tel.: 301 405 2686; fax: 301 405 6707; e-mail: reggia@cs.umd.edu.
[1] E-mail: hhchou@tigr.org.

can be in one of n possible states. At any moment most cells are in a distinguished "quiescent" or inactive state (designated by a period or blank space in this article) whereas the other cells are said to be in an active state. A self-replicating structure or "machine" is represented as a configuration of contiguous active cells, each of which represents a *component* of the replicating machine. At each instance of simulated time, each active cell or component follows a set of rules called the *transition function* to determine its next state as a function of its current state and the state of immediate neighbor cells. Thus, any process of self-replication captured in a model like this must be an emergent behavior arising from the strictly local interactions that occur. Based solely on these concurrent local interactions, an initially specified self-replicating structure goes through a sequence of steps to construct a duplicate copy of itself (the replica being displaced and perhaps rotated).

Von Neumann's original self-replicating structure was a complex "universal constructor-computer" embedded in a two-dimensional cellualr automata space that consisted of 29-state cells. Much subsequent work has focused on creating simpler replicants, for example, demonstrating that if the components or cell states meet certain symmetry requirements, then von Neumann's configuration could be done in a simpler fashion using cells having only eight states [5]. More recently, a qualitatively simpler structure referred to as a *self-replicating loop* was developed [8]. Self-replicating loops have a readily identifiable stored "instruction sequence" that is used by the underlying transition function in two ways: as instructions that are interpreted to direct the construction of a replica, and as uninterpreted data that are copied onto the replica. Thus, replicating loops are truely "information replicating systems" in the sense that this term is used by organic chemists [11]. Subsequent work led to progressively simpler and smaller loops [2,14]; see [15] for a recent review.

These past cellular automata models of self-replication have always been initialized with an original copy of the structure that will replicate, and have been based on rules that only work for a single, specific structure. The replicants exist in an otherwise empty space. This article demonstrates for the first time that it is possible to create cellular automata models in which a simple self-replicating structure (loop) emerges from an initial state having a random density and distribution of components (the "primordial soup"). These emergent self-replicating loops employ a general purpose rule set that can support the replication of loops of different sizes and their growth from smaller to larger ones. This rule set also allows random changes of loop sizes and interaction of self-replicating loops within a cellular automata space containing free-floating components. In contrast to another recent study using very different methods than those described here [12], our model demonstrates the emergence of very simple replicants that evolve to larger ones (rather than the opposite trend). We examine this difference in Section 5.

This paper is organized into three parts: a brief example simulation illustrating the emergence and growth of self-replicating loops, an explanation of the transition function underlying this behavior,[2] and the results of simulations done while varying model parameters. These latter, systematic simulations show that emergence and growth of replicants is very robust, occurring often and essentially independently of the cellular space size, initial random pattern of components, and initial density of components over a broad range of these parameters. The number of replicants and the total number of components they incorporate approach quasi-stable values with time. A final section discusses the implications of this work and compares it to some recent related results.

2. An example

A continuous example running in a randomly initialized, small (40×40) cellular automata space using an initial component density of 25% is shown in Fig. 1. We use the term "component" to refer to any single cell that is active. Periodic boundary conditions are used (opposite edge are taken as connected), so the space is effectively a torus.

[2] The complete transition function is available via the internet at http: //www.cs.umd.edu/~hhchou/download.html.

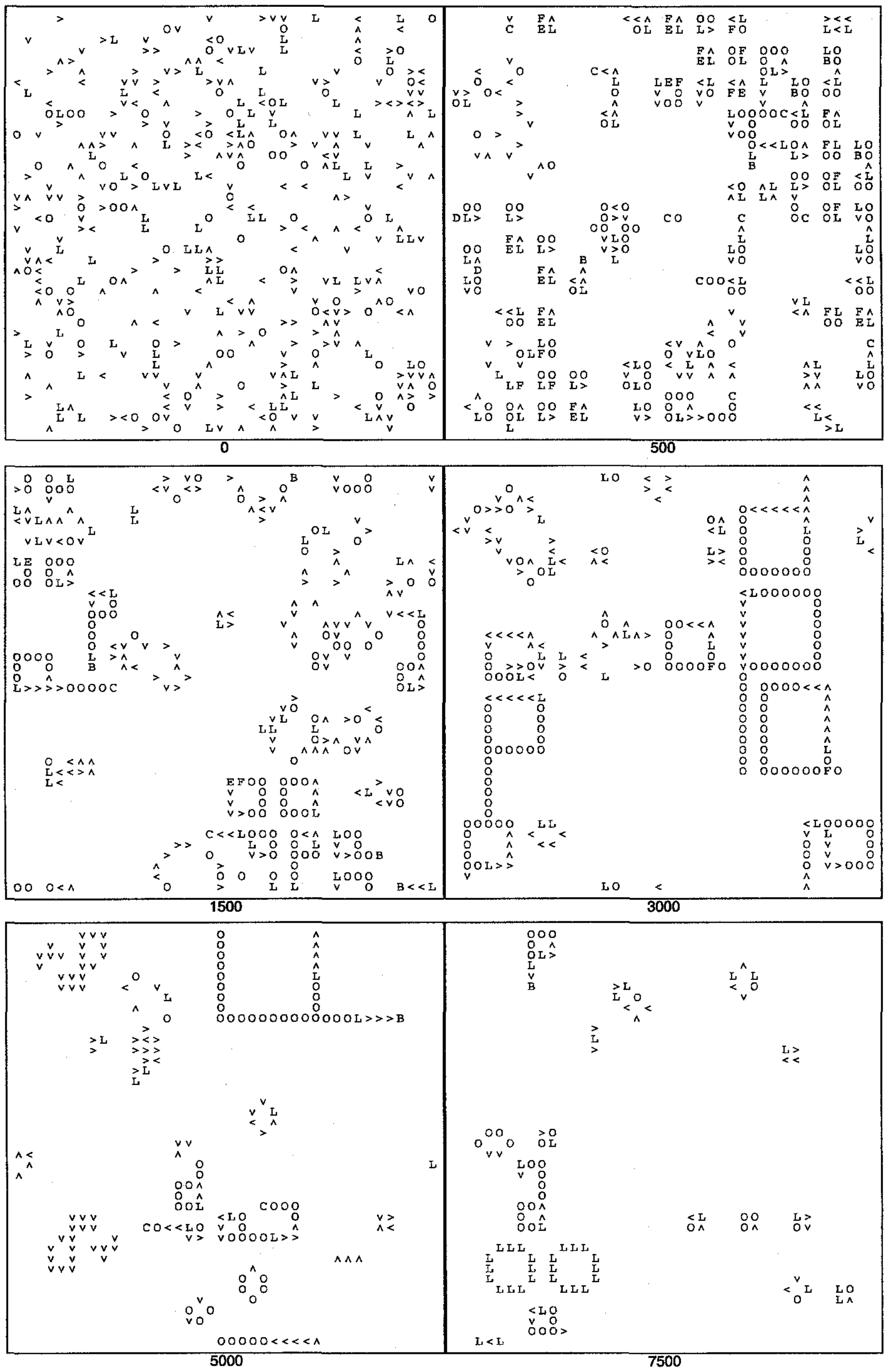


Fig. 1. A running example of emergent self-replication. Epoch numbers are shown.

Initially, at epoch or time $t = 0$ (upper left, Fig. 1), the space is 25% filled by randomly placed, non-replicating components designated graphically as

O > L C B E F or D,

while cells in the quiescent state are indicated by blank space. All components have strong rotational symmetry [14] except > which is viewed as being oriented, i.e., its orientation is significant.

This simulation is characterized by the initial emergence of very small, self-replicating loops and their progressive evolution to increasingly large and varied replicants. During this process a replicant may collide with other loops or with free-floating components, and either recover or self-destruct. Thus, by epoch 500 (upper right of Fig. 1), very small self-replicating loops of size 2×2 and 3×3 are present. By epoch 1500 a 4×4 loop is about to generate a 5×5 loop in the middle left region. At epoch 3000 the biggest loop is 8×8 and it is about to generate a 9×9 loop. By epoch 5000 many very large loops have annihilated each other and only one intact 10×10 loop is left. By epoch 7500 all large loops have died, but there are new 3×3 loops in the space. These loops will replicate and it is not clear when (if ever) this example will cease its activity. In this example, the size of the replicating structures became too big to fit comfortably in such a small world (40×40 only), and the large loops started to annihilate each other.

3. A general purpose self-replication rule set

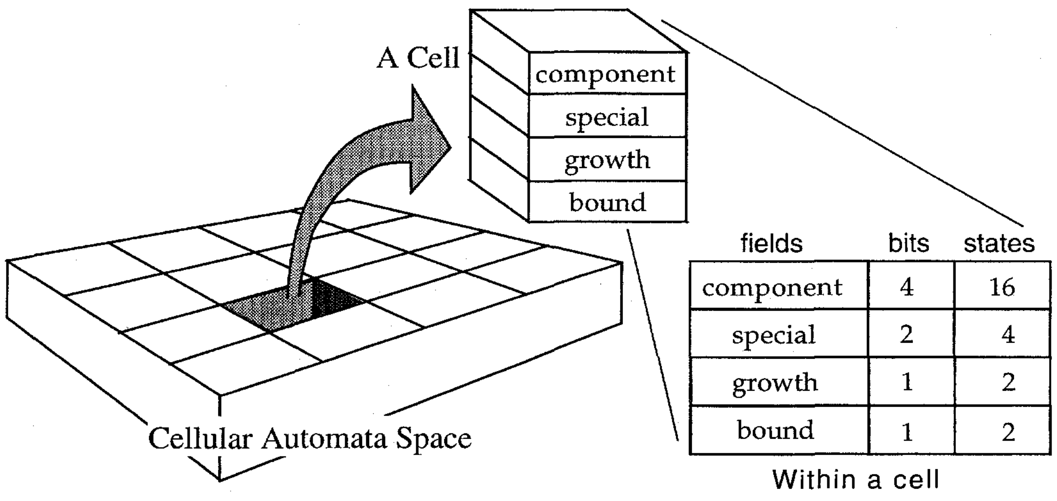
As can be seen from the above example, the transition function supporting these self-replicating loops differs from those used in previous cellular automata models of self-replication in several ways. A self-replicating structure emerges from an initial random state rather than being given, replication occurs in a milieu of free-floating components, and replicants grow and change their size over time, undergoing annihilation when replication is no longer possible. All of this occurs in the presence of a single transition function based on the Moore neighborhood, which we now consider. The complete rule set is specified in Appendix A.

The transition function is based on a functional division of data fields [10,17]. As seen in Fig. 2, the original bit depth of a cellular automata cell (in our case 8 bits) is functionally divided into four different *fields* (4, 2, 1 and 1 bit each) such that each field encodes different meanings and functions to the rule writer. The utilization of field divisions greatly simplifies the cellular automata rule programming effort, and makes the resulting rules much more readable. In the illustrations in this paper, only the component field is shown unless explicitly indicated otherwise.

3.1. Data fields

As noted above, each non-quiesent or active cell is taken to represent a potential "component" of a cellular automata structure. A cellular automata structure can be just a single cell, i.e., one with no conceptual connection with any adjacent non-quiescent cells, and in that case we call it an *unbound component*. On the other hand, a cellular automata structure can consist of several contiguous nonquiescent cells that are functionally interrelated, behaving as a whole, such as a self-replicating loop. In the latter case we call the structure a multi-component structure or simply a structure, and we call its components *bound components* (their *bound bit* is set; see Figs. 2 and 3).

Some cell states direct specific steps during the self-replication process and can be viewed as moving through the other components within the cellular automata structure. We will refer to these as *signals* in the self-replicating process. A sequence of signals on a cellular automata structure comprises the *signal sequence* (algorithm) that describes and controls the self-replication process. For example, in Fig. 3 those non-quiescent states whose bound



fields	bits	states
component	4	16
special	2	4
growth	1	2
bound	1	2

Fig. 2. The 8 bit state variable in each cell is conceptually sliced into four different bit groups called *fields*. Each field represents a specific piece of information.

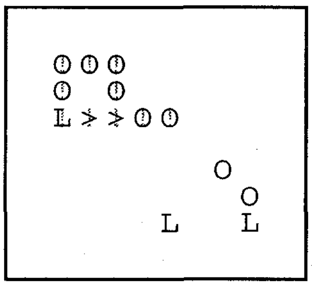


Fig. 3. A 3 × 3 self-replicating loop (upper left). Bound components are defined by set bound bits which are marked in light gray. Bound components are treated differently by the transition function than unbound ones.

bit is set are at the upper left region of the figure. There are three signals in this cellular automata structure, i.e., L>>.

The four data fields (see Fig. 2) and their states in the transition function are:

– The four-bit **component** field accounts for most normal operations of cellular automata structures. It encodes its 12 state values (out of 16 possible) as follows:

O The building block used in self-replicating loops; allows passage of a signal sequence, providing a pathway for the flow of information.

> The extrude signal that directs the extension/growth of a cellular automata signal pathway into the adjacent quiescent space. This actually represents four states $>$, $\vee$, $<$ and $\wedge$ which are designated in the rule set as '>', '>, 1', '>, 2' and '>, 3', respectively.

B The birth component left by the extrude signal >. A quiescent neighbor will first be converted into this state before it becomes part of a cellular automata structure.

L The left turn signal that changes the direction of growing signal path by 90° counterclockwise.

C The corner component left by the turning signal L. A signal > reaching it will be rotated 90° counterclockwise to form a corner.

EF This pair of signals in sequence directs the branching of a signal pathway.
D The detachment component which separates parent and child structures.
There is also the quiescent state which is denoted by '.' when referenced in rules; quiescent states are shown as white space in all figures.

- The two-bit special field denotes special situations that arise occasionally in the cellular automata space. There are four possible states:
  '.' No special situation.
  '*' A branching signal sequence (EF) will be generated.
  '–' A cell will not allow a signal sequence to pass, effectively deleting the signal sequence.
  '#' A bound component in the dissolve mode; will become unbound in the next epoch.
- The one-bit growth field, if set (denoted by '+'), marks a stimulus that may cause the existing signal sequence to increase in length.
- The one-bit bound field, if set (denoted by '!'), marks a cell as part of a multi-cell structure; otherwise the cell is an unbound component.

3.2. Signal transmission

The basic building block of a self-replicating loop is the component O which allows a stored signal sequence to "flow through it". The general form of the signal sequence which is transmitted in our model is exemplified by L>>>>, where an arbitrary number of signals > are followed by a signal L (the signals flow toward the right here; so they are read from right to left). A typical example of signal sequence flow is shown in Fig. 4, being similar to that used in past self-replicating loops [14].

The transition function rules in our model are based on the Moore neighborhood. They can be precisely represented in a high-level language for clarity, where neighbor position prefixes are designated no, ne, ea, se, so, sw, we and nw, for north, northeast, east, southeest, south, southwest, west and northwest, respectively. North is arbitraily taken to be up. For example, the rules that implement the signal sequence flow illustrated in Fig. 4 are:

```
if (component=='O') rot if (we:component=='>') component='>';
rot if (component=='>') if (we:component=='L') component='L';
                        else if (we:component=='>') component='>';
if (component=='L') component='O';
```

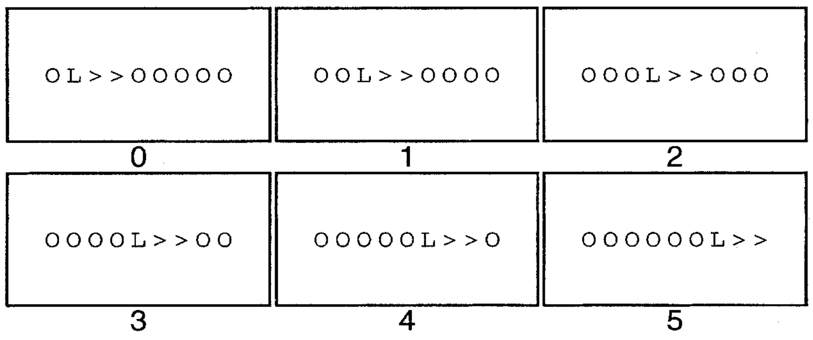


Fig. 4. Signal sequence flow. Each box is a snapshot of the same region in the cellular automata space during successive epochs (iterations). Numbers denote the epochs. Signal > is followed by either a signal > or by the signal L. Signal L always changed to O, the latter changing to > if pointed at by >.

The first, outer *if* statement says that if there is a component O with a signal > as its west neighbor that points at the O, then component O will change to > in the next epoch. The rotated if prefix *rot* of the inner *if* statement expands this condition to the other three equivalent possibilities in this isotropic space, i.e., a ∨ to the north, a < to the east or a ∧ to the south. The second, outer *if* statement says that if the west neighbor of a > signal is an L signal, the > signal will change to L, but if the west neighbor is instead a >, it will stay as >. Again, the rotated prefix *rot* expands the condition to the other three cases: a signal ∨ with an L or a ∨ as a north neighbor, asignal < with an L or a < as an east neighbor, and a signal ∧ with an L or a ∧ as a south neighbor. The last *if* statement simply says L always change to O.

3.3. Extended replication

The complete set of rules forming the transition function support replication of loops in a fashion similar to that used in the past [8,14]. As illustrated in Fig. 5, as a signal sequence circulates around a loop, it sends duplicate copies out an arm that triggers the extension, turning, rejoining, and separation of the child loop. Usually a self-replicating loop will produce another loop with the same size and signal sequence as itself, as shown in Fig. 5. This can change if the loop's signal sequence is lengthened, as shown in Fig. 6, resulting in a larger child loop. We refer to this process wherein a loop's replicant is of a different (larger) size as *extended replication*. A loop's signal sequence can become modified to generate different structures than itself by an active growth field appearing in one of its cells during the arm branching process. The growth field (Fig. 2) is set when a loop dies, as explained below.

When extended replication is under way, there is a timing problem, as seen in Fig. 6. The problem is that, due to the different sizes of parent and child structures, there will be an extra partial signal sequence in the new loop, together with a complete and correct signal sequence. This partial signal sequence must be erased to guarantee a

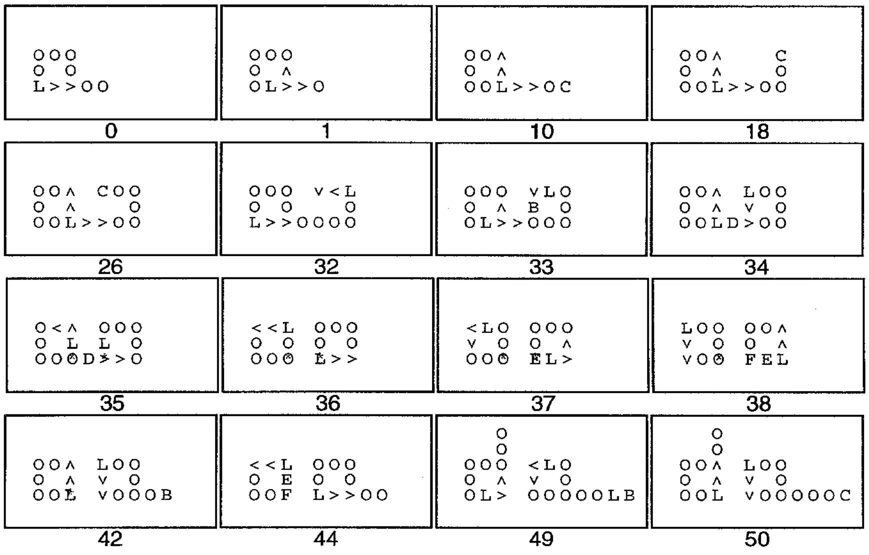


Fig. 5. Replication and separation of loops in isolation. The original 3 × 3 loop (epoch 0) goes through a series of extruding and turning steps (epochs 1–32) until finally its arm closes on itself (epoch 33). Component C designates a location where a corner (left turn) will form, while component B indicates the "birth" of a new component. When a loop closes on itself, D forms in the connecting cell between the two loops (epoch 34). D in turn triggers the setting of '*' values in the special field for both loops, shown in a light gray color at epoch 35. These special markers result in both separation of the loops and altered signal sequences in both loops (EF suffix at epochs 38 and 44), causing the generation of a new arm in each loop.

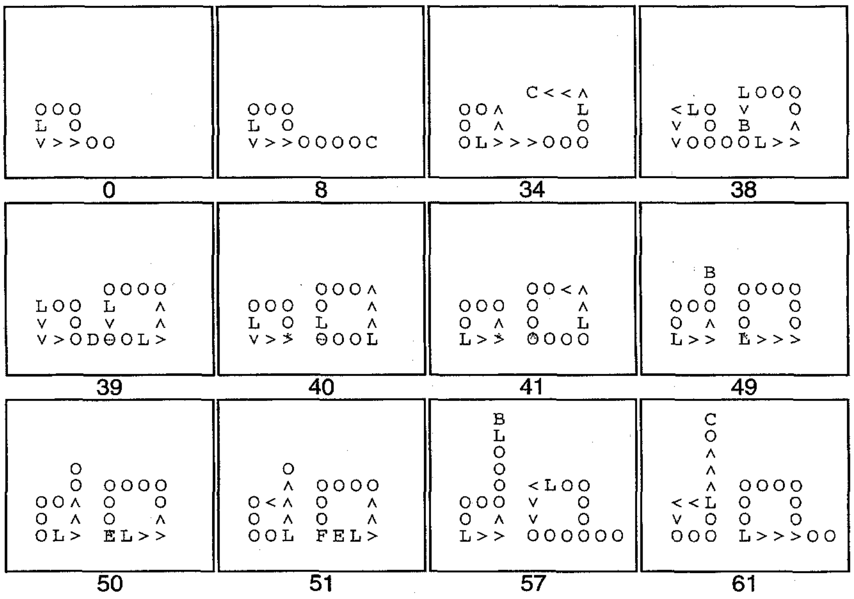


Fig. 6. Extended replication of loops. In epoch 0 the parent loop has one more > signal than it normally has (compare to Fig. 5). This causes a bigger loops to be generated. By epoch 8 this signal sequence generates a longer arm. By epoch 34 the arm has completed about 3/4 of its closing process. At epoch 38 the arm closes on itself, causing D to be set in the connecting cells as usual, and a '–' value to appear in the special field of the child loop (epoch 39). Setting the special field immediately stops that O from passing a signal sequence until an incomming signal L appears (epoch 40). Then the special field changes to the value '$*$' as in epoch 41 (L is not copied). From then on the new arm branching process is under way. Note that the parent (left) loop completes its arm branching process and starts a new replication cycle in epoch 50. The larger child loop (right) takes longer to finish replication (in epoch 61 the child loop is complete).

healthy new loop. This is achieved by setting the special field to '–' in the closing corner cell of the new loop. The cell in state O will detect an extended replication in progress and will set its special field accordingly. Once its special field is set to '–', that O stops copying signal sequences, thus effectively erasing any signal sequence going through it, as shown in epochs 39–41 in Fig. 6. This erasing process ends when the O detects an incoming L, which is the tail of the signal sequence being erased, and the cell in state O resets the special field '–' to '$*$' to start the process of generating a new arm.

Cellular automata rules that support extended replication are new. In the past, a different rule set has been required for each size replicating loop [14]; here the emergence of different size loops and their simultaneous replication is supported by a single rule set. This is very important, since it permits an initially small emergent self-replicating structure to grow in size. Unlike past self-replication rules, those described here abstract the self-replication phenomena out of any particular assumption about size of the self-replicating structures.

3.4. Collision detection and clean up

In all past work on self-replicating loops, replication occurs in an otherwise empty space and the transition function does not need to handle unanticipated events. In other words, while writing the rules one has complete control over the behaviors occurring in the cellular automata space, including the initial state. In contrast, here the very first assumption is that there is no a priori knowledge about the interactions between self-replicating loops, or what the cellular automata space is like in epoch zero. Although the behaviors (and their associated rules) considered so far can reliably direct a structure to do replication or extended replication in isolation, they cannot guarantee that

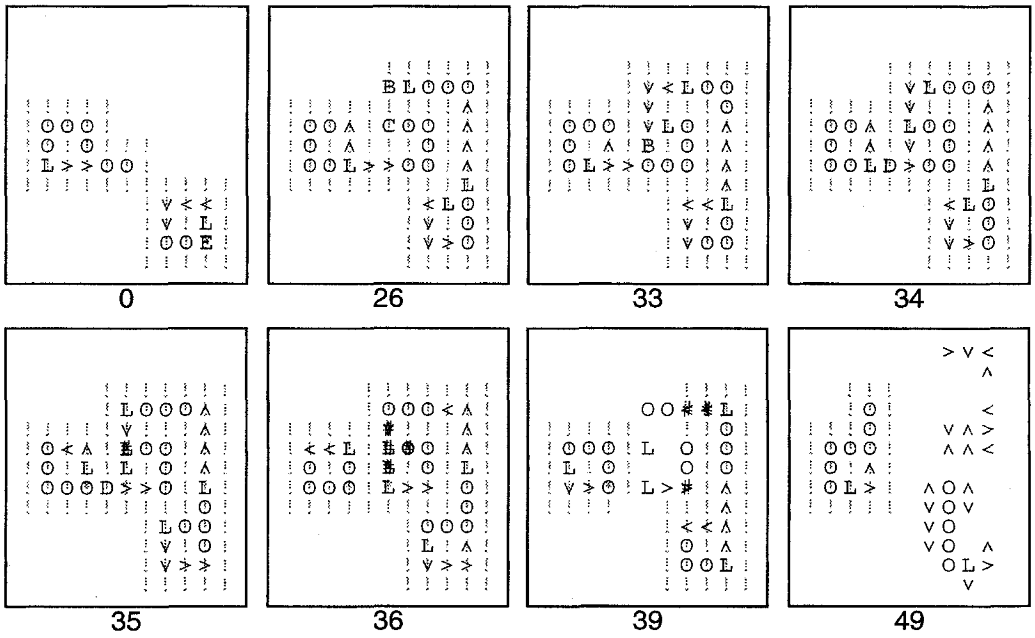


Fig. 7. Dissolving cellular automata structures. The bound bit, if set, is shown as a light gray exclamation mark '!' in the background. It denotes the part of the cellular automata space which "belongs" to a cellular automata structure. In epoch 0 we have two self-replicating loops both starting their replication cycles. By epoch 26 both have attained partial replication. In epoch 33 the smaller child loop has closed on itself, but the larger child loop is colliding with it. In epoch 34 the separation state D is formed at the connecting cell. The upper left L cell of the inner loop detects a failure situation and sets its failure mark '#' (light gray, epoch 35). In epoch 36 the D has disappeared and the leftmost original loop is secure, while the failure mark spreads in the larger loops. The spread of the failure mark '#' in a loop is fast (epoch 39), such that by epoch 49 all but the leftmost cellular automata structure has dissolved completely into unbound components. The original left loop will continue its replication in a new direction.

a structure will not run into another structure, that two structures will not try to replicate into the same region of the cellular automata space, or that a replicating loop will not run into free-floating unbound components. These factors are all "randomly" determined. Experiments (unpublished) with several past transition functions for self-replicating loops demonstrated that even minimal introduction of random cell state changes could completely disrupt the replication process (i.e., past transition functions have been *brittle*). In contrast, the transition function used here assumes that not all designated regular procedures will always be followed without interruption or disturbance from other structures. It includes rules that will detect any failed regular procedures and clean up the cellular automata space after such failure.

A failure situation happens when something prevents the regular replicating procedures from continuing, such as an obstacle in the extrusion path, or two loops colliding into each other. When such a situation occurs the failure value '#' appears in the special field. When a loop has any of its cells enter this failure mode, this mode quickly spreads throughout the whole structure, as shown in Fig. 7, causing the loop to *dissolve* completely. The loop's components become unbound and revert to being controlled by the rules governing unbound components. Cell state D blocks a failure mark from passing through it, thus protecting a failed child from its parent, and vice versa.

To illustrate some of the rules used during the failure detection process, recall the following rule for passing signal > through an O that was introduced earlier:

```
if (component =='O') rot if (we:component='>') component='>';
```

To check for failure, in the actual rule set additional conditions are present to detect when there is more than one > signal trying to move into the same O. If that is the case the fail mark should be set, as follows:

```
if (component=='O')
  rot if (we:component=='>')
     if (no:component!='>,1' && no:component!='>,2' &&
         ea:component!='>,2' && ea:component!='>,3' &&
         so:component!='>,3' && so:component!='>')
     component='>';
else special='#';
```

Here ! = means "not equal to" and && is a logical AND operation. Other failure checking rules are similar, i.e., they are all elaborated forms of normal rules for various regular functions.

Finally, a way is needed to set the growth bits that trigger extended replication. Recall that there is no a priori information about when and where these should be placed, and no growth bits are set initially. We adopted the following approach. Whenever a signal L dissolves, it leaves behind a growth bit at its location. A loop usually has only one L signal, so one dissolving loop usually produces one new growth bit in the cellular automata space. This way, the generation of the growth bit becomes part of the behavior of the cellular automata space, since when and where a loop will dissolve is determined purely by the interactions within the cellular automata space.

The growth bit is utilized during the arm branching phase of self-replicating loop to extend the signal sequence in a loop. As shown in Fig. 8, this is a two-step strategy. First, if a signal > sees a growth bit in its place and it

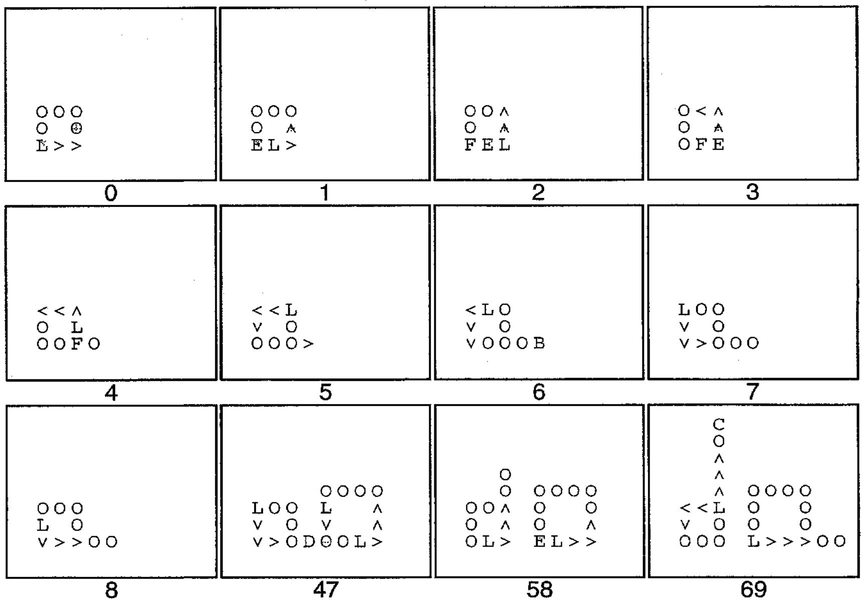


Fig. 8. The growth of a larger loop. In epoch 0 the branch special flag in the lower left cell and the growth bit in the middle right cell are both set. In epoch 2 the normal arm branching EF signal sequence is generated. In epoch 3 the signal sequence becomes >>> and subsequently the growth bit is unset. By epoch 8 the parent loop is about to start the replication cycle with one more > signal than it normally has. By epoch 47 a whole new loop bigger than the original one is generated. By epoch 58 the two loops have separated and the original one is just about to start another replication cycle. In epoch 69 the new, bigger loop is finished and is starting its own replication cycle.

is the last > before the signal L, it does **not** copy the signal L behind it as it normally does. Instead, it stays at its current value > for one more epoch, thus effectively increasing the size of the signal sequence by one. The signal L disappears temporarily since it is not copied, but reappears when the signal > sees a trailing signal F and the growth bit in its position. The growth bit is unset after signal L is regained, so the same growth bit does not cause another growth stimulus. Thus, when a loop dies, it leaves a set growth bit behind, and when a loop expands, it consumes a growth bit. This provides an interesting ecological balancing factor in the cellular automata universe.

3.5. *The minimum loop and emergence of replication*

What is the smallest possible loop capable of self-replication? The answer to this question is important because the probability of the random occurrence of a specific self-replicating structure arising from random occurrences of individual unbound components would in general decrease as its size increases. Our previous research in minimizing the size of self-replicating loops led to self-replicating structures with only five components [14]. However, the rules described here so far can only support the self-replication of 3×3 loops and larger because the signal sequence rules require that two signal sequences not be adjacent to each other. In addition, a critical limiting factor in the number of available signal carrying cells in a loop. For the arm extrusion sequence (EF) to work we need on some occasions two more cells in addition to those for the normal replication sequence. At least one extra loop cell is also needed if we want to allow extended replication where loops grow in size. For a 3×3 cell structure the total number of required cells is 6. For a 2×2 cell structure that number becomes 5, a seemingly impossible case since it only has four cells. Nevertheless, with the addition of rules that specifically address these problems, it is possible to support replication of 2×2 loops, as illustrated in Fig. 9. With such a small replicating loop, it is straightforward to provide for the emergence of self-replication from an initially random set of unbounded components. This is achieved by allowing the unbound components to translate and change or appear at "random", i.e., by "stirring the primordial soup". The *unbound component rules* that do this can be summarized by:
- If a quiescent cell has exactly three active neighbors, it will itself become active in the next epoch. Its active value will be determined quasi-randomly based on the state of its neighbors. [3]
- If an active cell has exactly two or three active neighbors, it will stay active; otherwise, an active cell will return to the quiescent state in the next epoch.

These rules, of course, are generalizations of those used in the Game Of Life [7] from binary to non-binary states. Fig. 10 shows a simple example of how these rules work. They generally produce a continually varying distribution of unbound components.

All that is then required for the emergence of self-replication is a small set of rules that watch for the "random" formation of the smallest loop configuration (the 2×2 loop). Once such a configuration occurs, all four members of it simultaneously set their own bound bit and produce an active smallest loop in the next epoch. This is how the first self-replicant is formed. This is possible using only local operations because the minimum loop configuration is so small that it is within a 3×3 Moore neighborhood, so each component can simultaneously "see" the same configuration. An example of how the final unbound component rule set works and how it leads to the first self-replicating structure is demonstrated in Fig. 11.

[3] Since the transition function is deterministic, the identity of a newborn component is determined by adding the integer values of its neighboring cell states mod 6. This selects one of the six states >, ∨, <, ∧, L or O. Only the initial state of the cellular space (same six components; see Fig. 1) is based on a pseudo-random number generator.

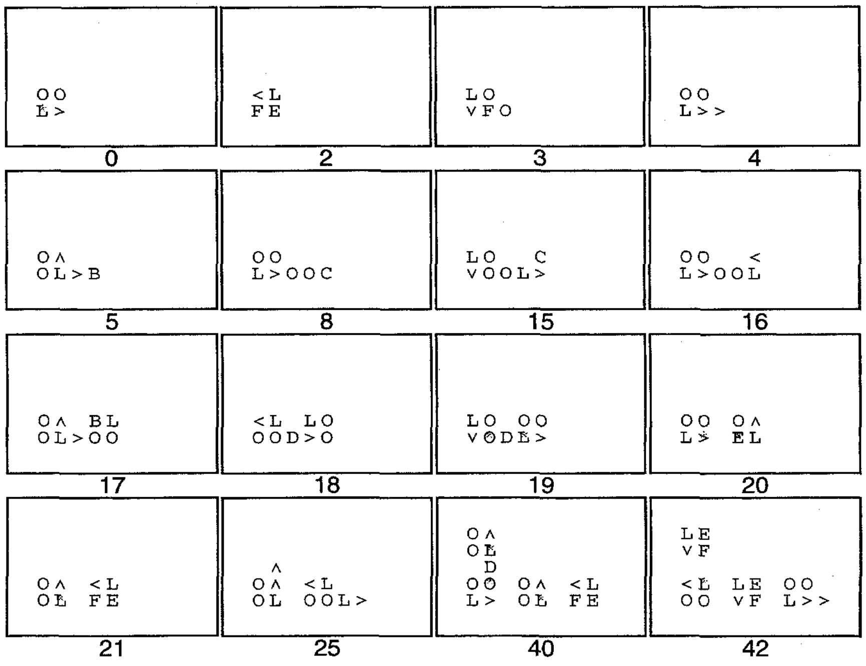


Fig. 9. The smallest self-replicating loop shown in isolation. This structure starts as only a 2 × 2 loop with its lower left cell's special field set to '∗'. By epoch 5 the arm appeared. By epoch 17 the new loop is closing on itself and in epoch 18 the separator D is formed as usual. In epoch 19 the two branching special values '∗' are set for both loops. Note that although the child (right) loop has not detached from the parent (left) loop here, it has precisely replicated the parent loop (epoch 0). In epoch 20 the D disappears, completely separating the parent and child loop. By epoch 42 there are four loops, and all are still actively replicating.

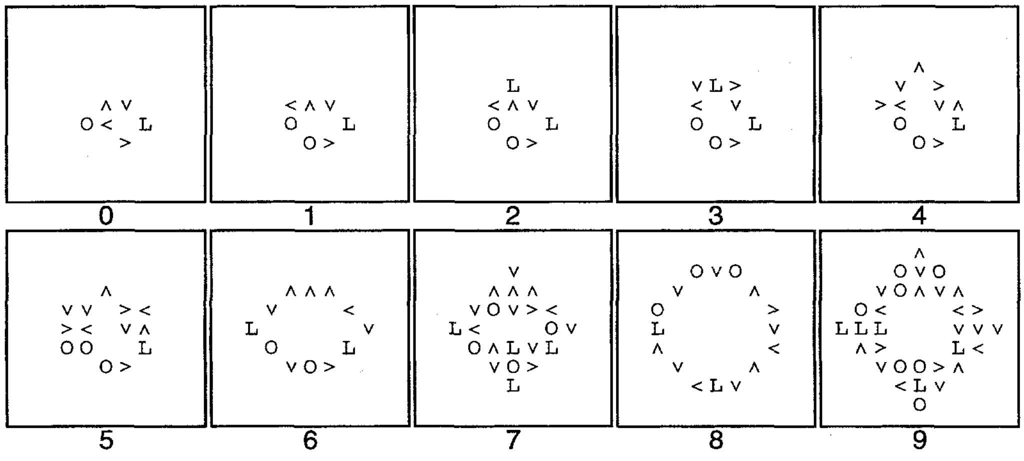


Fig. 10. The effects of unbound component rules. In epoch 0 a few random active states are introduced into the cellular automata space. In epoch 1 the middle '<' component dies because it has more than three active neighbours (Moore neighborhood). In the same epoch two new components, one 'O' and one '<' are born since there were exactly three active neighbors at their locations. Note that the state value 'O' and one '<' are determined by state values of the active neighbors (see text). In epoch 2 one new 'L' component is born. This process continues indefinitely.

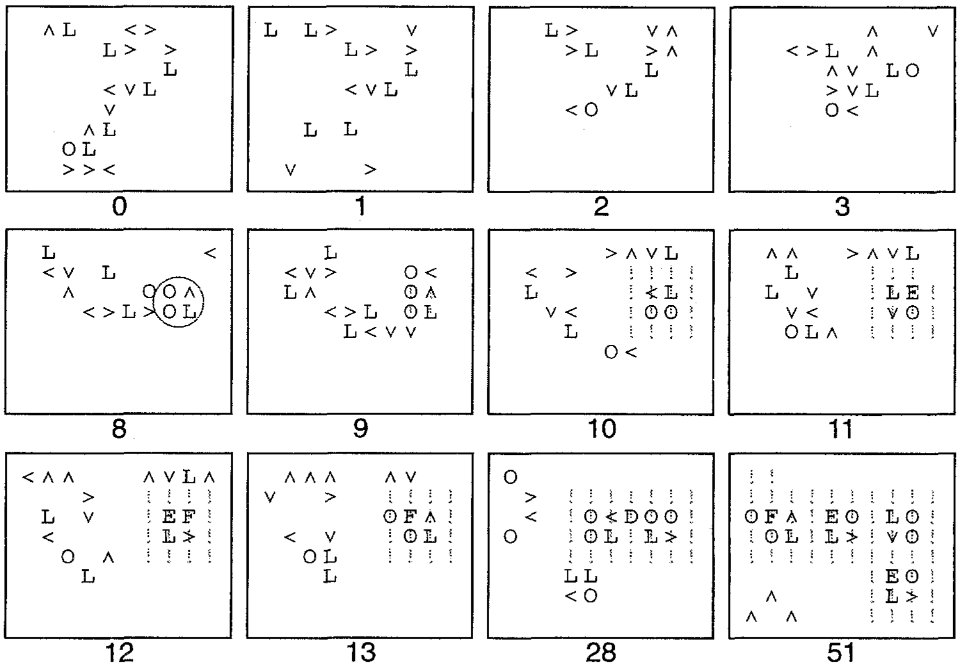


Fig. 11. The emergence of a self-replicating structure. Components of structures are marked by a non-zero bound bit, or an '!' mark. In epoch 0 a randomly generated initial space is given. This space has only unbound components until epoch 8, when the pattern of the smallest replicating loop (circled) appears. In epoch 9 this configuration turns into a functioning self-replicating loop when its four cells set their bound bit simultaneously. Its peripheral cells clear and the arm branching process begins (epochs 10–13). By epoch 28 the first sibling is about to separate. By epoch 51 four loops are obtained and all are actively engaging in the replication processes.

4. Experiments and analysis

The rule set that we designed is intended to produce self-replicating loops when starting with an initial space having randomly placed unbound components. However, it is not obvious that this rule set will work frequently or quickly, how often self-replicants will arise, whether they will persist, and how model parameters will affect the occurence of self-replication. In this section we describe systematic experiments that examined these issues, including how the space size and initial component density affected model behavior. We also consider the long-term behavior of the model.

4.1. Methods

A series of simulations were conducted while varying the cellular automata space size (50×50, 100×100, 150×150 and 200×200), initial unbound component density (10%, 20%, 30%, 40% and 50%) and random initial configuration. All simulations were run for 30 000 epochs unless the cellular automata space ceased all activity before then.

Since it was not feasible to collect the simulation data manually, a data collecting module was built into the simulator to analyze the cellular automata space configuration while the simulation was running. It counts the number of cells of certain types and also recognizes some higher-level structures. Replicating loops were automatically identified, and their sizes were recorded. Data collected for each simulation epoch contained the number of active cells, the number of active bound cells, the number of growth bits, and the number and size of individual loops in the cellular automata space. These data were later batch processed by another program to determine the average size of loops for each epoch.

4.2. Basic results

Eighty one simulations were done while systematically varying the space size (50 × 50 to 200 × 200) and initial density of unbound components (10% to 50%). A new, randomly assigned initial state was used with every simulation. It was found to be harder to support long term evolution of emergent self-replicating loops in the smallest

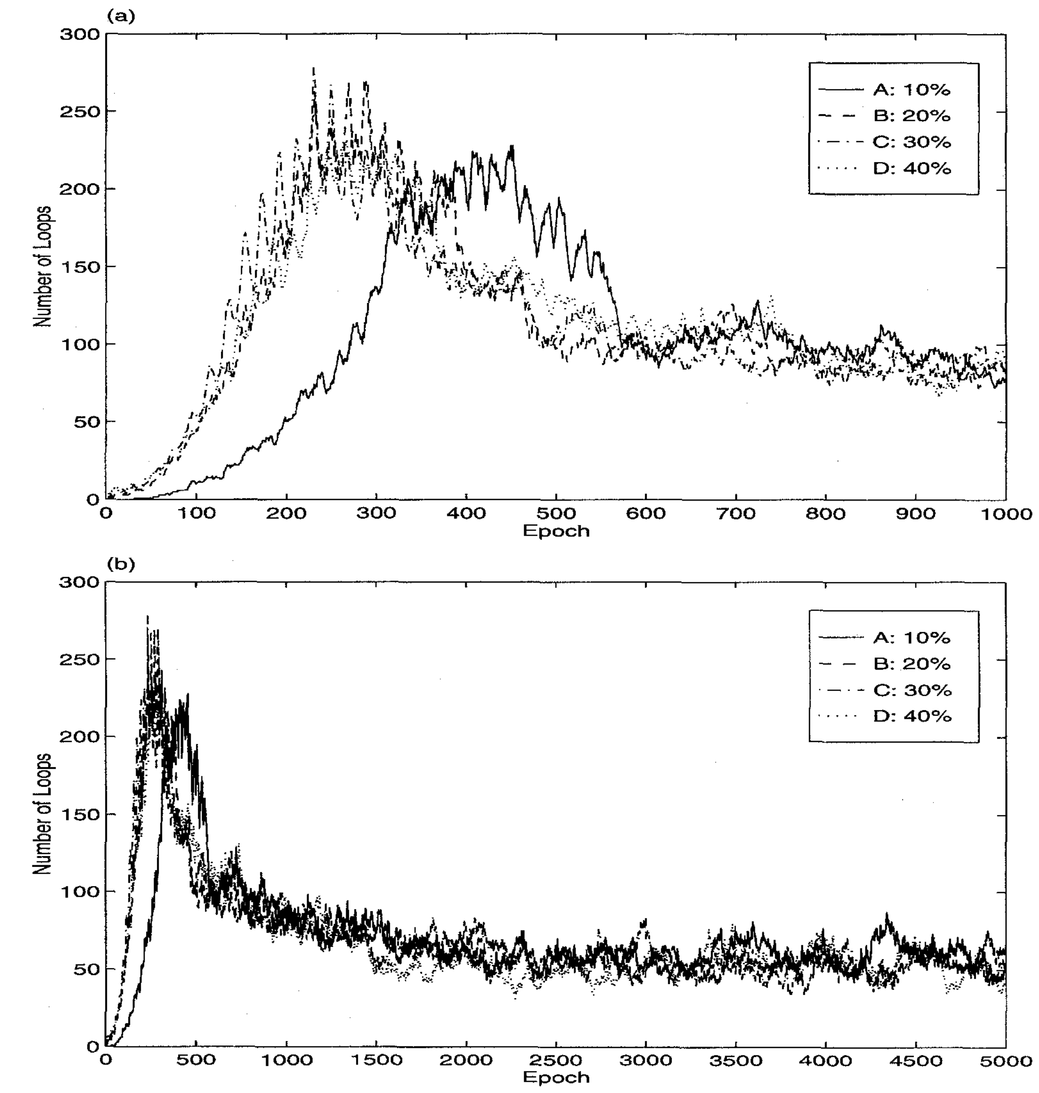


Fig. 12. Comparison of simulations having different initial densities of unbound components. In this example simulations for a cellular automata space of 100 × 100 cells are compared using four different initial densities (10%, 20%, 30% and 40%). The number of loops present is shown during the first 1000 epochs (a) and during the first 5000 epochs (b).

cellular space size examined (50×50). Only one-third of the simulations for the 50×50 world size reached epoch 30 000, the predetermined termination point. The other two-thirds ceased all activity before reaching the termination epoch 30 000 (usually between 15 000 and 28 000 epochs). Despite the shorter life span of simulations for this space size, they all still generated self-replicating loops except for one unusual simulation with an initial low (10%) unbound component density, where all activity ceased by epoch 95. Since only one out of 81 simulations done failed to generate self-replicating loops, apparently it is easy to generate self-replicating loops with the rule set described in this paper.

Simulation behavior was not much influenced by the initial random configurations. All simulations sharing the same parameters (space size and initial unbound component density) exhibited the same characteristic behavior. For example, the total number of active cells and number of growth bits set were qualitatively and almost quantitatively the same over time, and with both of these measures the quantities asymptotically approached near-constant values.

More surprisingly, the initial unbound component density also did not significantly change the simulation results, at least over the range of values examined. For example, Fig. 12 shows the number of self-replicating loops in the cellular automata space during four simulations having different initial unbound component densities for a cellular automata space size of 100×100. Although these curves are somewhat different for the four simulations, especially initially, the initial density of unbound components did not make any significant difference beyond epoch 1000. This similarity occurred with all other cellular automata space sizes and for all other data collected. Clearly the emergence, proliferation and persistence of self-replicating loops is a robust phenomenon relatively insensitive to the initial conditions of a simulation.

Simulations were also run with different cellular automata space sizes. If all data are normalized by dividing them by a measure of relative size of the cellular automata space, the resulting values are again mostly similar to each other, except for the average loop sizes. To do this normalization, we use a *size ratio* of 1 for a 50×50 space, 4 for a 100×100 space, 9 for a 150×150 space, etc. With such normalization, the number of active cells per unit area is found to be roughly the same and thus largely linearly scalable and independent of the cellular automata space size. Plots of active cells over time for smaller cellular automata spaces tend to have greater variances as would be expected, but the average tendency is still the same. Fig. 13 shows the number of replicating loops in the cellular automata space, starting with an initial unbound component density of 40%, for different space sizes. The curves for smaller cellular automata spaces tend to have greater variances, but the general trends are the same. This and other similar comparisons show that the number of active cells, the number of bound cells, the number of growth bits and the number of loops per unit area are essentially all independent of the cellular automata space size. In contrast, the normalized average loop size is not the same for different space size simulations. In Fig. 14, for example, both the normalized and unnormalized average loop size curves are shown for four simulations having the same initial unbound component density of 30% but different space sizes. It can be seen that although a larger cellular automata space tends to allow larger loops, the increase is not linear.

In summary, the following properties are observed:

- Different simulations having the same cellular automata space size and initial unbound component density but different random initial configurations behave the same.
- Different simulations having the same cellular automata space size but different initial unbound component densities behave the same.
- Different simulations having different cellular automata space sizes and initial unbound component densities behave the same except the average loop size.

Thus there is a very stable and characteristic dynamics under the emergent self-replicating rule set. Because of these observations, in the following we just describe the data for 200×200 cellular automata space simulations as examples, with the understanding that the other different space sizes behave fairly similarly.

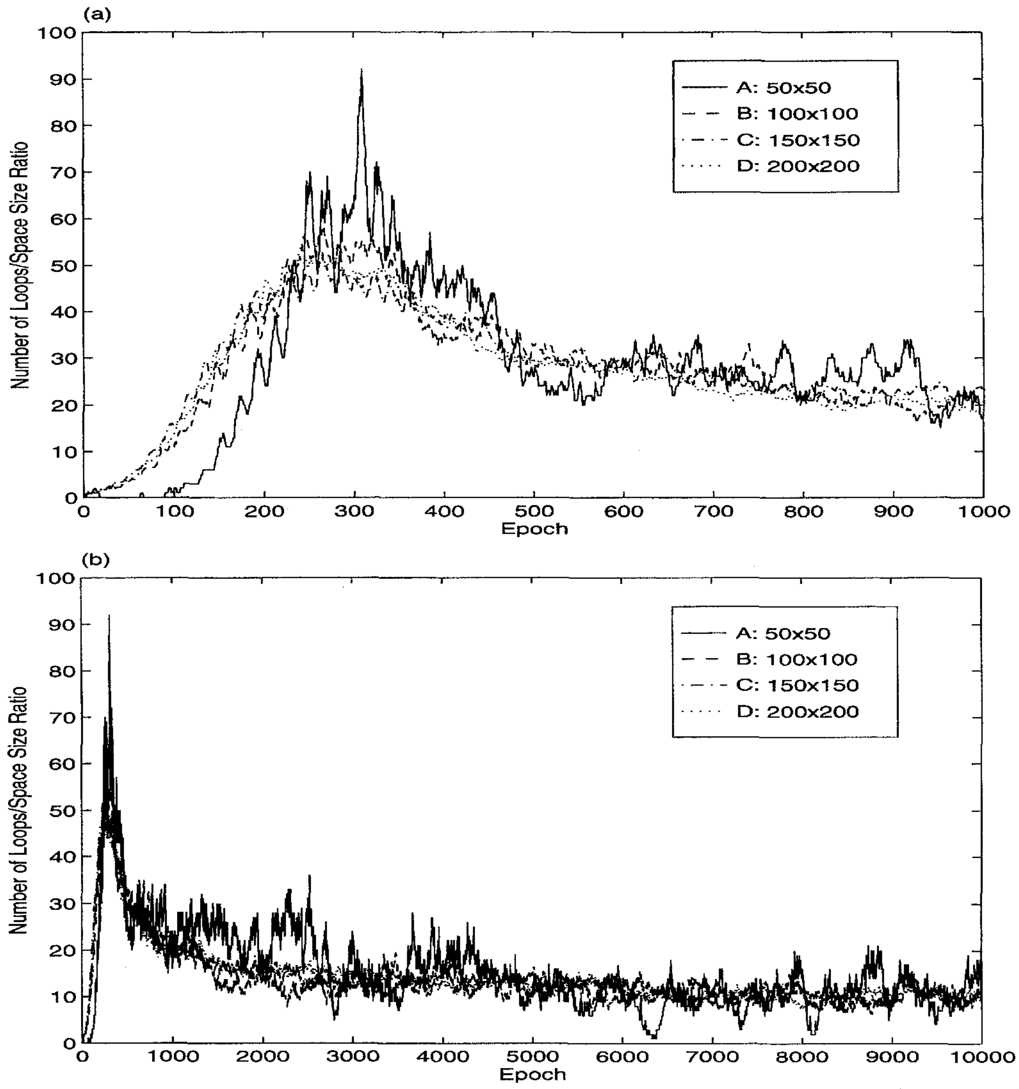
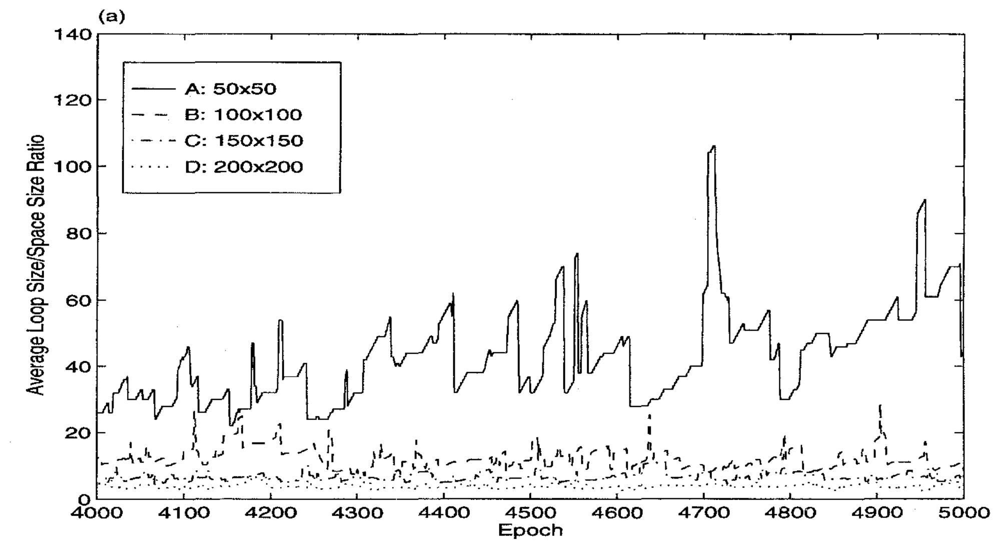


Fig. 13. Comparison of number of loops for different cellular automata space sizes. The number of loops in the cellular automata space is scaled by the space size ratio for the four simulations.

4.3. Further analysis

Fig. 15 shows the number of active cells and the number of bound cells for a simulation with a 200 × 200 cellular automata space and an initial unbound density of 30%. These two curves have an almost constant difference between their values, in this case an average of 2480 cells, representing the number of unbound components. The same average differences for three similar 200 × 200 cell simulations with an initial unbound component density of 10%, 20% and 40% are 2475, 2477 and 2473. This value appears to be very stable over time.

To determine if this constant population of unbound components is the natural tendency of the unbound component rules in isolation, or if it is the joint property of these rules and those governing self-replication, simulations were



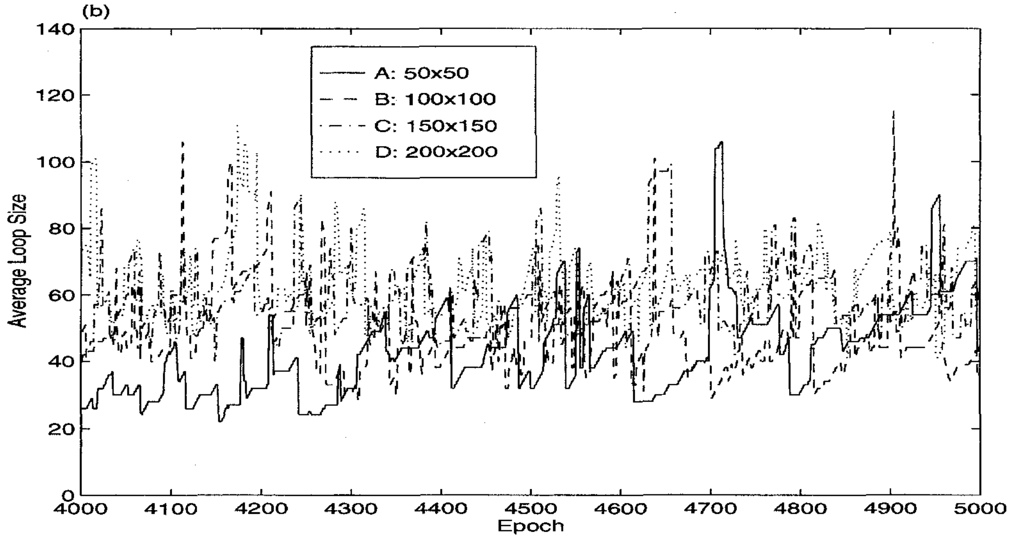


Fig. 14. Comparison of average loop sizes for different cellular automata space sizes. In this figure, the average loop sizes for each simulation are compared: (a) normalized by space size ratio, (b) not normalized. The relative vertical positions of curves are reversed in these two graphs, i.e., the curve for a 200 $\times$ 200 space is the lowest in (a) but the highest in (b).

conducted where bound cells were not allowed to be generated. None of these simulations maintained a constant unbound component population as high as that of 2480 in a 200 $\times$ 200 cellular automata space. Actually, all of them ceased activity long before reaching 30 000 epochs. Thus, the relatively constant level of unbound components depended on the entire rule set.

The behavior of active cells can be seen most clearly in Fig. 15(b). At the beginning, there are 12 000 active cells, or 30% of the total cells in the space. This number drops rapidly since the unbound component rules alone are not capable of sustaining such a vast number of active cells. If no bound components are formed, this number will continue to fall. However, around epoch 100 bound components appear with the emergence of self-replication. There is an initial surge of bound components due to the initial proliferation of small and rapidly replicating loops.

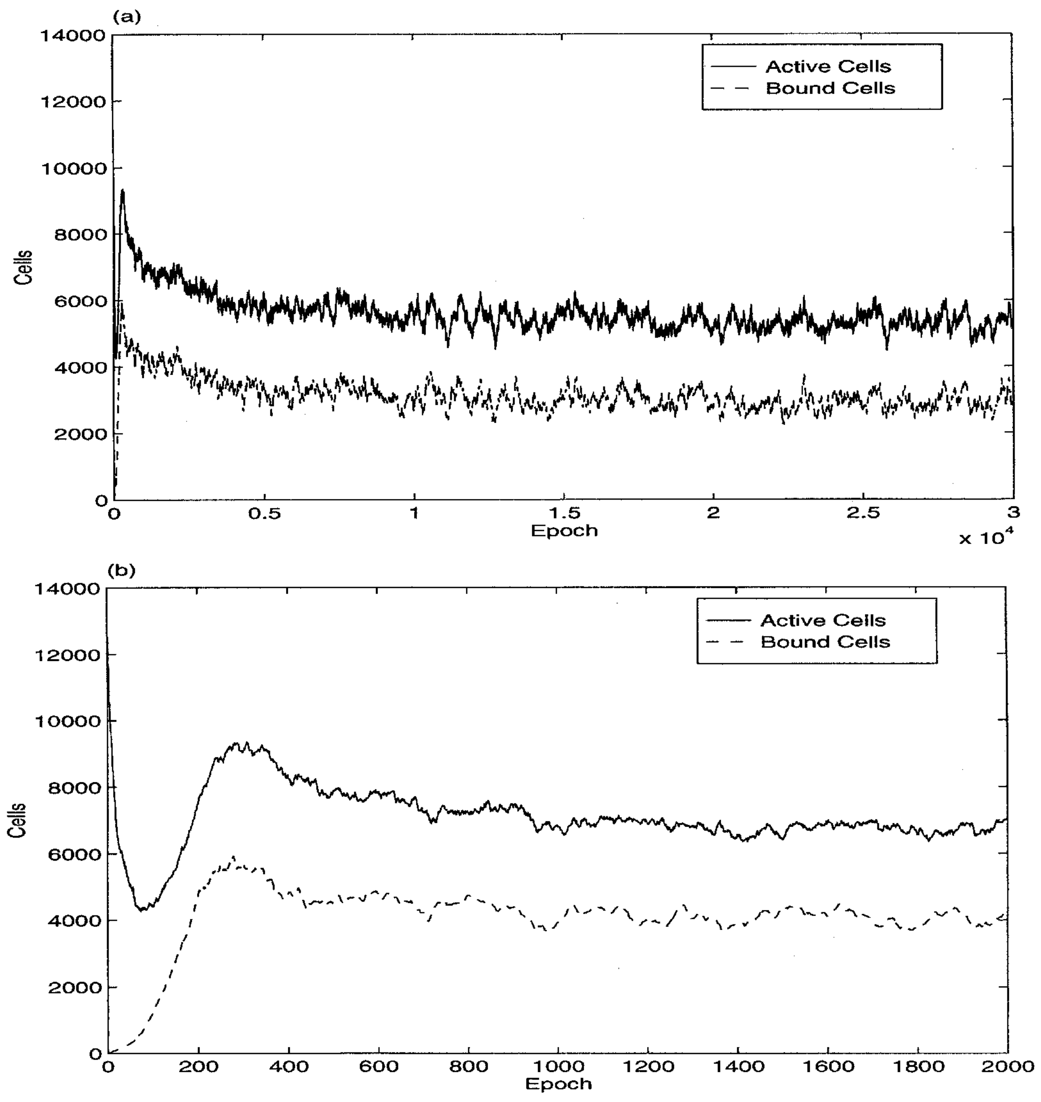


Fig. 15. The relationship between active cells and bound cells for a simulation with 200 × 200 space size and 30% initial unbound component density. Graph (b) is an expanded view of the first 2000 epochs of graph (a).

Gradually, a balance is achieved and the number of unbound and bound components approximate constant values asymptotically.

The change in the number of growth bits is shown in Fig. 16(a) for a 200 × 200 cell simulation with 30% initial unbound component density. The number of growth bits climbs rapidly at the beginning, but gradually tends to level off as consumption and generation of growth bits becone balanced. The final growth bit density occurs at 56% in this case.

The number of replicating loops generally stabilizes too. In Fig. 16(b) the change in the number of loops with time is shown for the same simulation. After 10 000 epochs, there is no significant change of the average number of loops (147) in the cellular automata space. In the beginning of the simulation, there is a transient overshoot of this

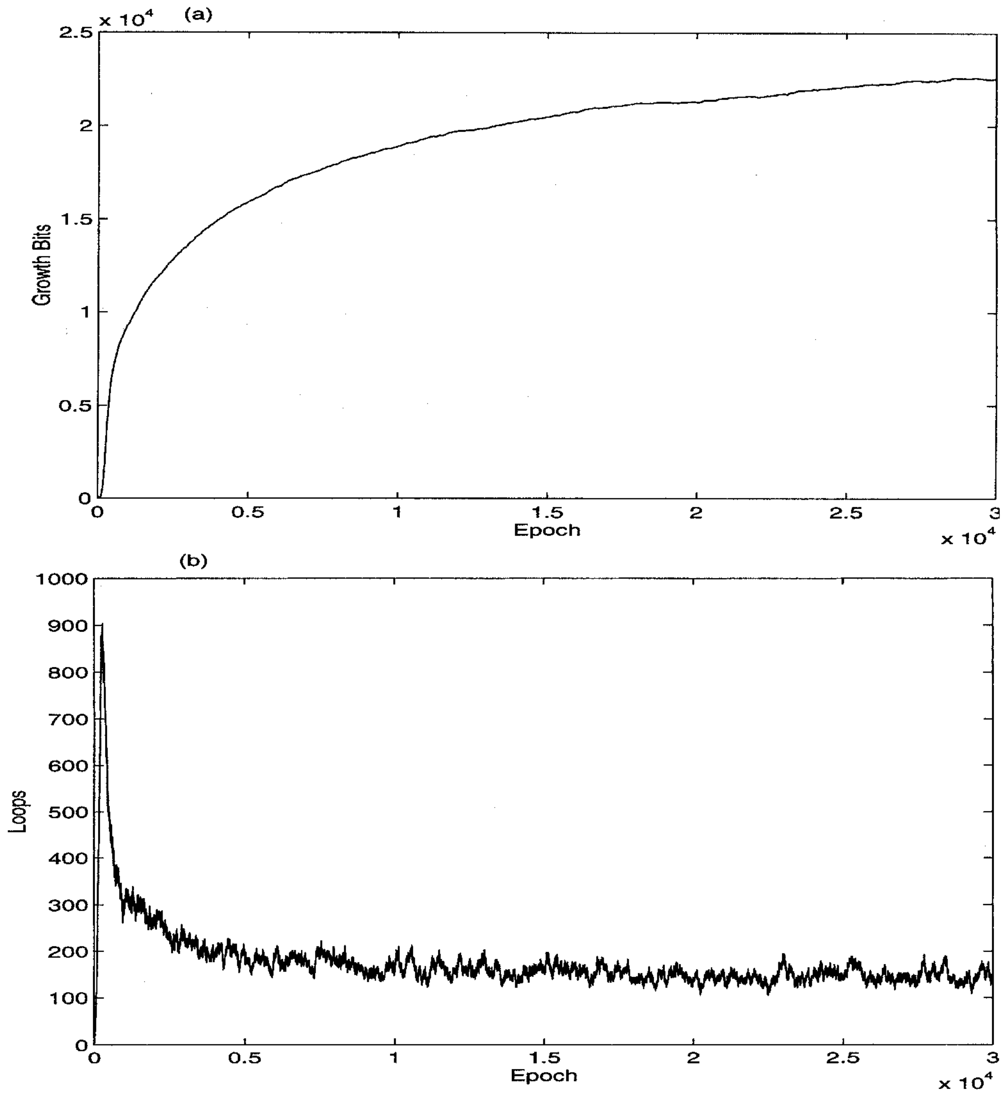


Fig. 16. Change in number of growth bits (a) and loops (b) with time (200 × 200 cellular automata space size, 30% initial unbound component density).

long term average number of loops. This is for the same reason that the number of bound components overshoots, i.e., smaller loops are proliferating rapidly at the beginning but slow down when competitive pressure raises.

The change in average loop size is shown in Fig. 17(a) for the same 200 × 200 cell simulation with a 30% initial unbound component density. A detailed portion of this is shown in Fig. 17(b) where it is seen that, the average loop size is almost cycling at times. This behavior is in exact accordance with what one observes visually in the cellular automata space during a simulation. Recall the example of Fig. 1 where loops in the cellular space showed the tendency to grow bigger and bigger (epochs 500–5000), until there is no more space to grow any larger, and then the bigger loops disappeared, replaced by smaller loops again (epoch 7500). It is this quasi-cycling behavior which produces the zigzag shape of the curve and may reflect an underlying chaotic dynamics.

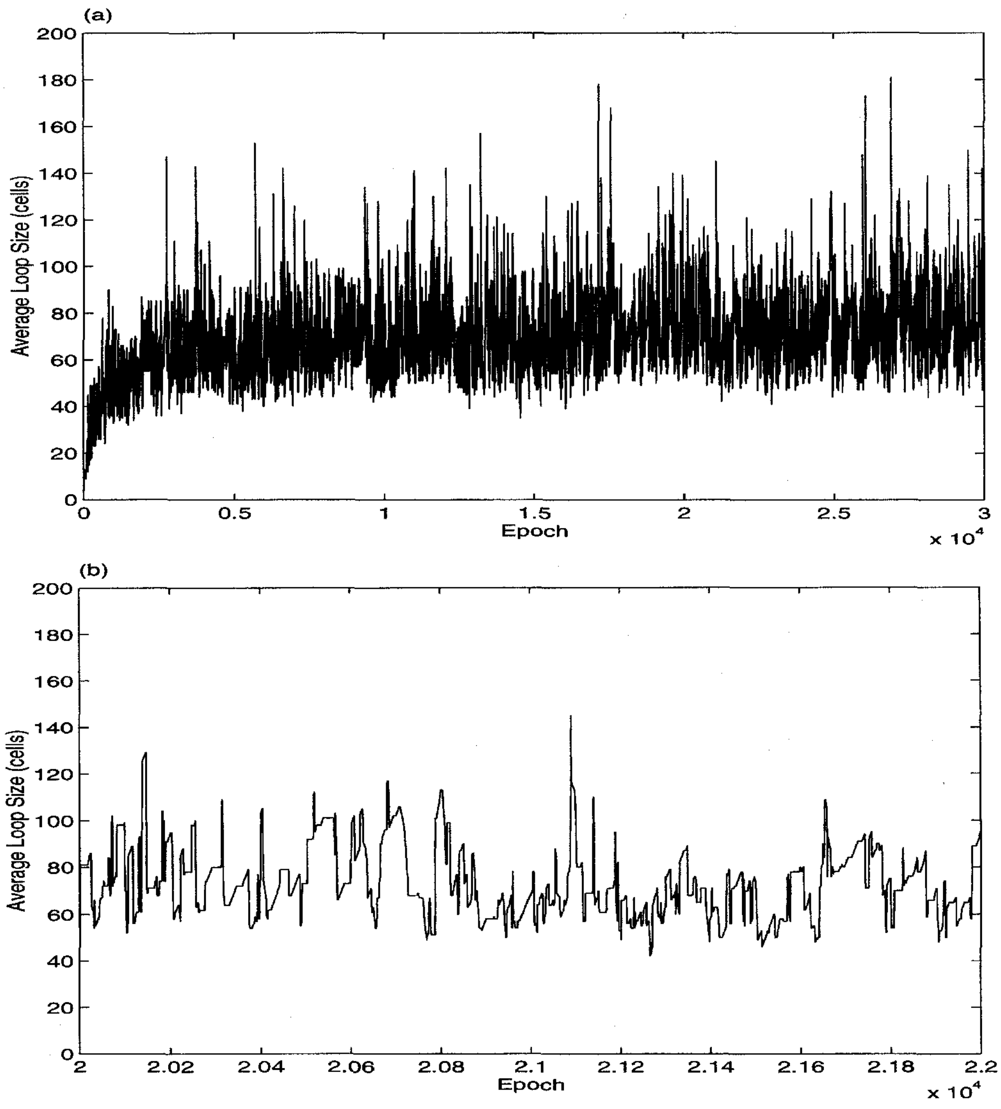


Fig. 17. The change in average loop size with time. Graph (a) is for the whole course of the simulation, graph (b) for epochs 20 000–22 000.

The average loop size after the cellular automata space has reached a stable condition is about 73 cells for a 200×200 cellular automata space size. As shown earlier, the average loop size is not linearly scalable with the cellular automata space size. The average loop size for a 150×150 cell space is 64 cells, for a 100×100 space it is about 56 cells, and for a 50×50 space it is about 39 cells. Two term curve-fitting with the available simulation data reveals a close fit of the curve $\mathit{loopsize} = -56.53 + 24.38 \log(\mathit{space\ size})$ to the data. The fitted curve and the simulation data are compared in Fig. 18.[4]

[4] This curve is solely intended to suggest a logarithmic relation for the existing data and not for extrapolation beyond the limited range of sapce sizes examined.

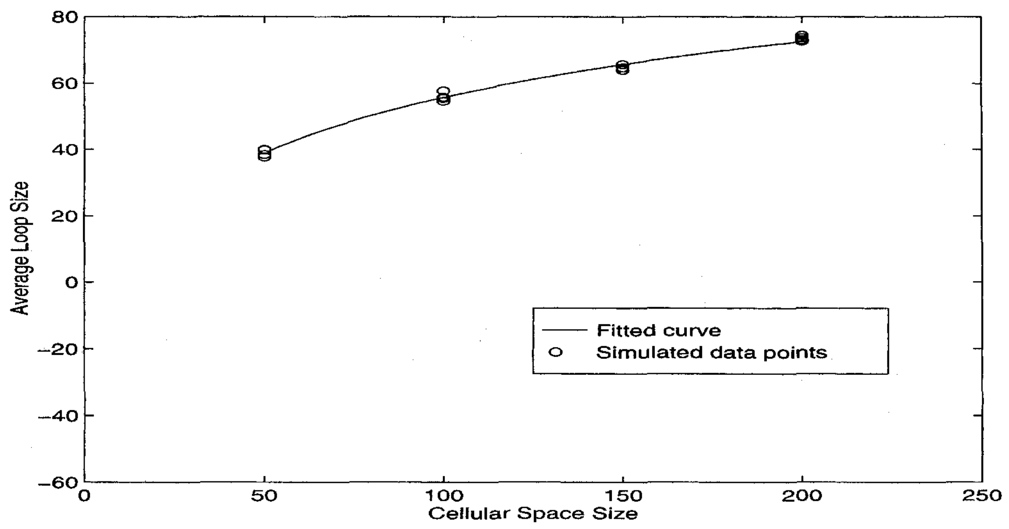


Fig. 18. The long term average loop size for each simulation is indicated by 'o' marks. The fitted curve is shown by a line.

Finally an extremely long simulation was done with a 200 × 200 space and an initial unbound component density of 30%. This simulation was allowed to run for 613 920 epochs, about 20 times longer than the above simulations. The results showed no sign of changing once a quasi-stable status had been reached. An important issue is whether the cellular automata space might be going through some sort of cycle, i.e., whether the configuration of the cellular automata space is repeating itself. To study this, the positions and numbers of self-replicating loops of the long term simulation for each epoch were cross-compared with each other. No duplication was found during the entire 613 920 epochs.

5. Discussion

The results reported here show for the first time that non-trivial self-replicating structures can emerge in a cellular automata space initialized with a randomly distributed set of components. Unlike past cellular automata models, the initial states in our simulations did not contain any replicating structures, and were different in each simulation. The emergence of replicating loops was quite robust, occurring in 80 of 81 simulations having different space sizes, densities of components, and initial configurations of these components.

Once small self-replicating loops appeared, they gradually increased in size, eventually reaching an average size that was characteristic of the size of the cellular space. However, the number and size of self-replicating loops never reached an equilibrium. Instead, these values oscillated around an average value. The oscillations were of varying amplitude and non-periodic, suggesting that the behavior of the model has a chaotic dynamics.

To our knowledge, only one previous computational study of emergent self-replication has been done [12]. That investigation used a very different (non-cellular automata) model having an initial state composed of randomly generated sequences of computer operations. It evolved self-replication via a mutation operation. The primary conclusion, backed up by simulation results, was that the probability of a randomly generated sequence of operations becoming self-replicating increased with the number of computer operations it contained. Further, self-replicating sequences decreased in size once they appeared. Our cellular automata model provides a "proof by existence" that such behaviors are not necessarily an inherent aspect of emergent self-replication, in that very small self-replicants can arise first and then increase in size, as is often argued to have occurred with the origins of biological replication. We attribute the differences in our results to the fact that we started with random individual components rather than random initial *sequences* of computer operations, that our rules were hand crafted, and that cellular automata are based solely on highly local operations (e.g., there is no global copy operation that copies a loop to a nearby region of the space).

The results of this study and previous work are encouraging in that they suggest that computational models of self-replicating structures may be used effectively to explore hypotheses related to the origins of life. In addition, recent demonstrations that replicating loops in cellular automata spaces can be programmed to solve problems as well as to replicate [4,13,16] raises the possibility that these abstract models of replication may ultimately lead to practical applications. Two other future avenues of investigation are also suggested by this work. First, there is a clear need for theoretical analysis and explanation of the long term behavior of the self-replication process modeled here. Second, development of more realistic models of replication that can be more directly related to experimentally studied replication of oligonucleotides [3] would allow a tighter coupling of theoretical/computational work and experimental work.

Acknowledgements

Supported in part by NASA award NAGW-2805.

Appendix A. Rule listing

The following rules differ somewhat from those in the text because, for efficiency, more *else-if* statements are used rather than plain *if* statements, and they are organized according to the component types they deal with rather than the functions of the rules. By default, if no rule matches a cell's current state, its state is unchanged at the next epoch. A fully commented version of these rules is available online at the website listed earlier.

```
// *********  Default rules are used when no other rule apply **********
default component=component; default growth=growth; default bound=bound;
default rot if (component && special=='.' &&
               (ea:component=='D' && no:component && we:component ||
                we:component=='D' && no:component && ea:component))
            special='*';
        else special=special;

// ***************   Variable and Function Declarations    ****************

int count,      // number of neighbors meeting a certain condition
    value;      // accumulates component values from neighbors
nbr y;          // a variable used for looping through all neighbors
int Error()     // detection of anomalous conditions (e.g., loop collision)
  {count=0;
    over each other y: if (y:component) count++;
    if (count && count<=5) return 0; else return 1;}
```

```
// *****************   Unbound component rules    ********************
if (bound==0) {
   count=0; value=0;
   over each other y:
      { if (y:component) { count++; value = value + y:component; }
        if (y:bound && y:component && y:special!='#') {count = 99; break; }}
   if (count==99) { component='.'; bound='!'; }
   else if (special) special=0;
   else rot if (component=='>' && we:component=='L' && nw:component=='0' &&
                no:component=='0') { bound='!'; special='*'; }
   else rot if (component=='L' && ea:component=='>' && no:component=='0' &&
                ne:component=='0') bound=1;
   else rot if (component=='0' &&
                (so:component=='L' && se:component=='>' && ea:component=='0' ||
                 so:component=='>' && sw:component=='L' && we:component=='0'))
        bound=1;
   else if (count<2 || count>3) component='.';
   else if (component=='.' && count==3) component = (value+1)%6+1;
} // closure of the "if (bound=0) { ..." statement

// *************  Bound component rules (loop components)  *************
else {
  if (special=='#') bound=0;
  else rot if (component && component! ='D' &&
              (no:special=='#' || ne:special=='#'))
        { special='#';  if (component=='L') growth='+'; }
  else if (component && Error()) special='#';
  else {
    if (component=='0') {     // **************** Block 0 rules
      if (special!='-') {
        rot if (no:component=='B' && nw:component=='.' &&
               (we:component=='0' || ea:component=='L')) special='-';
        else rot if (we:component=='>')
          if (no:component!='>,1' && no:component!='>,2' &&
              ea:component!='>,2' && ea:component !='>,3' &&
              so:component!='>,3' && so:component!='>') component='>';
          else special='#';
        else rot if (so:component=='>')
          if (we:component!='>',1' && we:component !='>' &&
              no:component!='>',2' && no:component !='>,1' &&
              ea:component!='>, 3' && ea:component!='>,2') component='>,3';
          else special ='#';
        else rotif (so:component=='F' && se:component=='.' &&
                    no:component=='.' && ea:component=='.' &&
                    we:component=='.') component='>,3';
        else rot if ((nw:component=='>,3' || nw:component=='0') &&
                     (ne:component=='B' || ne:component=='>,1') &&
                     no:component=='.' && ea:component && we:component )
                  component='D';
  } else rot if (we:component=='L') special='*'; }

else rot if (component=='>' {   // **************** Signal > rules
  if ((nw:component=='>,3'|| nw:component=='0') &&
      (ne:component=='B' || ne:component=='>,1') &&
     no:component=='.' && ea:component && we:component) component='D';
  else if (we:component=='L')
    if (nw:component=='E' && (growth || no:component=='>,3')) growth='+';
    else component='L';
  else if (no:component=='L') component='L';
  else if (we:component=='>') component='>';
  else if (we:component=='>,1') component='>';
  else if (no:component=='>,1') component='>';
  else if (growth && nw:component=='F') { component='L'; growth=0; }
  else component='0';          }

else if (component=='B') {   // **************** Birth B rules
  rot if (we:component=='L' && no:component=='0') component='L';
  else rot if (we:component=='L') component='C';
  else rot if (we:component=='>') component='>';
  else rot if (no:component && so:component) special='#';
  else component='0';          }

else if (component=='L') {   // **************** Signal L rules
  if (special=='*') component='E';
  else rot if (we:component=='>') special='#';
  else rot if (so:component=='E' && sw:component!='F' &&
              (no:component=='.' || we:component=='.')) component='E';
  else component='0';          }

else if (component=='.') {   // **************** Quiescent state . rules
  rot if (we:bound && we:component=='>' && (nw:component=='.' ||
           nw:component=='>,1' || nw:component=='L')) component='B';
  else rot if (so:special==0 && so:component=='E' &&
              se:component=='.' && no:component=='.') component='0';
else {count=0;
      over each other y:
        if (y:bound && y: component && y:special==0) count++;
      if (count==0) bound=0; }   }

else if (component=='C') {  // **************** Corner C rules
```

```
  rot if (no:component=='E' || no:component=='F') special='#';
  else rot if (we:component=='>' || we:component=='>,1') component='>,3'; }

else if (component=='D')   // **************** Detach D rules
  { rot if (ea:special) component='.'; }

else if (component=='E')   // **************** Signal E rules
  { component='F'; special='.'; }

else if (component=='F') {   // **************** Signal F rules
  rot if (no:component=='>,2' && ea:component)
    if (ea:component=='E' || so:component=='O') component='>,1';
    else special='#';
  else rot if (no:component ==' O' &&
             (ea:component=='O' || ea:component=='L') &&
             so:component=='.' && we:component=='.') special='#';
  else component='O'; } } }
```

References

[1] A. Burks, Von Neumann's self-reproducing automata, in: Essays on Cellular Automata, ed. A. Burks (University of Illinois Press, Urbana, IL, 1970) pp. 3–64.
[2] J. Byl, Self-reproduction in small cellular automata, Physica D 34 (1989) 295–299.
[3] H. Chou, J. Reggia, R. Navarro-Gonzalez and J. Wu, An extended cellular space method for simulating autocatalytic oligonucleotides, Comput. Chem. 18 (1994) 33–43.
[4] H. Chou and J. Reggia, Solving SAT problems with self-replicating loops (1997), submitted.
[5] E.F. Codd, Cellular Automata (Academic Press, New York, 1968).
[6] K. Drexler, Biological and nanomechanical systems, in: Artificial Life, ed. C. Langton (Addison-Wesley, New York, 1989) pp. 501–519.
[7] M. Gardner, The fantastic combinations of John Conway's new solitaire game "life", Scientific American 223 (4) (1970) 120–123.
[8] C. Langton, Self-reproduction in cellular automata, Physica D 10 (1984) 135–144.
[9] C. Langton, Life at the edge of chaos, in: Artificial Life II, eds. C. Langton, C. Taylor, J. Farmer and S. Rasmussen (Addison-Wesley, Reading, MA, 1991) pp. 41–91.
[10] K. Morita and K. Imai, Self-reproduction in a reversible cellular space, Proc. Int. Workshop on Universal Machines and Computations (Paris, 1995) pp. 29–21.
[11] L. Orgel, Molecular replication, Nature 358 (1992) 203–209.
[12] A. Pargellis, The evolution of self-replicating computer organisms, Physica D 98 (1996) 111–127.
[13] J. Perrier, M. Sipper and J. Zahnd, Toward a viable, self-reproducing universal computer, Physica D 97 (1996) 335–352.
[14] J. Reggia, S. Armentrout, H. Chou and Y. Peng, Simple systems that exhibit self-directed replication, Science 259 (1993) 1282–1288.
[15] J. Reggia, H. Chou and J. Lohn, Artificial self-replicating systems, Adv. in Comput. (1998), under review.
[16] G. Tempesti, A new self-reproducing cellular automaton capable of construction and computation, in: Proc. 3rd European Conf. Artificial Life, eds. F. Moran et al. (Springer, Berlin, 1995) pp. 555–563.
[17] T. Toffoli and N. Margolus, Cellular Automata Machines (MIT Press, Cambridge, MA, 1987).
[18] J. von Neumann, The Theory of Self-Reproducing Automata (University of Illinois Press, Urbana, IL, 1966).