Sensors and Materials, Vol. 38, No. 3 (2026) 1319–1334
MYU Tokyo

S & M 4379

Analysis of Sugarscape Game of Life as Computational Resource

Tadanobu Misawa[1*] and Kazuya Yamashita[2]

[1]Faculty of Engineering, University of Toyama,3190, Gofuku, Toyama-shi, Toyama 930-8555, Japan.
[2]Faculty of Education and Research Promotion, University of Toyama,
3190, Gofuku, Toyama-shi, Toyama 930-8555, Japan.

(Received November 27, 2025; accepted Feburary 24, 2026)

Keywords: cellular automaton, Game of Life, reservoir computing, lightweight machine learning, edge AI

Recent advancements in sensor development and network technologies have increased the demand for lightweight machine learning techniques that operate efficiently in computational resource-constrained environments, such as edge AI. Among these methods, reservoir computing has attracted attention as a form of lightweight machine learning. Reservoir computing enables fast learning owing to its low computational cost; however, achieving high accuracy requires proper reservoir design, with long-lasting chaotic dynamics generally preferred. In this study, we investigated whether the Sugarscape Game of Life (GoL) can function as a reservoir. The Sugarscape GoL, a model incorporating elements from Sugarscape, is expected to exhibit a range of chaotic dynamics depending on parameter settings. Experiments using random initial configurations demonstrated that the transient step count (step count needed to reach equilibrium) was significantly increased compared with the conventional GoL, and chaotic dynamics persisted for extended durations. In addition, varying the parameters altered the transient step count, producing diverse chaotic dynamics. Clarifying the characteristics of the Sugarscape GoL and applying them to reservoir computing can contribute to technologies that enhance daily convenience and industrial productivity in an IoT-driven society.

1. Introduction

Advances in sensor technology and network systems have accelerated the emergence of an IoT society, where nearly all objects are connected to the Internet. In such an environment, the data collected from sensors and devices have become increasingly vast and varied, promising improvements in both societal convenience and industrial productivity. Machine learning, especially deep learning, is being utilized to efficiently process and derive value from this vast, diverse data. For instance, the ease of collecting and scaling large datasets has enabled their application in large language models (LLMs), such as ChatGPT. However, computational costs have increased substantially. These large-scale models require immense training data and computational power, raising concerns about environmental impacts, such as increased energy consumption.(1)

*Corresponding author: e-mail: misawa@eng.u-toyama.ac.jp
https://doi.org/10.18494/SAM6086

ISSN 0914-4935 © MYU K.K.
https://myukk.org/

Consequently, research has shifted toward lightweight machine learning approaches that minimize computational demands through smaller models, with reservoir computing being one such method.(2) Reservoir computing provides a computational framework that employs a fixed reservoir and a trainable readout to approximate dynamic systems. In essence, reservoir computing achieves a substantial reduction in computational cost by training only the output weight parameters, in contrast to deep learning, which requires training all model parameters. Consequently, reservoir computing has been considered for edge AI applications,(3) where data gathered by various sensors is quickly processed at the edge.(4,5)

Designing an appropriate reservoir is crucial for achieving high performance in reservoir computing. The following properties are essential for a reservoir: nonlinearity, high dimensionality, and short-term memory. These properties generate chaotic dynamics and complex spatiotemporal patterns. Reservoir computing uses the complexity of these dynamics to its feature representability. Typical examples include echo state networks(6) and liquid state machines,(7) and various physical reservoirs have also been explored.(8) These include soft materials similar to octopus tentacles,(9) spin-wave systems,(10) and optical components.(11)

In this study, we employ the Sugarscape Game of Life (GoL)(12,13) as a reservoir. This model extends the two-dimensional cellular automaton known as the GoL(14) by introducing resources (nutrients), where the resources increase by a fixed amount each cycle, and when they exceed the parameter C, cells are generated by consuming resources with a parameter R. Thus, it more closely resembles natural ecosystems, such as cultured cell populations. In previous studies, it was demonstrated that the population dynamics of cultured cells can function as reservoirs in reservoir computing.(15) Cellular automatons have also been implemented as reservoirs, demonstrating their usefulness.(16,17) Therefore, reservoir computing can potentially be constructed using the Sugarscape GoL. However, even when reservoir dynamics are chaotic, their characteristics must be well understood to prevent premature convergence to equilibrium. Specific patterns in the Sugarscape GoL, such as oscillators(12) and methuselah (long-lived pattern),(13) were previously analyzed. However, reservoir applications necessitate examining the chaotic dynamics behavior that emerge from random initial configurations, particularly the transient step count.(18) The transient step count indicates the number of iterations needed for the system to reach an equilibrium state consisting solely of still lives and oscillators. A larger transient step count indicates more sustained chaotic dynamics and a greater variety of spatiotemporal patterns generated, making the model more suitable for reservoir computing. Moreover, because the Sugarscape GoL incorporates resource-related parameters, it is expected to generate a variety of chaotic dynamics depending on their values.

On the basis of these considerations, in this study, we examine the transient step count in the Sugarscape GoL to assess its potential use in reservoir computing.

2. Data, Materials, and Methods

The Sugarscape GoL is based on the Sugarscape model.(19) It employs binary cell states (0 or 1), where the birth of a cell depends on resource availability from surrounding cells, and resource consumption occurs in those surrounding cells upon birth. The detailed algorithm is described as follows.(12,13)

Sugarscape GoL

- **Rule 1-0:** When a cell is in state 0, and three of its eight neighbors are in state 1, the next state remains 0 if the condition $S < C$ is satisfied.
- **Rule 1-1**: When a cell is in state 0, and three of its eight neighbors are in state 1, if $S \geq C$, the cell consumes resource R and changes to state 1 in the next step.
- **Rule 2**: When a cell is in state 1 and has two or three neighboring cells in state 1, it remains in state 1 in the following step.
- **Rule 3**: In all other cases, the cell's state becomes 0 in the next step.

The parameters are defined as follows: S denotes the total resource value of the nine neighboring cells, C the birth threshold, and R the resource amount consumed at birth. Additionally, at each time step, $1/T$ of the resources are replenished, and each cell's resource level is constrained between a minimum of 0 and a maximum of 1. All cells update their states simultaneously in accordance with these rules.

This algorithm differs from the conventional GoL primarily in Rules 1-0 and 1-1. The conventional GoL has no resource constraints, and thus cells in state 0 with three cell neighbors unconditionally transition to state 1 in the next step. Therefore, the resource constraint introduced in the Sugarscape GoL reduces the number of new cells generated per step compared with the conventional GoL.

3. Results

3.1 Experimental overview

We evaluated the suitability of the Sugarscape GoL for reservoir computing by comparing the transient step count under random initial configurations with that of the conventional GoL.

We first conducted a parameter search to determine the transient step count in the Sugarscape GoL. Specifically, we set the cell space size to 50×50 with null boundaries, varied R from 0.1 to 1.0 in increments of 0.1, and varied C from 0.0 to 9.0 in increments of 0.1 at $T = 5.0$. For each parameter combination, 100 experiments with random initial configurations were performed to obtain the average transient step count. In this setup, the minimum resource value was 0, and thus the condition $C = 0.0$ corresponded to the conventional GoL.

Next, we analyzed array size dependence, that is, how the average transient step count changes with increasing cell space size, using the parameter combination that yielded the maximum transient step count in the previous experiment. For the conventional GoL, as the cell space size increases, the average transient step count grows approximately logarithmically.(18) For the Sugarscape GoL, we examined how this behavior changes with cell space size and parameter values. Finally, we conducted additional experiments on the basis of the obtained results for more detailed validation.

3.2 Experimental results

3.2.1 Parameter search

As Sugarscape GoL incorporates resource-related parameters, the effects of these parameters must be verified. Therefore, experiments were conducted on various combinations of these parameters. Figure 1 presents the results of the parameter search. Each plot depicts the graph obtained when parameter R is fixed and parameter C is varied. The horizontal axis indicates the value of C, and the vertical axis shows the average transient step count. Figure 1 demonstrates that the average transient step count begins to increase at C values corresponding to R = 0.3. When R is 0.4 or higher, certain C values cause a sharp rise in the average transient step count, suggesting behavior similar to that of the edge of chaos. In addition, as the value of parameter R increases, the value of parameter C that maximizes the average number of transient steps tends to decrease, and the maximum average transient step count differs in accordance with the parameter value, implying the presence of multiple forms of chaotic dynamics.

Figure 1 suggests that different applications of Rule 1-0 arising from parameter combinations give rise to various chaotic dynamics. Therefore, Fig. 2 illustrates the relationship between the frequency of Rule 1-0 application [(number of times Rule 1-0 is applied)/(number of times Rules 1-0 and 1-1 are applied), "rule change rate"] and the average transient step count. The horizontal axis represents the rule change rate, and the vertical axis represents the average transient step count. As shown in Fig. 2, the average transient step count increases when the rule change rate is very small. This indicates that even a minor rule change rate can substantially affect chaotic dynamics.

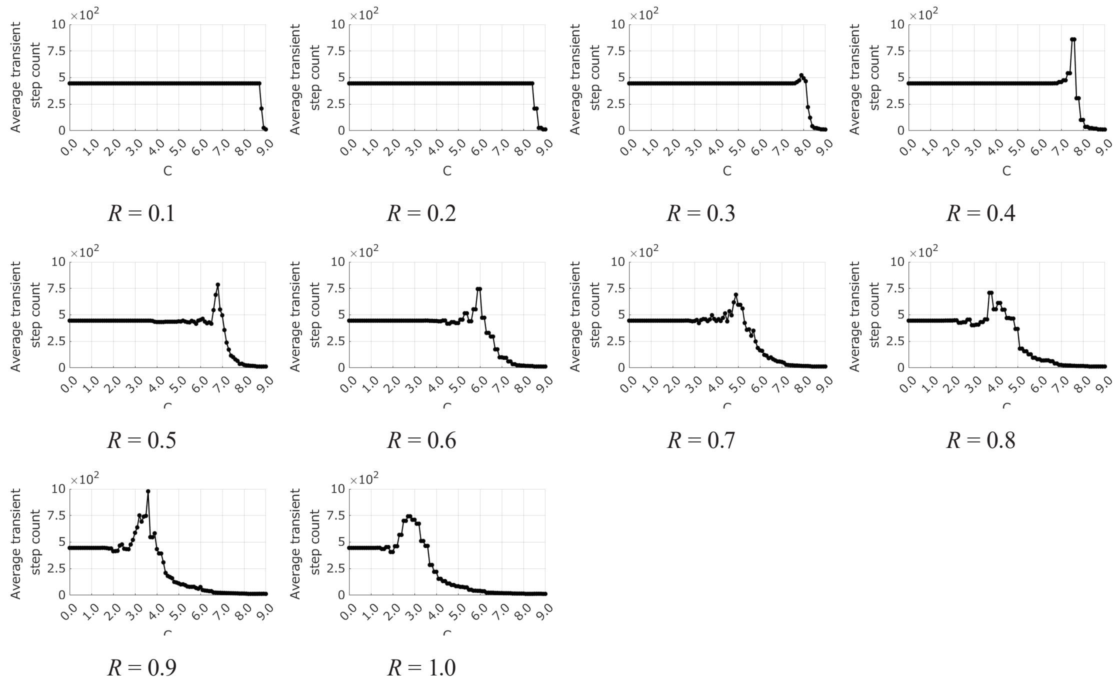


Fig. 1. Parameter search results.

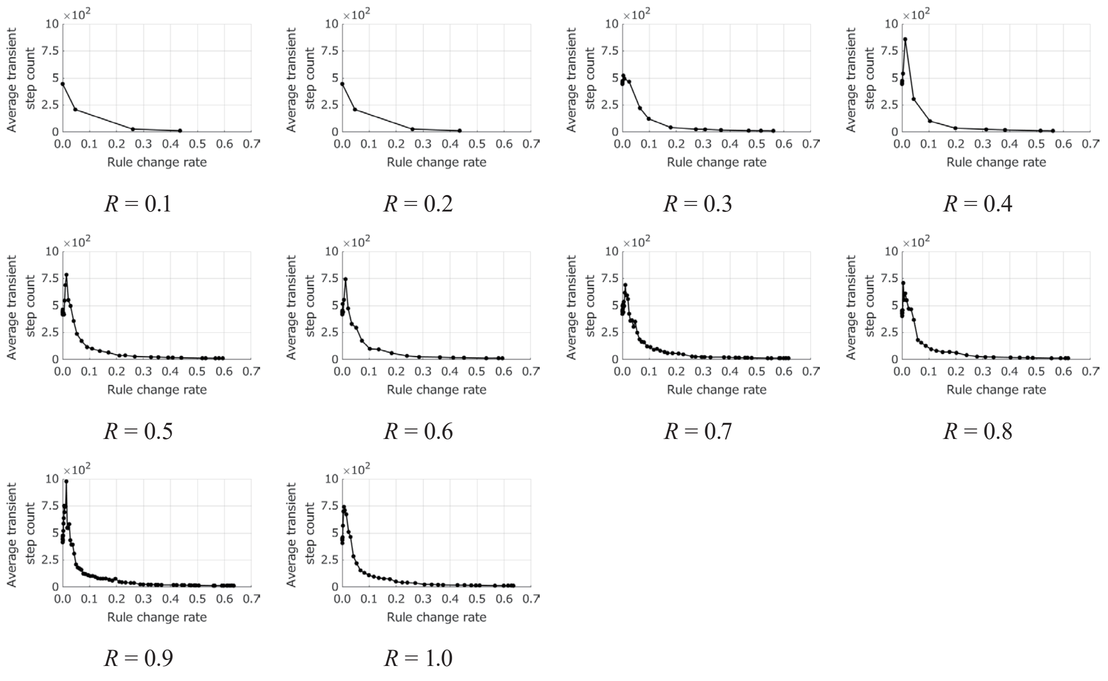


Fig. 2. Relationship between rate at which Rule 1-0 is applied (rule change rate) and transient step count.

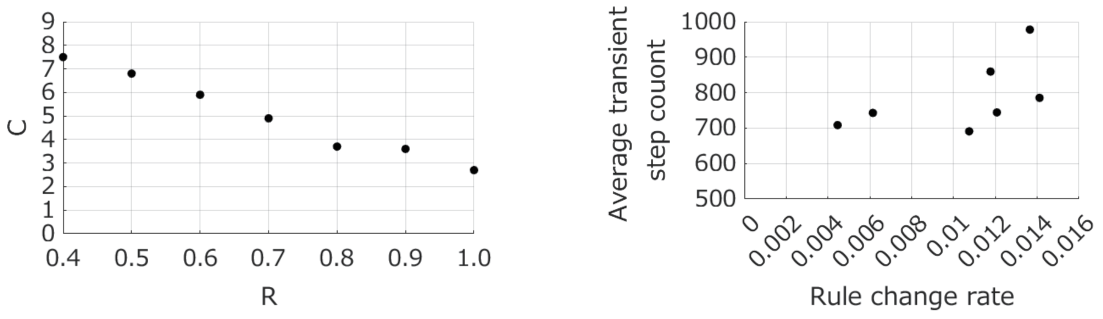
To understand the conditions under which more chaotic dynamics are generated in Sugarscape GoL, Table 1 summarizes the *C* values, the rule change rate, and the average transient step count when *R* is 0.4 or higher, at which the transient step count is maximized. Figures 3(a) and 3(b) display scatter plots showing the relationships among the parameters *R* and *C*, rule change rate, and average transient step count. Table 1 reveals that the rule change rate that maximizes the average transient step count lies between approximately 0.4% and 1.4%. The impact of such an extremely low rule change rate will be reexamined in Sect. 3.2.3. Figure 3(a) shows a negative correlation between the parameters *R* and *C*. Therefore, the value of *C* that maximizes the average transient step count can be approximated for a given *R*. Figure 3(b) shows a positive correlation between the rule change rate and the average transient step count, indicating that rule change rates within approximately 0.4–1.4% generally yield higher average transient step counts. In this study, the increment size of the parameters was set broadly. However, setting it more finely may further clarify the trend. The parameters listed in Table 1 are examined in the following section; however, for simplicity, only *R* values are referenced, whereas *C* values are omitted.

3.2.2 Array size dependence

Various spatiotemporal patterns are more likely to emerge in larger cell spaces, which aids the understanding of chaotic dynamics. Therefore, experiments that altered the array size were

Table 1
Summary of parameter search results.

R	*C*	Rule change rate	Average transient step count
0.4	7.5	0.011770	859.45
0.5	6.8	0.014113	785.19
0.6	5.9	0.012076	744.17
0.7	4.9	0.010748	690.79
0.8	3.7	0.004457	708.38
0.9	3.6	0.013651	977.76
1.0	2.7	0.006144	742.82



(a) (b)

Fig. 3. Relationships between *R* and *C*, rule change rate, and transient step count.

conducted. Figure 4 presents the average transient step count calculated from 100 random initial configurations as the cell space increases from 50 × 50 to 150 × 150 in increments of 10. In these experiments, we used parameters that maximized the average transient step count in both the conventional GoL and Sugarscape GoL. Figure 4(a) corresponds to null boundaries, and Fig. 4(b) shows torus boundaries. The horizontal axis represents cell space size, and the vertical axis represents the logarithm of the average transient step count. However, if any random initial configuration produced more than 100 million transient steps, that trial was terminated. As shown in Fig. 4, in the conventional GoL, the average transient step count appears to approach convergence as the cell space size increases. This observation aligns with the results of prior research,[18] suggesting a logarithmic relationship. In contrast, the Sugarscape GoL exhibits a sharp rise in the average transient step count with increasing cell space size, showing no signs of convergence and following an exponential trend. This effect is especially evident for torus boundaries. This result suggests that, unlike conventional GoL, which tends to limit the generated spatiotemporal patterns, Sugarscape GoL generates extremely diverse spatiotemporal patterns. Moreover, the average transient step count varies with *R*, and thus adjusting the parameters can produce diverse chaotic dynamics behavior. In addition, for the 150 × 150 cell space with *R* = 0.4 and 140 × 140 and 150 × 150 cell spaces with *R* = 0.9, transient step counts occasionally exceeded 100 million (those runs were halted and are therefore not displayed). Such extremely high values indicate that chaotic dynamics can persist over extended durations.

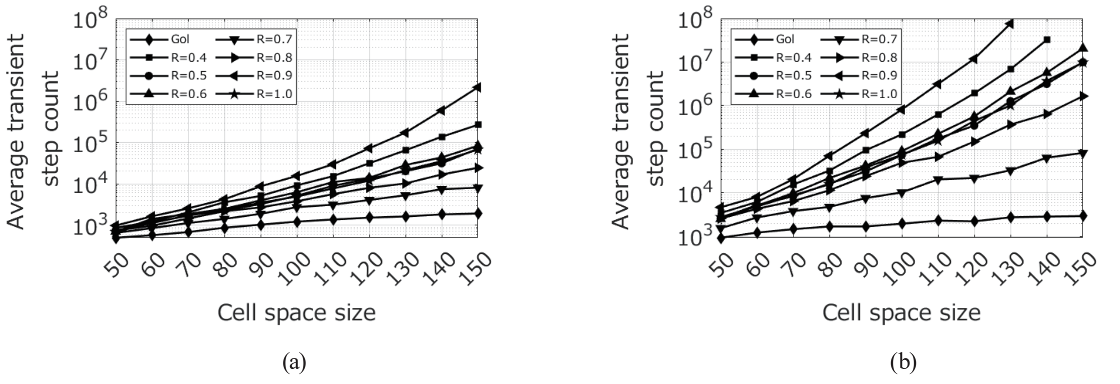


Fig. 4. Dependence on cell space size: (a) null boundaries and (b) torus boundaries.

3.2.3 Rule change rate

As shown in Sect. 3.2.1, the average transient step count increases when the rule change rate is low. This may be because rare rule changes significantly alter system patterns, resulting in higher average transient step counts. To test this, we conducted 100 experiments with random initial configurations on a 50 × 50 cell space, randomly setting the rule change rate between 0.1 and 50% in 0.1% increments. Figure 5 shows the results. In Fig. 5, the horizontal axis shows the actual random rule change rate, and the vertical axis shows the average transient step count. The figure demonstrates that the transient step count decreases as the random rule change rate increases. This suggests that rule changes in the Sugarscape GoL are meaningful rule changes.

We next examine the mechanism underlying these rule changes by investigating the fluctuations in resources and the state of neighboring cells, with $R = 0.4$ serving as the most straightforward example. When $R = 0.4$, resources decrease by 0.4 at birth and increase by 0.2 (1/5) per cycle, always returning to the maximum level within two steps. Therefore, in the current state, for any given cell, resources decrease by 0.4 if it was born one step earlier and by 0.2 if born two steps earlier. For a target cell to be born, three neighboring cells in state 1 are required, and this configuration depends on the number of cells born one and two steps earlier. The results of this analysis are summarized in Table 2. Table 2 lists the number of cells born one step earlier, the corresponding resource depletion, the number of cells born two steps earlier, and the total resource depletion across nine neighboring cells during state determination. For instance, if three cells were born one step earlier, the resource depletion equals 3 × 0.4 = 1.2. If five cells were born two steps earlier, their depletion is 5 × 0.2 = 1.0, giving a total depletion of 1.2 + 1.0 = 2.2. The maximum resource value per cell is 1, and thus the total resource available among nine neighbors equals 9 − 2.2 = 6.8.

When $R = 0.4$, the average transient step count reaches its peak when $C = 7.5$. As shown in Table 2, if three cells were born one step earlier, the total neighboring resources must exceed 7.5, indicating that the target cell can only be born if one or fewer cells were born two steps earlier. Similarly, if two cells were born one step earlier, the target cell can be born only if three or fewer cells were born two steps earlier; if one cell was born one step earlier, the threshold is five or fewer cells; and if none were born one step earlier, seven or fewer cells are allowed. Thus, the

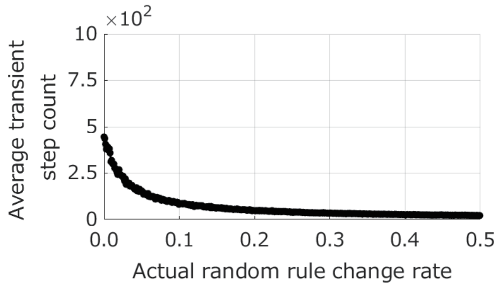


Fig. 5. Transient step count when rule change rate is randomly implemented.

Table 2
Relationship between changes in resource amount and Rule 1-0 at R = 0.4.

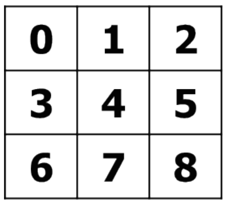
1 step ago		2 steps ago		
Number	Resource depletion	Number	Resource depletion	Total resources in 9 neighboring cells
3	1.2	0	0.0	7.8
		1	0.2	7.6
		2	0.4	7.4
		3	0.6	7.2
		4	0.8	7.0
		5	1.0	6.8
		6	1.2	6.6
2	0.8	0	0.0	8.2
		1	0.2	8.0
		2	0.4	7.8
		3	0.6	7.6
		4	0.8	7.4
		5	1.0	7.2
		6	1.2	7.0
		7	1.4	6.8
1	0.4	0	0.0	8.6
		1	0.2	8.4
		2	0.4	8.2
		3	0.6	8.0
		4	0.8	7.8
		5	1.0	7.6
		6	1.2	7.4
		7	1.4	7.2
		8	1.6	7.0
0	0.0	0	0.0	9.0
		1	0.2	8.8
		2	0.4	8.6
		3	0.6	8.4
		4	0.8	8.2
		5	1.0	8.0
		6	1.2	7.8
		7	1.4	7.6
		8	1.6	7.4
		9	1.8	7.2

state changes occur such that the number of cells born, including past conditions, does not become excessive. In particular, rules are adopted on the basis of the previous overcrowding state through resource feedback, generating complex chaotic dynamics. Furthermore, for $R \geq 0.5$, greater resource depletion at birth indicates that even more complex rule changes occur, reflecting past overcrowding states.

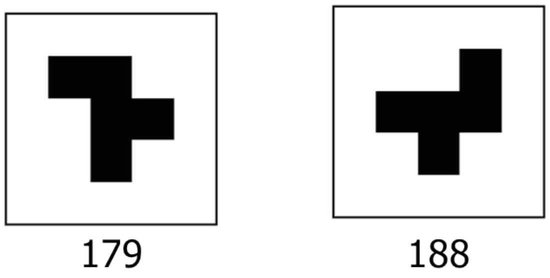
3.2.4 Case where all 512 3 × 3 patterns are placed at center of cell space

To analyze more specific characteristics, we analyzed all 512 possible 3 × 3 patterns that are likely to emerge in random initial configurations. Each pattern was encoded as bits, as shown in Fig. 6(a), converted to decimal form, and assigned a number. Figure 6(b) shows an example of this numbering system. Some patterns produce identical results when rotated or flipped, and thus the number of patterns tested was 102. We conducted experiments to measure the transient step count for these 102 patterns when placed at the center of torus cell spaces of sizes 51 × 51, 101 × 101, and 151 × 151. Figure 7 shows the results for the 51 × 51 cell space. In Fig. 7, the horizontal axis denotes the pattern number, and the vertical axis represents the transient step count on a logarithmic scale. Figure 7 identifies several patterns that exhibit high transient step counts. Detailed analysis revealed recurring patterns among these, indicating that only a limited subset of patterns have a high transient step count. Therefore, we extracted and arranged the patterns with increasing transient step counts, starting from the smallest pattern number, to identify three main systems, namely, (a) r-pentomino system, (b) Π-heptomino system, and (c) t-tetromino system (shown in Fig. 8).

Pattern number 122 in Fig. 8(a) corresponds to r-pentomino, which is a well-known methuselah in the conventional GoL. When simulated in a sufficiently large space, it exhibits 1,103 transient steps. Figure 9(a) shows the transient step count for various R values and cell space sizes for the r-pentomino. The horizontal axis represents R, and the vertical axis represents the transient step count on a logarithmic scale. Figure 9(a) indicates that the transient step count increases significantly at R = 0.5, 0.6, 0.9, and 1.0, with the effect becoming more pronounced as the cell space expands. In particular, for R = 0.9 and a 151 × 151 cell space, the transient step count surpasses 600 million. Additionally, patterns 87 and 111 produce identical results when considering the transient step count.



(a)



(b)

Fig. 6. Pattern number (layout) and specific examples.

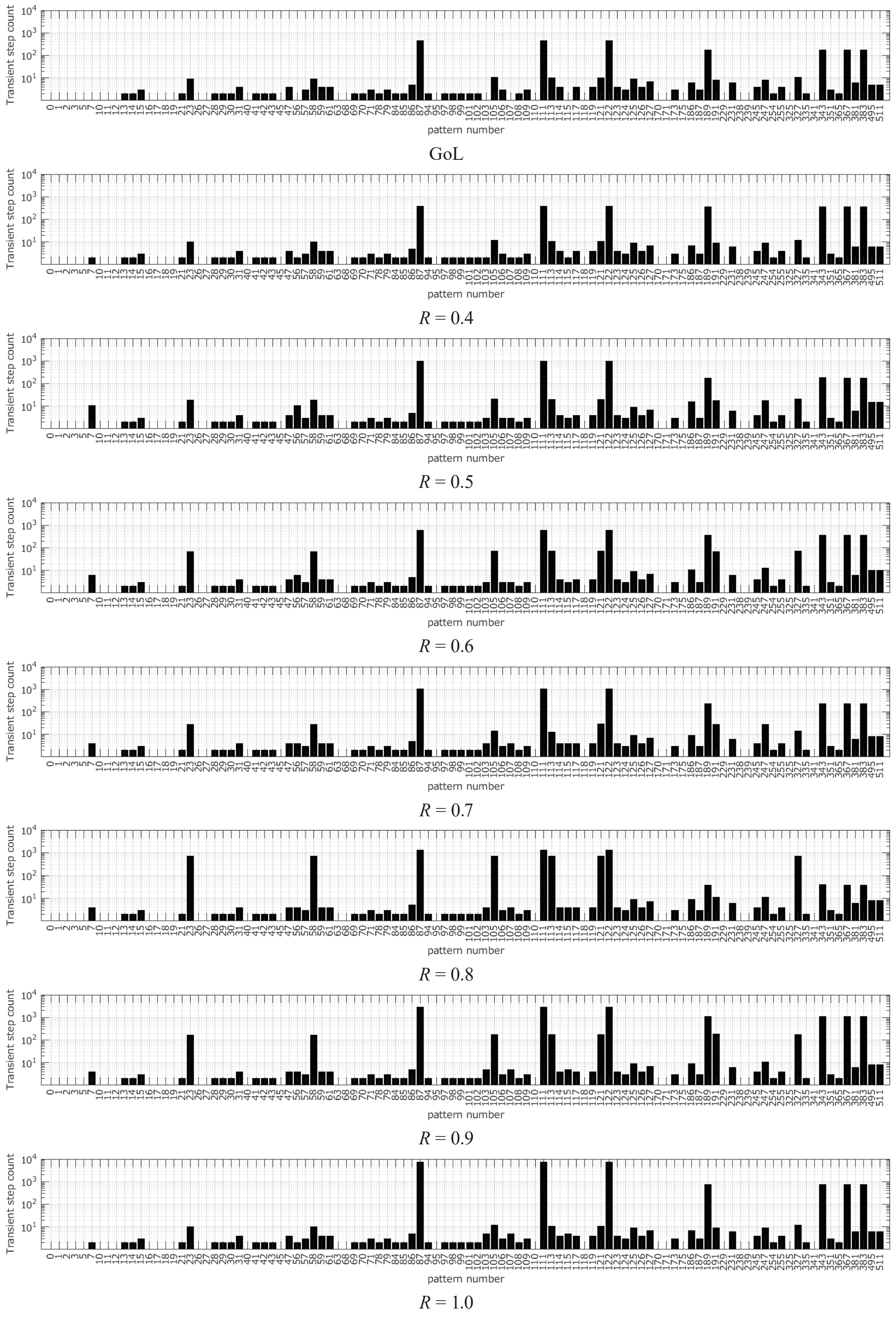
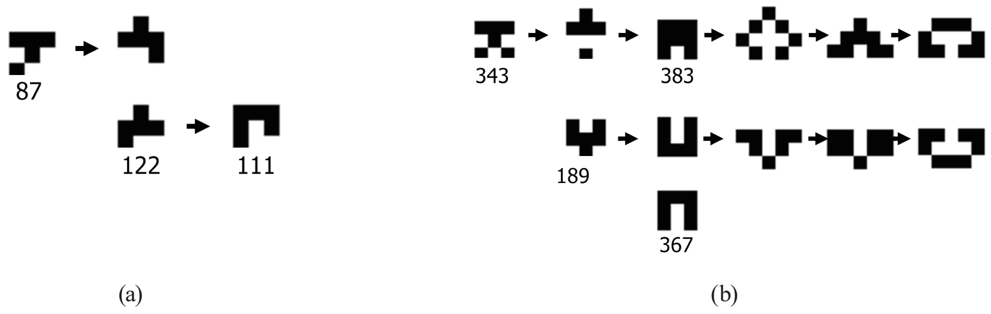


Fig. 7. Transient step count in 102 51 × 51 patterns.



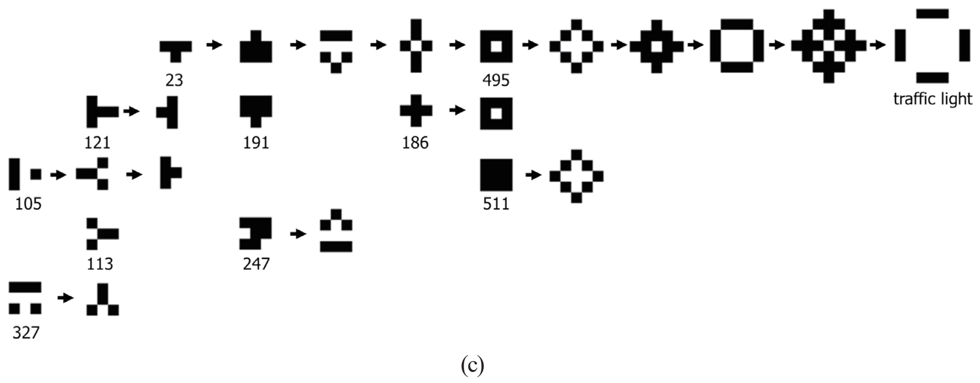
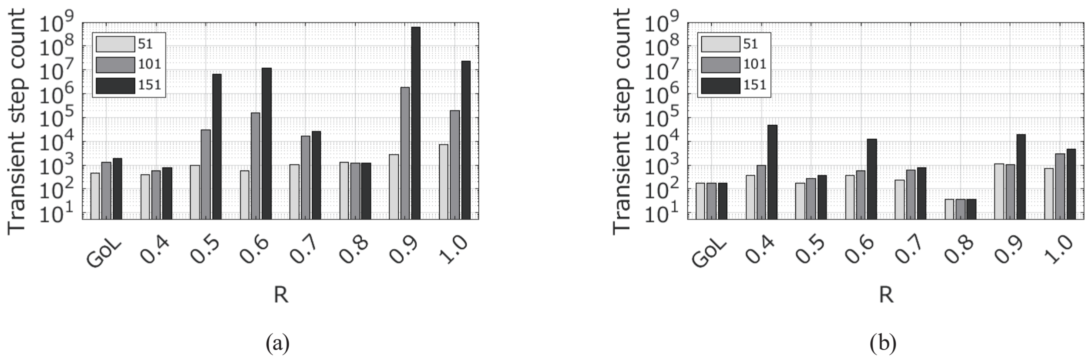


Fig. 8. Methuselah patterns: (a) r-pentomino system, (b) Π-heptomino system, and (c) t-tetromino system.



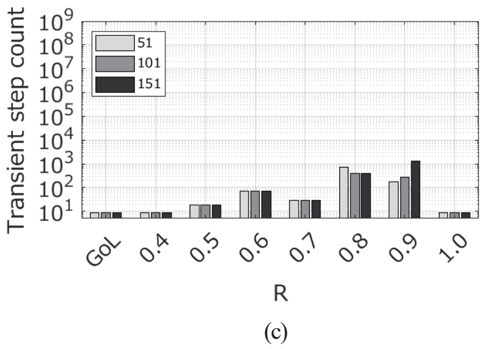


Fig. 9. Transient step count for patterns 122, 367 and 23: (a) pattern 122 (r-pentomino), (b) pattern 367 (Π-heptomino), and (c) pattern 23 (t-tetromino).

Pattern number 367 in Fig. 8(b) corresponds to the Π-heptomino, which is another methuselah in the conventional GoL, exhibiting 173 transient steps. Figure 9(b) presents the transient step counts for the Π-heptomino for various R values and cell space sizes. The horizontal axis represents R, and the vertical axis represents the transient step count on a logarithmic scale. As shown in Fig. 9(b), similar to the r-pentomino, R = 0.6, 0.9, and 1.0 yield large transient step counts. However, contrary to the r-pentomino, R = 0.4 has a tendency for a large transient step count, and R = 0.5 has a tendency for a smaller transient step count. Furthermore, patterns 189, 343, and 383 yield identical results when the transient step count is considered.

Pattern 23 in Fig. 8(c) corresponds to a t-tetromino. In the conventional GoL, a t-tetromino evolves into an oscillator comprising four blinkers, called a traffic light, after ten steps. Figure 9(c) presents the transient step counts for each value of R and for each cell space size when using the t-tetromino. The horizontal axis represents R, and the vertical axis represents the transient step count on a logarithmic scale. As shown in Fig. 9(c), the transient step count tends to increase when R = 0.8, which contrasts with the lack of notable changes observed for the r-pentomino and Π-heptomino cases. In addition, patterns 105, 113, 121, and 327 produced identical outcomes when the transient step count was considered. However, patterns 186, 191, 247, 495, and 511 yielded results that diverged from those of pattern 23. Patterns 186, 495, and 511 did not activate Rule 1-0 and therefore behaved similarly to the conventional GoL. Pattern 247 resembled pattern 23 when R = 0.5 and 0.7, whereas in other cases, it behaved like the conventional GoL. Pattern 191 matched pattern 23 at R = 0.5, 0.6, and 0.7, but aligned with the conventional GoL at R = 0.8 and 1.0. At R = 0.9, it differed from pattern 23 by 15 steps. Moreover, in the t-tetromino system, additional patterns, such as those displayed in Fig. 10 appeared depending on the parameter values, specifically within the square frames in the figure. All of these are mobile patterns, and Fig. 10(c) in particular advances in a rocket-like form, occupying a larger area than the glider, which is a common mobile pattern in the conventional GoL. Figure 10 represents only one example, and although not all parameters and patterns were tested, other large-area and long-period oscillators were also observed. This suggests that a broader set of patterns is generated than in the conventional GoL. Area patterns larger than in the conventional GoL are especially likely to interact with other patterns, which increases pattern variety and raises the transient step count.

Collectively, these findings imply that even patterns defined within a 3 × 3 region show substantial parameter-dependent differences, and such differences become even more significant when using more varied configurations, such as random initial configurations. This indicates that the Sugarscape GoL can produce a wide range of chaotic dynamics.

4. Discussion

In this study, we investigated the characteristics of chaotic dynamics in the Sugarscape GoL using transient step counts. The findings revealed that the transient step counts were substantially higher than those of the conventional GoL and increased exponentially with expanding space size. These results indicate that chaotic dynamics persist for extended periods, suggesting that this model is suitable as a reservoir in reservoir computing. The effectiveness of using chaotic

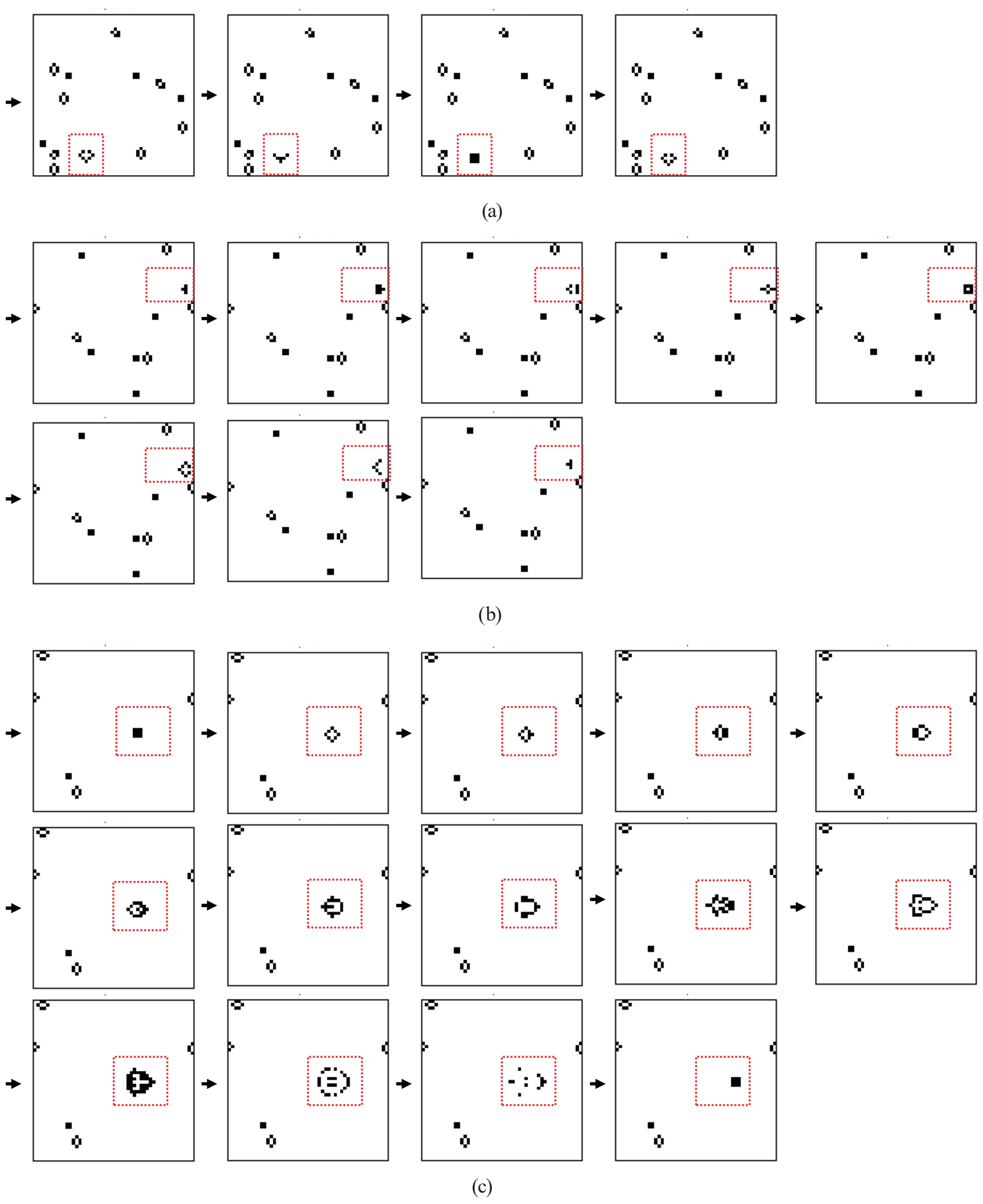


Fig. 10. (Color online) Example of patterns observed in pattern 23 (t-tetromino). (a) $R = 0.5$, $C = 7.6$, (b) $R = 0.6$, $C = 7.1$, and (c) $R = 0.7$, $C = 6.4$.

dynamics via asynchronous cellular automata in reservoir computing has been demonstrated,(17) and Sugarscape GoL is also likely to be effective. The chaotic dynamics were also shown to differ depending on parameter values. For instance, although R = 0.5 and R = 1.0 produced similar average transient step counts in the random initial configurations, they differed significantly for pattern 367, implying distinct chaotic behaviors. Furthermore, there is a connection between $R = 0.9$, which frequently results in large transient step counts for 3×3 patterns, and $R = 0.7$, which generally yields small transient step counts for 3×3 patterns and also produces small values in the random initial configurations. However, $R = 0.4$ does not generally generate significantly large transient step counts for 3×3 patterns, yet in the random initial configuration, it produced the second largest transient step count after $R = 0.9$. In reservoir computing, the properties of the reservoir depend on the target task. Although design guidelines exist, trial and error is often necessary.(20) Sugarscape GoL can generate a variety of chaotic dynamics depending on its parameters, suggesting a wide range of potential applications. Sugarscape GoL generates a greater variety of spatiotemporal patterns within the same array size than does conventional GoL. This property makes it effective for reservoir computing using GoL, for example, enabling reductions in reservoir size. Therefore, understanding the chaotic dynamics associated with each parameter requires evaluating them with measures such as 1/f fluctuations,(21) entropy,(22) and spectral analysis.(23) Recently, various physical reservoirs other than cellular automata have been proposed. Comparing physical reservoirs is not straightforward, but a detailed examination of different physical reservoirs using, for example, spatial metrics,(24) may clarify mechanisms, such as how chaotic dynamics in reservoirs affect accuracy. Accuracy may be improved by ensemble learning that combines multiple reservoirs with distinct chaotic dynamics behaviors.

Recent work has highlighted the downsizing of learning models to enable operation in local environments where confidentiality is required. BitNet b1.58,(25) for example, reduces memory requirements and increases computational speed by quantizing parameters. In the Sugarscape GoL model analyzed in this study, resources vary from 0 to 1 in increments of 0.1, restricting the possible value configurations and enabling a broad variety of chaotic dynamics while keeping memory usage low. Additionally, we measured the execution time for 100 runs with random initial configurations (torus boundary) in a 100×100 cell space, with 10,000 steps, and calculated the average. The execution time was 1.54 times longer for Sugarscape GoL than for conventional GoL. Although resource-related calculations not focused in the traditional GoL are added, the computational cost does not increase significantly are added. Reservoir computing also reduces parameter counts substantially compared with conventional deep learning. This reduction allows learning to proceed quickly, creating a highly real-time method that adapts well to rapidly changing environments. This method may also be advantageous for edge AI as it is desirable to process data from numerous sensors in real time. Edge AI demand is expected to increase as sensor technologies continue to advance and become smaller, as edge systems must handle large volumes of a variety of data in real time despite limited computing resources. For instance, edge AI may help address a wide range of social and economic issues, including disaster prevention for increasingly frequent heavy rainfall events and the development of autonomous driving, which is regarded as a next-generation service. Given that the GoL is a model capable of simulating phenomena such as the evolution of life, Sugarscape GoL may have a high affinity with life-science-related fields (e.g., biomedical engineering). Reservoir computing is also applied in these fields.(26,27) Furthermore, since acquiring large amounts of data is often difficult, reservoir computing, which can be utilized with less data compared with deep learning, is effective. Moreover, lightweight machine learning methods such as reservoir computing are power-efficient and thus may contribute to an energy-saving electronic, information, and communication society that minimizes environmental impact in terms of energy.

5. Conclusions

In this study, we investigated how the chaotic dynamics of the Sugarscape GoL differ from those of the conventional GoL by examining transient step counts in the context of reservoir computing. The results showed that transient step counts were substantially higher than in the conventional GoL, resulting in long-lasting chaotic dynamics. We also observed that the characteristics of the resulting patterns varied in accordance with parameter values. These outcomes indicate that the Sugarscape GoL can generate diverse forms of chaotic dynamics over long durations, making it suitable for use as a reservoir. In this study, we focused on the two-dimensional cellular automaton that is GoL, but it is likely that other reservoirs can be constructed by extending the Sugarscape concept to other cellular automaton types, including the one-dimensional elementary cellular automaton.

In future work, we will implement reservoir computing based on the Sugarscape GoL analyzed in this study, evaluate the link between chaotic dynamics and computational performance, and apply it to real-world datasets. This can contribute to improving convenience and productivity in an IoT society in which all devices are interconnected.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP23K11282. We express our appreciation here.

References

1 A. Katal, S. Dahiya, and T. Choudhury: Cluster Comput. **26** (2023) 1845. https://doi.org/10.1007/s10586-022-03713-0
2 M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann: Neuromorph. Comput. Eng. **2** (2022) 032002. https://doi.org/10.1088/2634-4386/ac7db7
3 Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief: IEEE Commun. Surv. Tutorials **22** (2020) 2167. https://doi.org/10.1109/COMST.2020.3007787
4 G. Tanaka, C. Gallicchio, A. Micheli, J. P. Ortega, and A. Hirose: IEEE Trans. Neural Networks Learn. Syst. **33** (2022) 2571. https://doi.org/10.1109/TNNLS.2022.3172586
5 G. Tanaka: Jpn. Soc. Artif. Intell. **40** (2025) 353 (in Japanese). https://doi.org/10.11517/jjsai.40.3_353
6 H. Jaeger and H. Haas: Science. **304** (2004) 78. https://doi.org/10.1126/science.1091277
7 W. Maass, T. Natschläger, and H. Markram: Neural Comput. **14** (2002) 2531. https://doi.org/10.1162/089976602760407955
8 G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, and A.Hirose: Neural Networks **115** (2019) 100. https://doi.org/10.1016/j.neunet.2019.03.005
9 K. Nakajima, H. Hauser, T. Li, and R. Pfeifer: Sci. Rep. **5** (2015) 10487. https://doi.org/10.1038/srep10487
10 T. Kanao, H. Suto, K. Mizushima, H. Goto, T. Tanamoto, and T. Nagasawa: Phys. Rev. Appl. **12** (2019) 024052. https://doi.org/10.1103/PhysRevApplied.12.024052

11 G. Van der Sande, D. Brunner, and M. C. Soriano: Nanophotonics **6** (2017) 561. https://doi.org/10.1515/nanoph-2016-0132
12 T. Misawa, K. Yamashita, Y. Inazumi, and K. Okino: The Inst. Electron. Inf. Commun. Eng. **103** (2020) 352 (in Japanese). https://doi.org/10.14923/transinfj.2019jdl8009
13 T. Misawa, K. Yamashita, Y. Inazumi, and K. Okino: IEEJ Trans. Electron. Inf. Syst. **143** (2023) 352 (in Japanese). https://doi.org/10.1541/ieejeiss.143.101
14 W. Poundstone: The recursive universe: Cosmic complexity and the limits of scientific knowledge (Courier Corporation 2013).
15 M. Ushio, K. Watanabe, Y. Fukuda, Y. Tokudome, and K. Nakajima: R. Soc. Open Sci. **10** (2023) 221614. https://doi.org/10.1098/rsos.221614
16 N. Babson and C. Teuscher: Complex Syst. **28** (2019) 433. https://doi.org/10.25088/ComplexSystems.28.4.433
17 D. Uragami and Y. P. Gunji: Complex Syst. **31** (2022) 103. https://doi.org/10.25088/ComplexSystems.31.1.103
18 S. Ninagawa, M. Yoneda, and S. Hirose: Trans. Jpn. Soc. Artif. Intell. **16** (2001) 164 (in Japanese). https://doi.org/10.1527/TJSAI.16.164
19 J. M. Epstein and R. Axtell: Growing artificial societies: social science from the bottom up (Brookings Institution Press 1996).
20 G. Tanaka: J. Inst. Electron. Inf. Commun. Eng. **106** (2023) 521 (in Japanese).
21 S. Ninagawa, M. Yoneda, and S. Hirose: Physica D **118** (1998) 49. https://doi.org/10.1016/S0167-2789(98)00025-6
22 M. D'amico, G. Manzini, and L. Margara: Theor. Comput. Sci. **290** (2003) 1629. https://doi.org/10.1016/S0304-3975(02)00071-3
23 S. Ninagawa: Complex Syst. **17** (2008) 399. https://doi.org/10.25088/ComplexSystems.17.4.399
24 J. Love, R. Msiska, J. Mulkers, G. Bourianoff, J. Leliaert, and K. Everschor: Phys. Rev. Appl. **20** (2023) 044057. https://doi.org/10.1103/PhysRevApplied.20.044057
25 S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, and F. Wei: eprint arXiv:2402.17764 (2024). https://doi.org/10.48550/arXiv.2402.17764
26 U. Masayuki and K. Nakajima: Jpn. J. Ecology **74** (2024) 229 (in Japanese).
27 L. Bozhkov, P. Koprinkova-Hristova, and P. Georgieva: Neurocomputing **231** (2017) 28. https://doi.org/10.1016/j.neucom.2016.03.108