EMERGENT COMPUTATION USING A NEW MODEL OF CELLULAR AUTOMATA

YASSER HASSAN and EIICHIRO TAZAKI
Department of Control and System Engineering
Toin University of Yokohama
Yokohama, Japan

In recent years, an approach-termed emergence system has gained popularity in a variety of fields, however, emergent behavior in decentralized spatially extended systems, such as in Cellular Automata, is still not well understood. The difficulties we face in adopting a definition of the concept of emergence are reminiscent of the complications faced by early Artificial Intelligence (AI) researchers in defining intelligence. Emergent computation allows the constraints of the task to be represented more naturally and permits only pertinent task-specific knowledge to emerge in the course of solving the problem. For accepting that a system is displaying emergent behavior, the system should be constructed by describing local elementary interactions between components in a different way than describing global behavior and properties of the running system over a period of time. In this paper, we introduce a general model for describing the emergent computational strategies.

INTRODUCTION

AI has made great strides in computational problem solving using explicitly represented knowledge extracted from the task. If we continue to use explicitly represented knowledge exclusively for computational problem solving, we may never computationally accomplish a level of problem-solving performance equal to humans. The need for more effective methods to generate and maintain other global nonfunctional properties suggests an approach analogous to those of natural processes in biological systems, social behavior, and economic systems in generating emergent properties (Angelone 1994; Gotts 2000; Hassan and Tazaki 2001; Olson and Siqueira 1995; Thornton 1997). Emergence system can be defined as: "A system composed
of independent agents, which behave according to explicit instructions. The system exhibits implicit spatial and/or temporal patterns that arise as a result of interactions between these subcomponents and/or between them and their environment. The patterns are apparent at a higher level than the agents, and are not explicitly coded in their specifications” (Olson and Sequeira 1995). The term “environment” can convey two meanings (Sipper 1995). In the strict sense, it refers to the surroundings, excluding the organisms themselves, while the broad sense refers to the total system, i.e., surroundings and interacting organisms.

A Cellular Automaton (CA) (Hanson and Crutchfield 1997), as the term is used in this paper, is a discrete state system consisting of a countable network of identical cells that interact with their neighbors. This network can take on any number of dimensions, starting from a one-dimensional string of cells. The Cellular Automata model is perhaps the simplest, most general model available (Sipper 1995). It is simple in that the basic units are small, local, finite state machines (cells). It is general because Cellular Automata supports universal computation, and the rules represent a general form of local interaction. The two-dimension Cellular Automata model is a grid of squares, each square having surrounding adjacent neighbors. A cell occupying a square is born, lives, or dies based on the number of living neighbors it has. Cellular Automata is considered a dynamical system in which space and time are discrete; each entity or cell is in one of the \( k \) states at any given time, and all cells change their states at the same instant, according to local interaction rules. The rules of local interaction change the cell’s state based on its current state, in addition to the state of the cells that are adjacent to it. The main difficulty with the Cellular Automata approach seems to lie with the extreme low-level representation of the interactions, and the important issue is the coding of the problem onto the Cellular Automata structure, which is non-trivial and may be complex.

We construct, in this paper, a new model of a Cellular Automata system. In this new model, the behavior of the overall system emerges from the interactions of the quasi-independent computational components or cells. Closely related to the concept of emergence is that of evolution in natural settings, as well as in artificial ones.

Another main interest in this paper lies in studying evolution and adaptation in the model. The paper is organized as follows: in the next section, we detail the basic design of our model. The section after that delineates the functioning of our system that displays several behaviors, including: reproducing, fitness value, and evolution of rules in this model. In the “Case Study,” we will provide details of the traffic system, as a case study of our new model. However, the model is not specific to the real-life system, but universal to the emerging system. The discussion of the model, with the advantages and drawbacks, is outlined “Discussion” as well as the differences
between our new model and the Cellular Automata model. Finally, the paper is concluded.

THE FORMAL DEFINITIONS OF THE NEW CELLULAR AUTOMATA MODEL

In this section, we describe the overall design of our automaton. The model is a two-dimensional grid of cells, each cell containing the same or different rules according to which cell states are updated in a synchronous and local manner. The neighbors of any cell in the grid are the cell itself plus the four orthogonal adjacent cells (Figure 1). Once the automaton was embedded in the grid, the cell, as an individual finite state machine, began to follow the rule that applied to it. A single cell cannot do much without interacting with other cells, and it has no concept of the whole. Yet, in combination, it can play its part in producing complex results as emergent from local interactions. Briefly, each cell from the grid can, in one time step:

1. Access its state and that of its immediate neighbors.
2. Change its state and the states of its immediate neighbors. Contention occurs when more than one neighbor attempts to change the state of the same cell. Such a situation is resolved randomly, i.e., one of the neighbors “wins” and decides the cell's state at the next time step.
3. Copy its rule into a neighboring dead cell (travail/self-reproducing). Contention occurs if more than one neighbor attempts to copy itself into the same cell. Such a situation is resolved randomly, i.e., one of the neighbors “wins” and copies its rule onto the cell.
4. Neither read nor write directly to other cells except its immediate neighbors.

The lattice starts out with an initial configuration (IC) of cell states in which the following hold:

- Most cells are in 0-state (initial state).

![FIGURE 1. The neighbor's cells.](image-url)
There are, however, an infinite number of cells in a state other than 0, i.e., \( \in \{1, 2, \ldots, k\} \), but these are distributed at random. That is, there is some probability \( P \) (non-zero, but arbitrarily low) that a cell is in a state not 0, and the state of each cell is determined independently.

The model performing a computation means that the input to the computation is encoded as the IC, the output is decoded from the configuration reached at some later time steps, and the intermediate steps that transform the input to the output are taken as the steps in the computation.

Every cell changes its state with time according to a rule, which is local and deterministic. We consider in this paper a non-uniform case of Cellular Automata (Sipper 1995) where different cells may contain different rules. At a given moment, only one rule is active for the cell and determines the cell’s function. A non-active rule may be activated in the next time steps.

Let any cell in the lattice be labeled by its position \( c = (i, j) \), where \( i \) and \( j \) are the row and column indices. A function \( S_c(t) = S(t; i, j) \) is associated with the lattice to describe the state of cell \( c \) at time step \( t \). The rules of the model are specific as to how the state \( S(t + 1; i, j) \) is to be computed from the States at time step \( t \) according to the equation:

\[
S(t + 1; i, j) = \lfloor S(t; i, j) + \delta \rfloor \mod k
\]

where \(-k \leq \delta \leq k\) and \( k \) is the number of states for cell \( c \). If the function \( S_c = 0 \), then cell \( c \) dies. We can assign the expected formula for \( \delta \) as:

\[
\delta = \begin{cases} 
\mu, & \text{if condition(a) true,} \\
-S(t; i, j), & \text{if condition(b) true,} \\
0, & \text{otherwise.}
\end{cases}
\]

where \( \mu \) is a positive integer number less than \( k \); we often use it as \( \mu = 1 \). Formulas (1) and (2) can be considered as the general formula for any rule of a Cellular Automata model. Conditions (a) and (b) differ from one rule to another, and the evolutionary process takes place under these conditions of the rule. Appendix (1) mentions some examples of conditions (a) and (b).

The set of rules for a cell, in many cases, is only one rule per cell, and, if there are more than one rule, then only one is active at a time. The other rules may be activated in the next time steps. Again, the set of rules differ in the form of conditions (a) and (b), but all rules have the same general form as in formula (1). It is interesting to observe the formation of such a high-order structure, which operates by applying local rules. Let one of the states for the cell be reference state \( R \). If we let \( P(R) \) be the percentage of transitions to state \( R \), then we define the parameter, \( \lambda = 1 - P(R) \). We now use \( \lambda \) to generate random Cellular Automata transition rules as follows. Pick a
rule for $R$ and divide the remaining probability of $1 - P(R)$ equally among the remaining states. Thus, for $\lambda = 0$, all the transitions save the non-$R$ homogeneous neighborhoods, will be to state $R$ for $\lambda = 0.5$. Half of the transitions will be to state $R$ and the remaining transitions will be equally distributed among the other states. For $\lambda = 1$, none of the transitions will be to $R$, and the other states will be represented equally. Now observe an example of emergent computation as the measurement of the density of the state $R$. Let $D_i$ be the density of state $R$ at the initial configuration (IC). Figure 2 shows the deviations from the initial density $D_i$ with time. The deviation from the initial state density is not constant through time steps, but it varies and emerges from the interaction between cells and is updated to other states.

From another view, the dynamical behavior of our model may give rise to emergent computation, referring to the appearance of global information processing capabilities that are not explicitly represented in the system elementary components (cells) nor in their local interconnections between them. An example of such emergent computation to use in this model is to determine the global density of configurations in the lattice. To formalize the notation of computational strategy in our model, and the resulting emergent computation, we model the Cellular Automata behavior in this system moving away from the underlying individual cells in a configuration to a higher-level description. Emergent computation is significantly different from classic computational paradigm (Kitano 1996), where control is top-down and centralized.

Figure 3 shows the state transition in our model, where the number of states is $k = 8$. We can observe that up to time $t = 52$, the lattices keep the symmetry in configuration, and, after that, the lattices take a random

**FIGURE 2.** Statistical transients: The emergent behavior of density of one state $R$ through iterations.
FIGURE 3. Transition of cell states in the model \((k = 8)\). Figures in cells stand for state values.

FIGURE 4. The number of cells that get the maximum state value \((k = 8)\).
distribution of states. Figure 4 shows the number of cells that take the maximum state through increasing the time step with the same parameter as in Figure 3. We observe from Figure 4 that the number of cells that take the maximum state is not fixed, and, in some time steps, there exists large numbers of cells with maximum state, e.g., between \( t = 100 \) and \( t = 120 \). But, in another set of time steps, this number becomes very small, e.g., between time steps \( t = 250 \) and \( t = 270 \).

Now define two important terms (Gotts 2000): pattern and cluster. Pattern is a finite set of cells in the same state other than 0. A cluster also called structure or configuration is a maximal set of cells in a state other than 0, such that a continuous path of neighborhood links going through no state 0 cells joins any two members. A pattern is a cluster and the number of cells in the cluster is called the size of the cluster.

It is interesting to observe the growth of patterns and clusters as emergent behavior in our model. Figure 5(a) shows some clusters that emerge from the initial configuration of the lattice. The figure proves that the technique of emergent computation provides growth of clusters. We observe that the clusters of size two that exist in the initial configuration grow to a cluster of size three after five time steps. Through that time, the cluster changes its shape many times and then grows to a cluster of size four after one time step from a growth time of size three. In the next time step, the cluster will extend itself to a cluster of size five. From the observation of emergent growth in the model, we find the shape cluster of size three in the initial configuration grow to a cluster of size five after one time step, but the other shapes have grown to clusters of size four first, after one or more time steps, before going to a cluster of size five (see the figure).

Figure 5(b) shows the growth of different patterns in the lattice. The patterns of size two have grown to a size three after five to eight time steps, depending on the shape of pattern (see the figure). In another way, the patterns of size three grow to patterns of size four after two time steps for some shapes and more than ten time steps for other shapes. But patterns of size four take varying times to grow to size five. In many cases (see the figure), patterns of size four change their shape first to other shapes with the same size in time between five and eleven and, after that, grow to patterns of size five. Except the block shape, it takes only one time step to grow. We observe that only the block shape pattern of size four does not change its shape to a pattern of the same size before it grows to a size-five pattern.

To increase the emergent behavior in the system and to make it more close to a biology system, we add what is called cell lifetime. Each organism has a local lifetime, after that, it dies. Figure 6 shows the values of the cells' lifetime for some cells of the grid. This idea will increase the varying state update in the model, which implies an increase in the emergent behavior that was observed in the model.
FIGURE 5. (a) Emergent of patterns in the model, (b) Emergent of clusters in the model.
MODEL CHARACTERISTICS

We present, in this section, the properties of our system, explicating its main functionalities: reproduction, fitness, and evolution. There are at least two possible kinds of cell reproduction, which can be described in this model. First, no cell replicates autonomously: all require interaction with other cells in a non-uniform environment. That will be called share-building operation. However, the same is true of real organisms; it is replication in an otherwise uniform environment that is un-biological. A second cell reproducing is called a self-replication operation and that is applied to the model, where any cell or organism develops into multiple copies of itself.

The Fitness

In this sub-section, we enhance our model by adding a measure of cell fitness, specifying how well it performs in a certain environment, depending upon the environment under consideration and the interaction with each other. This value is useful to decide whether the rule of the cell needs to apply the evaluation process with it or not.

Our approach is applied when each cell contains a single strategy and the cell’s score is then compared to its neighbour’s, and when the cell has a neighbor with higher fitness than its own, we apply the evolutionary process to enhance or change its rule. A cell’s fitness at time \( t \), can be defined as:

\[
 f^t(i,j) = S_c(t) + \alpha \left[ \frac{i}{5} + \frac{j}{5} + \sum_{i=1}^{4} S_i(t) \right]
\]

(3)

where \( S_c(t) \) is the state of the cell \( c \) at time step \( t \), \( S_i(t) \) is the state of cell \( i \) of the neighbors of cell \( c \), and \( \alpha \) is a parameter. The fitness values after the 100-time step for the sample grid is shown in Figure 7(a), and the varying of
the fitness value for one cell up to $t = 200$ is shown in Figure 7(b). From the figure, it is clear that the cells differ in their values of fitness, and the number of cells with maximum value in Figure 7(a) is not equal to the number of minimum values, which means the values of fitness are not uniformly distributed. Also, for fitness values with time, as in Figure 7(b), the values are changed quickly, but not as with linear growth, and are also non-uniform distributed.

A Self-Reproducing Technique

The effect of the local behaviors caused a global behavior to emerge: the self-reproducing structure interacted with neighboring cells and changed some of their states. It transformed them into the materials—in terms of cell states and rules—that made up the original organism. Eventually, by following rules of transition, the organism made a duplicate of its main body from parent to child as it emerged from self-reproducing the elementary cells.
of the organism. If any cell was given a description of an alleged self-reproducing machine, this machine needs to understand that it is supposed to be reproducing the machine, including its state and rules, onto one of the neighbors.

There is a good way to support a self-reproduction in the model. Append a supervisory unit to the cell to handle precisely such tasks. Then the self-reproduction occurs in two phases. First, under the guidance of the supervisory unit, the cell builds a new cell, plus the supervisory unit. When this construction is complete, the supervisory unit copies the state of the cell and transfers a copy of rules onto the new cell.

We need now to distinguish between two paradigms: trivial self-reproduction and non-trivial reproduction in which the construction of the copy is actively directed by the configuration or cluster itself. Our self-reproduction technique involves a simple self-reproducing loop motivated by Langton's work (Perrier, Sipper, and Zahnd 1996; Sipper 1995) that described such a structure in uniform Cellular Automata. The important issue to note is the two different uses of information—interpreted and un-interpreted—which also occur in natural self-reproduction; the former being the process of translation, and the latter transcription. In Langton's loops, translation is accomplished when the instruction signals are executed as they reach the end of the construction arms, and upon the collision of signals with other signals. Transcription is accomplished by the duplication of signals at the arm junctions.

Langton's self-reproducing is a loop constructed in eight-state, five-neighbor cellular space. While not claiming to be construction-universal nor computation-universal, it displays non-trivial self-reproduction. The loop is basically a closed data path, consisting of a string of core cells in state 1, surrounded by sheath cells in state 2. Data paths are capable of transmitting data in the form of signals, which are packets of two co-traveling states; the signal state itself (state 4, 5, 6, or 7) is followed by the state 0.

The loop considered in our model consists of any number of cells (cluster) surrounded by cells in the same state $h$ where $0 < h < k$. The arm extends itself by copying its state and rule onto an adjoining cell. The new cluster then acts as data to the arm, thereby providing the description by which the loop form is replicated. Now we have two ways to decide the transmitted direction of reproduction between the mother structure and the daughter one, according to the number of cell state $k$. If $k > 4$, then we can assign one or more states for each direction, i.e., if the surrounded cells are in a state indicated by move left, then the structure will copy itself onto left cells. In another case, if $k < 4$, then the direction of copy will be determined randomly. After determining the direction of copy, the structure will be divided into rows or columns according to the direction (rows if the direction is left or right and column if the direction is up or
down). Now each row or column will move to its direction and the cells will copy themselves with the states onto adjacent cells in the direction of transmission until the structure completely copies itself. If the structure consists of size \( m \times n \) and is surrounded by cell states indicating to move left or right, it takes \((n + 1)\) time steps to complete this operation. Figure 8 shows the self-reproducing technique for a grid with the number of states \( k = 6 \). We assign the direction of transition depending on the state of the surrounded cells: state 1 and 5 move right, state 2 and 6 move left, state 3 moves up, and state 4 moves down. The structure consists of size \( m = 4 \) and \( n = 8 \), and surrounded cells in state 1, i.e., the structure, will copy itself onto right adjacent cells.

**Growth and Replication**

At this point, we present a system involving the growth and replication of complex structures, which are created from grid cells. We can observe two cell type—the share-builder cell and the self-reproducing cell—floating around on the grid. The self-reproducing cell type is described in the previous sub-section, and we will explain how the share-builder cell type works. A share-builder cell interacts with another share-builder cell to produce new offspring, which contain a number of different rules as a mixture of the non-equal parts of its parents. The offspring begins with state 0 and determines its state according to its rule from the next time step. This operation is similar to the crossover operation that is used in Genetic Algorithms (Koza 1999). The parents may die immediately after the operation is complete or continue to live.

Let \((i,j)\) denote the grid position of the first share-builder, and \((i_n,j_n)\) the grid position of the randomly selected share-builder neighbor. The share-building operation is performed between the two groups of rules in cell \((i,j)\) and cell \((i_n,j_n)\), with probability \(P_b\). This probability can be fixed or not. If we need it dynamic, it can calculated as \(f(i,j) + f(i_n,j_n))/2\), where \(f(i,j)\) is the fitness of the cell at position \((i,j)\) and \(f(i_n,j_n)\) is the fitness of the selected neighbor for the share-building operation. If the cell has no share-builder neighbors, then no operation is effected. Note that the share-building operation is somewhat different than the crossover operation due to its “asymmetry:” cell \((i,j)\) selects cell \((i_n,j_n)\) while cell \((i_n,j_n)\) may select a different cell, i.e., cell \((i',j')\) such that \((i',j') \neq (i,j)\). It is felt that this slightly decreases the coupling between cells, thus enhancing locality and generality.

The share-building operation involves two cells operating in unison where each cell builds half of the structure of a new organism. Again, it is crucial that the whole process be local in nature since no global indicators can be used. The cells are provided for by the added feature above, which
FIGURE 8. Self reproducing ($k = 6, m = 4, n = 8$, and transition direction is to right).
specifies that a cell may contain a number of different rules, where only one is active at a given moment.

It is important to note the difference between our approach and Genetic Algorithms. Though we apply the operation in a similar fashion to the genetic operator, there is no selection mechanism operating on a global level using the total fitness of the entire population. Note also in the standard Genetic Algorithm model, each entity is an independent coding of a problem solution interacting only with the fitness function, never seeing the other entities in the population nor the general environment that exists. In contrast, in our case, fitness depends on interactions of evolving organisms operating in an environment.

**Evolution of the Rules**

Evaluation takes place not only in the state space as in the Cellular Automata model, but also in rule space, i.e., rules may change (evolve) over time. This process is applied to the rule of the cell when the cell’s fitness is less than all fitness values of the cells around it. The evolution of the set of rules that is stored in the lattice is achieved by using the genetic programming algorithm technique (Koza 1999). There exists three types of interactions or operations. First, replace one rule with another one (the same as reproduction operation). Second, interaction is made as mutation operation; one or more modifications are made to the rule to produce a new one. Third, the interaction exchanges part of the randomly selected rule with another part of a randomly selected one to make a new rule (crossover operator). Similar to the genetic programming algorithm, the initial population consists of a set of trees, so we will construct this set of trees corresponding to a set of rules that we have, i.e., each rule will represent one individual in the population. We will use here what is called a multi-tree technique (Langdon 1995) where the rule consists of two parts: condition (a) and condition (b). Each condition is represented by one tree. When the rule is used, e.g., during its fitness testing, then the tree corresponding to the desired condition evolves. So each individual within the population is composed of two trees. This multiple-tree architecture was chosen so that each tree contains a part, which has evolved for a single purpose. We define the genetic operations to act upon only one tree at a time; the other tree is unchanged and is copied directly from the first parent to the offspring. We have an example to show how a rule can be represented as a tree.

*Example.* Let Rule 1 from the appendix, which mentions:

**Rule 1.** The state \( S_c(t+1) \) of cell \( c \) at time step \( t+1 \) will increase by \( \mu = 1 \) until it reaches maximum state \( k \) if condition (a) is true, and it returns
to the initial state 0 if condition (b) is true. Otherwise, it will not be changed. 
*Condition (a).*

$$\sum_{i=1}^{4} y_i \geq S_c(t)$$

*Condition (b).*

$$\sum_{i=1}^{4} y_i \leq 2$$

Where $y_i$ is defined as $y_i = \begin{cases} 0, & S_i(t) = 0 \\ 1, & S_i(t) > 0 \end{cases}$, and $S_i(t)$ is the state of cell $i$ of the neighbors at time step $t$. This rule can be represented as two trees in Figure 9.

After $n$ time steps determined by the cell and its neighbors, depending on the fitness values, the cell simultaneously executes the evolitional algorithm. The first step in this algorithm is the construction of a set of trees corresponding to the set of rules, using its rule and the rules of its neighbors. Following that, two genetic operators are applied: The crossover operator and the mutation operator. The crossover operator is a mapping $C : C(T_1, T_2) = (T'_1, T'_2)$, where $T_1$, $T_2$ are randomly selected trees called parents and $T'_1$, $T'_2$ are called offspring (see Figure 10). It operates on two parental trees and creates two new offspring consisting of parts of each parent. The parents are replaced by the offspring where these offspring trees are typically of different sizes and shapes than their parents.
The mutation operator generates a unique offspring tree $T'$ from existing tree $T$, i.e., $M(T) = T'$ where $T, T'$ are randomly selected trees (see Figure 11). It operates on one parental tree and creates one new offspring to be inserted into the set of rules at the next time step. The fitness function that is used with the evolutionary algorithm is the fitness function (3) of the cell that has this rule. After finishing the evolutionary algorithm, trees are again translated into rules and stored for the next time step.

We can summarize this algorithm by the following steps:

1. Convert rules from the cells into trees.
2. Perform crossover operation according to the probability $P_c$.
3. Perform mutation operation according to the probability $P_m$.
4. Replace old population with offspring, and return to step 2 until the maximum number of generation is reached.
5. Convert trees into rules and store this set of rules.
CASE STUDY (TRAFFIC SYSTEM)

We propose a road traffic model suitable to an urban environment as a case study of our new model described in this paper. North, east, south, and west car displacements and road crossings are possible. $k$-states multi-speed car motion is found to be a crucial ingredient to describe highway traffic and phenomena. If the cell has state 0, then it is free of cars. The cell with state $0 < h \leq k$ has a car with $h$ speed, where $k$ is finite (called maximum velocity).

We would like to construct a street network. This requires a model, in order to deal with several cars entering the same road junction. In addition, drivers' decisions at crossroads (going straight or turning) should be defined. The basic idea is to consider the grid of cells as three types: the first type of cells makes groups to represent buildings (type-$A$ cell), the second type of cells constructs groups representing the street along which a car can move (type-$B$ cell), and the third cell type represents the free space cells that neither connect with a building nor with any road cells (type-$C$ cell). The type $B$ cells take the state from 0 to $k$, which represents the speed of cars. Type-$A$ cells take state $(k+1)$ and type-$C$ cells take state $(k+2)$. We assume that a horizontal road consists of two groups of cells, one for eastward motion and another for westward motion. Similarly, vertical streets are composed of
northbound and southbound groups of cells where all cars travel in the same direction of the road. Their positions are updated synchronously, in successive iterations (time steps). During motion, each car can be at rest or jump to the nearest neighbor site, using its speed from 1 to \( k \), along the direction of motion. The drivers are shortsighted and do not know whether the car in front will move or if it is stuck behind another car, so the car cannot move until an empty cell has appeared in front of it. In the overall system, the car distribution tries to self-adjust to a situation.

We will consider an alternate update of each direction of motion, as if there were synchronized traffic lights at each lattice site allowing horizontal street motion at two time steps and vertical street motion at the next two time steps. Figure 12 represents a road crossing where we need four traffic lights. In the first time step, the light allows vertical motion, so car \( a \) and \( b \) can move along and can also turn right. In the next time step, the traffic light allows car \( a \) and car \( b \) to turn left only. The third time step is the opposite case, i.e., horizontal motion is allowed and cars \( c \) and \( d \) can go along or turn right. The fourth traffic light allows cars \( c \) and \( d \) to turn left, and then these four time steps are repeated.

If we try to apply the characteristics of our model of Cellular Automata to this application of a traffic system, we need to define the terms in this system such as the cell lifetime, reproducing, and the fitness value. The cell
Figure 13. The amounts of cells enter the serves state (lifetime = 0).

lifetime here means that after a number of time steps, the cell is closed and takes state (-1) for one time step to check it (service time). This period or lifetime depends on the cell itself, i.e., type-A cells (building cells) have also a limited lifetime, but it is very long compared to road cells' (type-B cells) lifetime. When the life of type-A cell is over, the cell takes the state \((k+2)\) or becomes free space (type-C cell). Figure 13 shows the amount of lifetime for cells of type B that enter the period of service time, or when their lifetime is over.

The reproducing system that was described in the previous section can be observed in our model of a traffic system as follows. When the type-A cell (building cell) finds that one of its neighbor cells is a free space cell (type-C cell), the type-A cell can copy itself onto this free space cell. In the same manner, the road cell (type-B cell) can copy itself onto the free space cell of its neighbors, but it begins with an empty car state or state 0. Using this technique of reproduction, the road network will be extended and take the shape of a round trip.

We can define the fitness value for cells as the priority for each group of cells that represents one road. Each road or group of cells has this priority value over the whole system of streets, i.e., all cells of the road have the same value of fitness. The car drivers prefer to travel over cells or roads with high priority. This priority or fitness value can be computed, depending on the average velocity along the street. In addition, the number of deadlocks that occurred on this street over the previous time steps can affect its fitness value. The fitness function can be defined for the traffic system as

\[ g_y(t) = \frac{\bar{v}_y}{d_y} \]  

where \(\bar{v}_y\) is the average velocity for the road \(y\), and \(d_y\) is the density of cars in this road \(y\).
It is possible to observe the emergent behavior in this application of our new model. The emergent behavior in this model lies in two levels as one emerges from the other. First, the roads in the system emerge from the simple cells, and the interaction between cells produces the movement of cars with different speeds. The properties and rules of each road emerge from the global result of the group of cells that build the road, i.e., the density and fitness value of the road emerge from the interactions between cells with the movement of the cars in the road. On another level, the whole performance of the system or global road network emerges from the performance of all roads in the model, as well as from the reproducing of cells in each road, which extends the road. Here, we consider the uniform case of Cellular Automata, with respect to the rule transition of cars in the first level. While with the level of the roads, we can apply a non-uniform case, i.e., different roads have different rules. The rules of the road describe the length, width, direction of car movement, and the fitness value of this road. The road width can be a mixture of three types (one, two, or three cells), so that, in some places, where the density of cars is high for a long time and the cells have a free space cell (type-C cell) beside them, the road can extend itself by allowing the cells to reproduce.

Let the transition rule for the movement of cars in the system derive from formula (1) and be mentioned as: the velocity \( v \) of the car in the cell \( c \) is changed according to the rule:

\[
v_{t+1}(c) = v_t(c) + \delta, \tag{5}\]

where \( v_{t+1} \) is the velocity of car in cell \( c \) at time step \( (t + 1) \), \( v_t \) is the velocity at time step \( t \), and \( \delta \) is a parameter that takes value \(-k \leq \delta \leq k\). If \( v_t = 0 \), then the cell is empty. If \( v_t = v \), then the cell has a car move with velocity \( v \). According to the previous rule, the velocity of each car is increasing or decreasing according to the state of the road, i.e., according to the value of \( \delta \), which can be obtained from the formula:

\[
\delta = \begin{cases} 
1, & v < \text{Next\_car} < k \\
-m, & \text{otherwise}
\end{cases} \tag{6}
\]

where \( \text{Next\_car} \) is the number of cells between the car and the car crossing in front of it, i.e., the car will accelerate its speed if its velocity if less than the maximum velocity \( k \), as well as if it has a free space in front of it. Otherwise, it will reduce its velocity by integer quantity \( m \); this quantity \( m \) can be calculated as:

\[
m = \begin{cases} 
v_t(c) - \text{Next\_car} & \text{if no crossing} \\
v_t(c) + 1 & \text{if crossing}
\end{cases} \tag{7}
\]
i.e., if the car has another car in front of it at distance $\text{Next}_\text{car}$ and its velocity is more than the quantity $\text{Next}_\text{car}$, then it will be reduced until it is equal to $\text{Next}_\text{car}$. But if the car has a road crossing in front of it, then the velocity will return to the initial state ($v = 1$).

In the model of a traffic system, the goal of the car or driver is the strategy: \textit{Drive as fast as you can and slow down if you have to!} While traveling, the goal of the system is always to get a stable flow or high global average speed for all cars. Figure 14 shows the varying flow of the traffic system with varying density.

We will go in new direction to increase the observation of emergent computation in the system by using what is called relative speed. Relative speed could measure the speed of each car by comparing its speed against the speed of other members of the system. Two ways to compute this relative speed have been investigated. First, the way we will use here, compute the global average speed for all cars $\bar{v}$, and then compare each speed value of car to this value as:
where $v_{rc}$ is the relative speed for car $c$, and $v_r$ is the speed of the car in cell $c$. A second way to measure relative speed is to pit the absolute car speed against each other in a single elimination tournament as shown in Figure 15.

This dependent speed measure is self-scaling in that as the power of the cars in the system increases, so does the difficulty of the speed. Since the relative speed is dependent on all the cars, it tracks through the cars non-monotonic development without the need for measuring the average member’s ability explicitly. By placing cars of the system in opposition, relative speed sets up interesting emergent dynamics. As the iterations continue, the ecological balance shifts in the system to take advantage of exploitable strategic niches. Given that the cars maintain some level of diversity, this is straightforward. The operation pressure toward a better flow of cars in the system emerges from the progressive diversity of cars. The constant diversity of the relative speed of cars provides a fertile environment for robust flow to emerge, providing a teleological effect.
We made the experiments using small sample data. The system size is \( L = 8366 \) cells (road cells). The system has fourteen horizontal streets and twenty-three vertical streets. The maximum velocity is five \((k = 5)\).

Figure 16 shows the variation of relative speed for some cars in the system. We observe also that the traffic is not distributed uniformly and some road segments are much more loaded than others. A simple strategy has emerged which specifies that a rule, upon occupation of a certain cell, remains there, thereby preventing occupation by another rule (which can only enter a vacant cell).

In this model, the traveling of cars lies only under the local control of driving, which gives the model an ability to describe it from the view of emergent computation. The emergent dynamical property of driving control logic is defined in this model clearly. The moving rule of formula (6) is shown to generate an emergent representation of a driver that can be viewed as an adaptively compensated local control system. Each driver has the freedom to go on or to stop at any site in the road. In addition, the driver has the ability to decide to go on or turn at the road crossing. Using the experimental result, the adaptive control and emergent controller property have appeared with the relative speed of cars and also with the effective density on the average speed of cars. Rule (6) produces realistic global traffic phenomena, such as jams and flow-density relationships emerging from the local control of cars. The model and global emergent of it are consistent with the more macroscopic concern of local iteration influences in an urban settlement evolution.

However, although the relationship of the model to global traffic structures has become clearer, the details of the local lattice site transition rules to some kind of driver control model have not. Transition rule (6) states that, in interaction with the lattice, the state updates the parameters of rules, and in local interaction among themselves, gives rise to an intermediate emergent phenomena that is justifiably called a local vehicle control system. That is, the driver emerges between the levels of the rule and the global traffic phenomenology. The driver logic, in this sense, is a much higher fidelity representation than previously suspected.

**DISCUSSION**

The model has provided a direct approach to studying how dynamical systems perform emergent computation; that is, how the interaction of simple components with local information storage and communication gives rise to coordinated global information processing. In real-life situations, the topology of the interconnections that give meaning to the term "immediate neighbors" can change frequently. Although every cell participating in the system must be able to communicate with its immediate neighbors, the system itself should not depend on the knowledge of the overall system
topology. The state transition within each cell could be identical throughout the system or unique to each cell. In practice, the state transitions within cells can most conveniently be viewed as shared by all cells, but with local adaptations as a function of either static or dynamic local conditions.

In a situation similar to that studied with the Cellular Automata model, we observe the same emergent behavior, which is described in this paper. The emergent behavior can, as a whole system, be applied to our new model, thus the global result of the model emerges from the interactions between small parts of the system (cells in the grid). On the other hand, our new model differs from the Cellular Automata model on some points:

1. Each cell of the grid can be considered as a whole system where it, at any time step, contains a set of different rules. At a given moment, only one rule is active and determines the cell’s function. A non-active rule may be activated in the next time steps. This feature does not exist in the Cellular Automata model where only one rule is applied for all cells in the system.

2. Each cell in our new model has a limited lifetime. This idea is not present in the Cellular Automata model. But it is useful to construct the traffic system, where we applied it as a servicing time to check for this part of the street. And it is an important factor in the biological model, where we need it to construct a model similar to the biology model, with the results of an emergence system.

3. Evolution takes place not only in state space as in the Cellular Automata model, but also in rule space, i.e., rules may change (evolve) overtime.

4. Each cell is considered to have certain fitness, depending upon the environment under consideration and the interaction with each other. The evolutionary process depends on the fitness value of the applied cells.

Looking further into the future, we can imagine a system based on our new model, comprising several interacting reproducing machines, each with a different functionality. This new technique of reproducing cells that is defined in this paper consists of two types. The first kind of reproducing cells is similar to Langton’s loop of self-reproduction, but the second kind is the first time we’ll describe it here. It is called share-building reproducing. The share-building technique depends on the interaction between two cells to produce offspring. These interactions could be of a cooperative or a competitive nature—the end result of the reproducing process being a system displaying some global functioning.

One of the advantages of our new model is the opportunity it offers in performing studies of the evolutionary process. It was noted that rules tend to self-organize in this model where an evolutionary component that is added to the system is increasing its adaptive capabilities. The paradigm should be well conditioned under genetic operators. This requirement is
less formal, meaning that evolution between successive time steps is usually well behaved, i.e., discontinuities occur only occasionally. In the other way, the evolutionary studies we performed were carried out in rather small grids, consisting of only a few numbers of cells. It seems reasonable to assume that, in order to evolve, interesting creatures require a larger number of units.

We have shown that urban traffic in a road network has some new generic properties. We proposed the concepts of road crossing, cell lifetime, and cell fitness to describe such systems. We observe that the overall dynamic is quite sensitive to the driver’s behavior at a road crossing when choosing a destination. Note that our model of a traffic system can be extended so as to give a different trip plan to each car. Of course, the continuous micro-traffic simulator provides more flexibility to study emergent computation. Our system differs from other two-dimensional Cellular Automata traffic models (Knospe, Santen, and Schadschneider 1999; Nagel and Paczuski 1995; Neubert, Lee, and Schreckenberg 1999) in two respects. First, these models do not have the concept of streets. Cars can be anywhere in a two-dimensional grid and can, typically, move to any nearest empty neighbor cell. As a consequence, they do not have the idea of a road crossing. Second, they have a rather primitive motion rule in order to avoid double occupancy. In addition, this model differs from the model in (Chopard, Luthi, and Queloz 1996) in that our model provides for multi-speed moving. The traffic system in (Chopard, Luthi, and Queloz 1996) has only two speed states: 0 if cell is empty and 1 if cell has one car, which can move one cell if the cell in front of it is empty. In addition, our system has some new notions, such as road priority and serving time (cell’s lifetime). In addition, the technique of the self-built road that was added to the model of the reproducing system is a new feature of the traffic system. We think that the road network can begin with a small number of roads and the system itself will build a whole network of roads, emerging from the interactions of the cells in the system. This idea is the target of our study with the traffic system.

A number of possible improvements and extensions to our system are possible:

1. The actual number of rules is small, thus rendering our system realizable. Still, it would be interesting to increase the number of rules.
2. The data that is used with the case study (traffic system) is simple, consisting of a small number of cells and a small number of cars. Unfortunately, the current lack of data concerning real traffic systems is the main difficulty in producing accurate predictions on the evolution of a specific traffic situation.
3. We discuss the trivial self-reproducing technique in the model of the traffic system; next, we want to add the non-trivial self-reproducing technique to
this model. We expect that we can develop this system to have the ability
to add or delete roads as the global system needs.
4. The evaluation algorithm is not applied to the traffic system in this paper,
but we will try to insert this feature to discuss the usefulness of roads and
to add or delete any road from the system.

CONCLUSION

Computer support for different human activities has grown in recent
years. Artificial life research opens new doors, providing us with novel
opportunities to explore issues, such as adaptation, evolution, and emer-
geence, that are central in both natural environments and manmade ones. In
this paper, we presented a system of simple organisms interacting in a two-
dimensional environment, which have the capacity to evolve. We first turned
our attention to designing a model that displayed several interesting behav-
iors. These included a self-reproducing function and a fitness function. After
our initial investigation of the Cellular Automata model, we turned our
attention to evolution in rule space, which occurs through changes in the
genotypes representing the rules by which the organisms operate. We have
also qualitatively analyzed the emergent computational strategies of the
model. The model provides a means to more rigorously formalize the notion
of emergent computational strategy in spatially extended systems and to
make quantitative predictions about the computational behavior and evolu-
tionary fitness of the model for Cellular Automata. This work has shed
light on the possibility of constructing such systems, and demonstrated the
feasibility of their practical implementation. This is an essential, quantitative
part of our research program, to understand how natural spatially extended
systems can perform globally coordinated computations and how evolu-
tionary processes can give rise to systems with sophisticated emergent com-
putational abilities. Since our model is an enhancement of the Cellular
Automata model, it is equivalent to the Turing machine or has computa-
tional completeness. There are still, however, many limitations that should be
addressed and a number of possible improvements and extensions to our
system.

REFERENCES


**APPENDIX**

We will show in this appendix some examples of the conditions (a) and (b), which described earlier are some rules that are used by our model through its lifetime. The state $S_i(t+1)$ of cell $c$ at time step $(t+1)$ will increase by $\mu$ ($\mu = 1$) until it reaches maximum state $k$, if condition (a) is true, and it will equal initial state 0, if condition (b) is true. Otherwise, it will not change. We need to define variable $y_i$, if the cell $i$ of the neighbors for cell $c$ is alive or not as:

$$y_i = \begin{cases} 
1, & S_i(t) > 0, \\
0, & S_i(t) = 0.
\end{cases}$$

where $S_i(t)$ is the state of cell $i$ at time $t$.

**Rule 1.**

**Condition (a).**

$$\sum_{i=1}^{4} y_i \geq S_c(t)$$

**Condition (b).**

$$\sum_{i=1}^{4} y_i \leq 2$$
Rule 2.
Condition (a).
\[ \sum_{i=1}^{4} y_i \geq S_c(t) - 1 \]

Condition (b).
\[ \sum_{i=1}^{4} y_i \geq 2 \]

Rule 3.
Condition (a).
\[ S_c(t) \leq \sum_{i=1}^{4} y_i + 2 \]

Condition (b).
\[ S_c(t) \geq \sum_{i=1}^{4} y_i - 1 \]

Rule 4.
Condition (a).
\[ \sum_{i=1}^{4} y_i + S_c(t) \geq 3 \]

Condition (b).
\[ S_c(t) > 1 \]

Rule 5.
Condition (a).
\[ S_c(t) - 5 \leq \sum_{i=1}^{4} y_i \]

Condition (b).
\[ S_c(t) + 3 \geq \sum_{i=1}^{4} y_i > 1 \]
Rule 6.
Condition (a).
\[ S_c(t) - 1 < \sum_{i=1}^{4} y_i \]

Condition (b).
\[ 2S_c(t) > \sum_{i=1}^{4} y_i \]

Rule 7.
Condition (a).
\[ \left( S_c(t) - \sum_{i=1}^{4} y_i \right) \geq 2 \]

Condition (b).
\[ \sum_{i=1}^{4} y_i \leq 3 \]

Rule 8.
Condition (a).
\[ S_c(t) \leq 3 \]

Condition (b).
\[ S_c(t) \leq \sum_{i=1}^{4} y_i + 2 \]

Rule 9.
Condition (a).
\[ S_c(t) > 1 \]

Condition (b).
\[ \sum_{i=1}^{4} y_i \geq 1 \]
Rule 10.
Condition (a).
\[ S_c(t) - 1 \geq \sum_{i=1}^{4} y_i \]

Condition (b).
\[ \sum_{i=1}^{4} y_i \leq 1 \]

Rule 11.
Condition (a).
\[ S_c(t) \geq \sum_{i=1}^{4} y_i + 2 \]

Condition (b).
\[ S_c(t) \leq 3 \]

Rule 12.
Condition (a).
\[ \sum_{i=1}^{4} y_i - 1 \leq S_c(t) \]

Condition (b).
\[ \sum_{i=1}^{4} y_i \leq 3 \]

Rule 13.
Condition (a).
\[ S_c(t) + \sum_{i=1}^{4} y_i \leq 3 \]

Condition (b).
\[ \sum_{i=1}^{4} y_i > 2 \]
Rule 14.
Condition (a).

\[ \sum_{i=1}^{4} y_i + S_e(t) > 2 \]

Condition (b).

\[ \sum_{i=1}^{4} y_i \leq 3 \]

Rule 15.
Condition (a).

\[ \sum_{i=1}^{4} y_i \leq S_e(t) - 1 \]

Condition (b).

\[ \sum_{i=1}^{4} y_i \leq 2 \]