Doctoral Dissertation

博士論文

Study on Amoeba-inspired Electronic Computing System for Solving Optimization Problems

（生物粘菌アメーバに倣った最適化問題を解く電子計算システムに関する研究）

Graduate School of Information Science and Technology,

Hokkaido University

北海道大学大学院情報科学研究科

Kenta SAITO

斉藤　健太

HOKKAIDO
UNIVERSITY

A dissertation submitted in partial fulfillment
of the requirements for the degree of Doctor of
Philosophy (Engineering)
in Hokkaido University, February 2021.
Dissertation Supervisor
Professor Seiya KASAI

Acknowledgements

This thesis describes my research work carried out at Research Center for Integrated Quantum Electronics (RCIQE) and Graduate School of Information and Science Technology, Hokkaido University.

First of all, I would like to thank my supervisor, Professor Seiya Kasai, for his helpful suggestions and ideas. I learned a lot of things from him such as scientific thought, writing technique, presentation, and so on. His help made my ability about science improve.

I am deeply grateful to Dr. Masashi Aono for drawing me into this study. He is an extremely attractive person and has passion. They exceedingly influenced my life and thought.

I am grateful for helpful advices and discussions to supervisors of RCIQE, Professor Eiichi Sano Professor Tamotsu Hashizume, Professor Junichi Motohisa, Professor Masayuki Ikebe, Associate Professor Taketomo Sato, Associate Professor Masamichi Akazawa, Associate Professor Shinjiro Hara, and Associate Professor Katsuhiro Tomioka.

I greatly appreciate valuable discussion with all the members of Kasai's laboratory, Dr. Masaki Sato, Dr. Yin Xiang, Dr. Ryota Kuroda, Mr. Kento Shi-

rata, Mr. Ryo Wakamiya, Mr. Yuki Inden, Mr. Syoma Okamoto, Mr. Kentaro Sasaki, Mr. Koki Abe, Mr. Katsuma Shimizu, Mr. Kazuki Inada, Mr. Renpeng Lu, Mr. Naoki Suefuji, Mr. Koichi Tajima, Mr. Syunsuke Saito, Ms. Syu Onuma, Mr. Shintaro Mizuno, Ms. Yoko Shigematsu, and Mr. Tomoya Fujiwara.

I am greatful for encouragement to my collegues, Mr. Syota kaneki, Mr. Syota Toiya, Mr. Ryoma Horiguchi, Mr. Keisuke Ito, Mr. Satoru Matsumoto, Mr. Dai Hasegawa, Mr. Naoshige Yokota, Mr. Taito Hasezaki, Mr. Syota Hiramatsu, Mr. Kohei Chiba, and Mr. Akinobu Yoshida.

I would like to express my gratitude to technical staffs and secretaries, Mr. Kenji Takada, Mr. Kiyotake Nagakura, Ms. Mizuho Tanaka, and Ms. Chieko Akiyama for their supports.

Finally, I would like to thank my parents, Masataka Saito and Yasuko Saito. They always cheered me up under any circumstances. If without their help, this work would not have completed.

February 2021 at RCIQE

Kenta Saito

List of Tables

List of Figures

Contents

Chapter 1

Introduction

1.1 Background

The decline in the working population due to the decline in the birthrate is an issue that must be resolved. In order to solve this problem, optimization of various resources, such as personnel assignment and distribution channels, is needed. These can be formulated as optimization problems. The optimization problem can be found anywhere: for example, traffic flow optimization enables to resolve the shortage of workers in carriers due to the increase in demands under the condition of the coronavirus pandemic [1]. In addition, many applications for resolving problems in the real world have been proposed: prime factorization [2], item-listing optimization [3], path optimization of automated-guided vehicle [4], shift scheduling of nurses [5], proposing evacuation route for Tsunami [6], job shop scheduling [7] radio resource allocation optimization in wireless communication systems [8], traffic signal optimization [9], and vehicle-routing optimization [10]. According to the wide range of applications in the optimization problem, it archieves the goals set out in Society 5.0. The objective of optimization problems is to find a best combination of variables that maximizes an objective function while the combination of variables satisfies constraints given by the problem to be solved. Generally, we cannot find

an optimal solution to optimization problems mathematically; therefore, we should search for the solution through trial and error. However, the computational time in a conventional computer with a general-purpose processor that operates sequentially becomes impracticable when the problem size is increased, because the number of solution candidates exponentially increases, which is often called "combinatorial explosion." The traveling salesman problem (TSP) is one of the typical examples as the optimization problems, and the number of solution candidates in the TSP is given by $(N-1)!/2$, where N is the number of cities [11, 12]. In addition, although recent progress in a complementary metal oxide semiconductor (CMOS) improves the performance of general-purpose processors based on the Moore's law and the Dennard scaling, we have faced the physical limitations of the miniaturization. We would not improve their performance [13–16]. Therefore, a domain-specific architecture (DSA) has been developed to realize a high-performance computing system [17, 18].

An Ising machine is one of the DSAs and candidates dedicated to solving optimization problems, which is inspired from the Ising model in statistical physics that describes spin alignments of magnetic body [19, 20]. However, Ising machines suffer from problem mapping because their connectivity is sparse due to the machine structure: it needs pre-processing to map the problem onto them [21–27]. They have to solve an optimization problem to map the problem to the Ising machine: Ising machines solve the problem twice. If Ising machines whose connectivity is sparse, the complete graph is artificially achieved by using redundant variables. However, the performance degrades compared to the system whose connectivity is complete [28]. In addition, the redundant variables need consistency: the states of redundant variables should be the same value; therefore, if the values of redundant variables when they converge at the final state are different, post-processing is needed to fix the states. Moreover, Ising machines have to tune a hyperparameter that is a coefficient of penalty terms in the Ising model, due to the Ising model in order to

obtain a legal solution that satisfies the constraints of optimization problems, before starting the solution search. As a result, if the parameter tuning is not enough to be performed, the system sometimes converges at an illegal solution [29]. Ising machines require pre- and post-processing to solve optimization problems, and such processings degrade its performance.

In contrast to the Ising machine, Aono et al. developed an amoeba-based computing system in which an amoeboid organism, *Physarum Polycephalum*, searches a solution while avoiding light illumination from a feedback system based on the problem to be solved. The amoeboid organism has high intelligence without a brain: it solves a maze and optimizes a transportation network by utilizing its body shape [30, 31]. It was also reported that the amoeboid organism could solve the TSP in only linear time growth as increasing the number of cities [32, 33]. The system exhibits interesting performance but it is not feasible for solving optimization problems as an alternative to a conventional computer because the moving rate of the amoeboid organism is nearly 1 cm/h [34]. Therefore, Kasai et al. developed an electronic computing system, called "electronic amoeba," by utilizing the current dynamics in a capacitor network, inspired from the amoeba-based computing system. They demonstrated the electronic amoeba could solve a simple optimization problem, Negative OR (NOR), that can be solved by a conventional computer. However, it has not been demonstrated and evaluated that the system can solve a complicated problem. To apply the electronic amoeba to practical applications in the real world as an optimization problems solver, the ability for the electronic amoeba to solve more complicated problems, such as the satisfiability problem (SAT), maximum cut problem (Max-cut), and also TSP, should be demonstrated. In addition, evaluation and improvement of its performance to these problems are needed to reveal the advantages of the electronic amoeba to Ising machines in terms of the solution search.

1.2 Objectives of this work

Based on the above background, the objectives of this thesis include to develop the electronic amoeba that solves the SAT, TSP, and Max-cut, to evaluate and to improve its performance. To change the problems to be solved by the electronic amoeba, a feedback circuit that implements interactions between variables is changed based on a specified rule for the amoeba-inspired computing system. Then, I develop an analog-digital hybrid electronic amoeba for solving the SAT and an all-analog electronic amoeba for solving the TSP and Max-cut. The performance of systems in the physically implemented electronic amoeba and simulated electronic amoeba using a circuit simulator to those problems is evaluated. From this study, it is found that its performance exceeds the performance of a local search algorithm implemented on a conventional computer. The quality of solutions is improved by a time delay, which is insipred from the dynamics of the physically implemented electronic amoeba.

1.3 Synopsis of this thesis

This thesis consists of seven chapters, including Introduction in this chapter.

Chapter 2 describes an optimization problem and the amoeba-based computing and -inspired electronic computing system.

Chapter 3 explains related works dedicated to solving optimization problems and points out the weakness points in Ising machines against the electronic amoeba.

Chapter 4 describes an analog-digital hybrid electronic amoeba for solving the SAT. We evaluate the influence of an error property on its solution-searching performance to the SAT. I pointed out that the asymmetric dynamics in the system due to charging-discharging time of the capacitors in a capacitor network plays an important role in terms of solution searching. It provided the system with the robustness

to the error property and improved its performance exponentially, compared to the original model.

Chapter 5 proposes and demonstrates the electronic amoeba integrating a resistance crossbar circuit that implements the problem to be solved as resistances. The crossbar circuit performs product-sum operations and thresholding. Maximum cut and traveling salesman problem were mapped onto the crossbar circuit and solved by the electronic amoeba on a circuit simulator and breadboard. Once the electronic amoeba is implemented by physical circuits, it is suggested that its performance will exceed a conventional computer by the circuit simulator.

Chapter 6 describes the influence of delayed feedback in the amoeba-inspired computing system on its solution-searching performance, which is inspired from the variables oscillation of the physically implemented electronic amoeba due to parasitic capacitances in analog circuits. The quality of solutions was improved by imposing an appropriate delay length to the system. By scheduling the magnitude of delay, further improvement was achieved.

Chapter 7 summarizes and concludes this thesis.

Chapter 2

Concept of amoeba-inspired computing system

2.1 Introduction

Optimization problems can be widely applied to practical application in the real world, and to quickly find a solution to optimization problems resolves social challenges: for example, we can resolve shortage of workers in Japan by optimizing vehicle routing [1–10]. However, it is hard for a conventional computer with a general-purpose processor to solve the optimization problems since the combinatorial explosion occurs when the problem size is increased. Special-purpose systems dedicated to solving optimization problems are expected to solve such problems at low power consumption and high speed, although the systems cannot be applied to general purpose problems, such as a problem dealt by a central processing unit (CPU). An amoeba-inspired electronic computing system, inspired from solution-searching behavior of an amoeboid organism, has the ability to find a solution to optimization problems [30–33, 35]. This chapter describes optimization problems, an amoeba-based computing system, and an amoeba-inspired electronic computing system.

2.2 Combinatorial optimization problems and computational complexity

2.2.1 Time complexity

Time complexity expresses the degree of the computational difficulty and is usually used in the research area of the computer science. The time complexity can be indirectly measured through algorithm: for example, the maximum number of steps to finish a computation for the input length, n, represents as a function of n. Because the time complexity is not quantity, one cannot compare the magnitude correlation between the time complexities but can compare them based on the increasing rate of function values; therefore, the order expression is used, which represents the upper bound of the function: for example,

$$x^2 + x + 1 = O(x^2). \tag{2.1}$$

The order of the following express is weak evaluation but correct equation, compared to Eq. 2.1:

$$x^2 + x + 1 = O(x^3). \tag{2.2}$$

The increasing rate in Eq. 2.2 is larger than that in Eq.2.1.

2.2.2 Computational complexity

Class P and NP are a set of judgement problems: given the input to a problem, the algorithm can judge yes-or-no to the problem in a polynomial time. When the time complexity of an algorithm to the input size, n, is represented by a polynomial express $O(n^k)$, where k is a constant depending on a problem. In this case, a solution to this problem can be found in a polynomial time. Such a problem belongs to the class P and a conventional computer can solve the problems belonging to class P in

a realistic time because the problem size is increased slowly as increasing the input size. On the other hand, when the order of the time complexity is represented by an exponential cost such as $O(k^n)$, the computational cost is increased exponentially; therefore, the computer cannot solve it in a realistic time. Problems belonging to class NP are defined as follows: given a solution that is "yes" to a problem whose input size n, the problem belongs to the class NP if the solution can be judged as "yes" in the order of a polynomial time, $O(n^k)$. Therefore, considering the definitions of the class P and NP, we can easily understand $P \in NP$, where P and NP are a set of the class P and NP, respectively.

Next, I explain a definition of the class NP-complete. We call as "reduction" transforming a problem x of a set of decision problem, $\boldsymbol{X}$, to a different problem $y(y \in \boldsymbol{Y})$ with same solution. Here, when $h(x)$ satisfies $x \in X \Leftrightarrow h(x) \in \boldsymbol{Y}$, $h(x)$ is reduction from $\boldsymbol{X}$ to $\boldsymbol{Y}$. If a function h can be calculated in a polynomial time, h is called polynomial time reduction. If such a function exists, $\boldsymbol{X}$ can be reduced by $\boldsymbol{Y}$. We describe the reduction as $X \leq_m^p Y$. When a decision problem satisfies following Eq. 2.3, the problem is called NP-hard

$$\forall L \in NP[L \leq_m^p X]. \tag{2.3}$$

When it also satisfies Eq. 2.4, it is called NP-complete.

$$X \in NP. \tag{2.4}$$

NP-complete is a set of problems themself belonging to NP and being more difficult than all NP problems. If one can solve problems belonging to NP-complete in a polynomial time, we can solve all problems belonging the class NP in a polynomial time. From above definitions, these relationships can be drawn as Fig. 2.1, where we assume $P \neq NP$. Nobody knows an algorithm that finds a solution to problems

of NP-complete in polynomial time, and most of mathematician predicts $P \neq NP$.

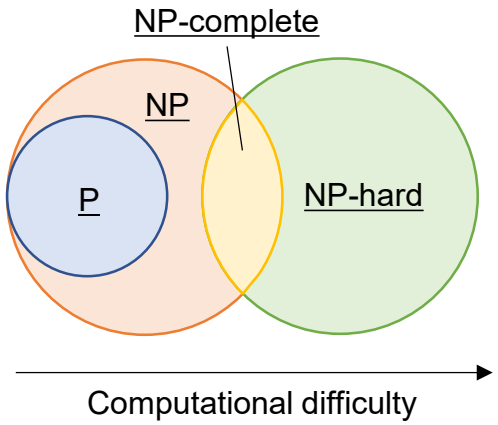

Figure 2.1: Computational complexity in case of $P \neq NP$.

2.2.3 Combinatorial optimization problems

Combinatorial optimization problems are defined as follows: finding an optimal combination of variables in the vast number of solution candidates, where each variable takes 0 or 1. Practical applications in the real world, such as artificial intelligence, information security, design support of large-scale integration, and so on, can be transformed to optimization problems. Examples of such problems include the satisfiability problem (SAT), traveling salesman problem (TSP), and maximum cut problem (Max-cut), which are described in the next subsection. Many of combinatorial optimization problems belong to the NP-complete or -hard; therefore, it is known that the combinatorial explosion occurs as increasing the problem size because the combination increases exponentially as increasing the problem size as shown in Tab. 2.1. Nobody knows an exact algorithm that finds an optimal solution in polynomial time. To quickly solve the problems, algorithms for searching for a solution have been developed.

Table 2.1: Combinatorial explosion

Number of variables (N)	Number of combinations (2^N)
10	1,024
20	1,048,576
30	1,073,741,824
40	1,099,511,627,776
50	1,125,899,906,842,624

2.2.4 Satisfiability problem

The SAT is a judgement problem stated as follows: finding a combination of variables such that the combination makes an objective function true. The SAT belongs to NP-complete; it can be converted from any problems belonging to the class NP. The objective function, $F(\boldsymbol{x})$, consists of a conjunctive normal formula (CNF), where $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_N)$ $(x_i \in 0, 1)$:

$$F(\boldsymbol{x}) = (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2), \tag{2.5}$$

where each logical sum of variables is called clause. In Eq. 2.5, the solution is $(x_1, x_2) = (1, 1)$. When the number of variables in each clause is k at most, the problem is called k-SAT. When $k \geq 3$, the k-SAT belongs to the NP-complete. When $k < 3$, a solution to k-SAT can be found in a polynomial time by using unit propagation. The SAT is a first proven problem as it is the NP-complete [36, 37]. Because the SAT belongs to the NP-complete and is represented by the simple formula, algorithms for solving the SAT have been widely studied [38–41]. Empirically, it is known that the difficulty in finding a solution to the SAT becomes maximum when the ratio of the number of variables to the number of clauses is near to 4.3 [42, 43]. This is because constraints are too strong to exist a solution (i.e., the number of clauses is large) and too weak to exist a large number of solutions (the number of variables is small); therefore, the median value 4.3 makes the difficulty maximize.

Algorithms for searching for a solution to the SAT are divided into complete and incomplete algorithms; complete algorithms judge that there is a solution or not; incomplete algorithms, on the other hand, find a solution quickly but cannot judge that there is not a solution. In practical, finding a solution quickly is important to apply the SAT to practical applications. So far, WalkSAT has been known as the fastest algorithm in incomplete algorithms [40]. A basic method of WalkSAT is simple. We randomly assign 0 or 1 to variables as initial values. We randomly select a clause that is false and randomly flip the variable in the selected clause to make the clause true. These steps are repeated until finding a solution. Flipping a variable in the false clause ensures to search the neighborhood of present variable vector space. However, if there does not exist a solution near to variable vector space, WalkSAT forever repeats a variable flipping and never find a solution, where this condition is called trapping a local optimum. To escape from local optima, stochastic flipping mechanism is usually introduced, which randomly selects a variable from all variables and flip it.

2.2.5 Traveling salesman problem

The TSP is one of the NP-hard problems and stated as follows: finding a minimum route such that the salesman should visit all cities only once and return to home city. An example of the TSP is shown in Fig. 2.2. The number of solution candidates increases in $(N-1)!/2$. When we define a intercity distance from city A to B as $R_{A,B}$, the symmetric TSP assumes $R_{A,B} = R_{B,A}$; on the other hand, the asymmetric TSP assumes $R_{A,B} \neq R_{B,A}$. There exist a lot of applications in the TSP, such as the vehicle routing problem (VRP). Finding a solution to the VRP is an urgent issue because the demand for optimizing logistics grows due to COVID-19.

An algorithm that quickly finds a solution to the TSP is the nearest neighborhood method (NN). The NN searches a nearest city at the current city and moves to the

nearest city. This process is made until all the cities are visited. Figure 2.3 shows an example solution to the TSP by using random sampling and the NN, where the random sampling decides the next visited city randomly. Comparing Figs. 2.3(b) and 2.3(c), the solution obtained by the NN is superior to the random sampling one, although the NN is a simple algorithm.

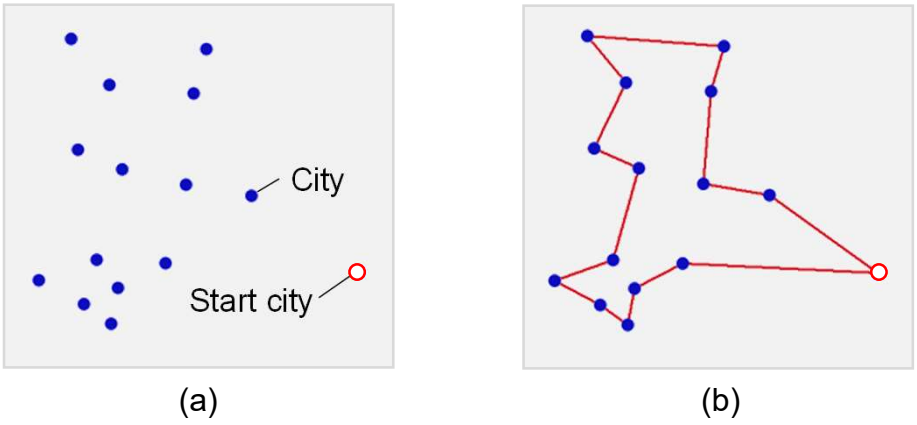

Figure 2.2: (a) Problem example of TSP and (b) optimal solution of TSP

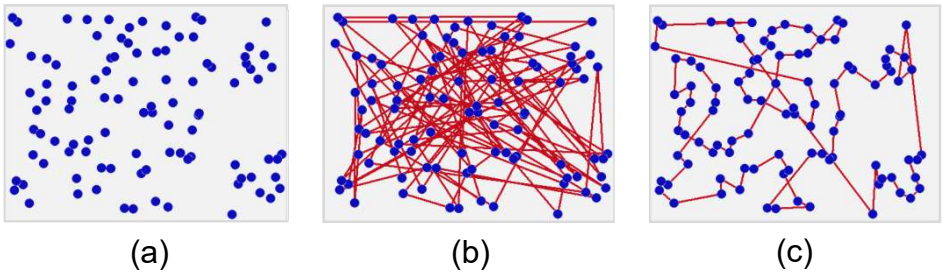

Figure 2.3: (a) 100-city instance and solution obtained by (b) random sampling and (c) nearest neighborhood method. Total route of (b) is 340696 and that of (c) is 65646.

2.2.6 Maximum cut problem

The Max-cut also belongs to class NP-hard, such as the TSP. The Max-cut is stated as follows: given a weighted graph, finding subsets of vertexes V, U_1 and U_2 ($U_1, U_2 \in V$, $U_1 \cap U_2 = \emptyset$) such that the sum of weighted edges between U_1 and U_2 is maximized, as shown in Fig. 2.4. The number of solution candidates increases

in $(2^N - 2)/2$, where N is the number of vertexes. Although practical applications of the Max-cut are few, Ising machines dedicated to solving optimization problems uses it to compare the performance between them. This is because the Max-cut is easy for Ising machines to solve it because the Max-cut does not have constraints such as TSP. An algorithm to find a solution to Max-cut is that a randomly or deterministically selected vertex is inserted in both U_1 and U_2, we calculate the cut size for both cases, and then, either the case of U_1 or U_2 that has the better cut size is adopted. The discrete-type Hopfield Neural Network is one of such algorithms, where the cut size is regarded as an energy value.

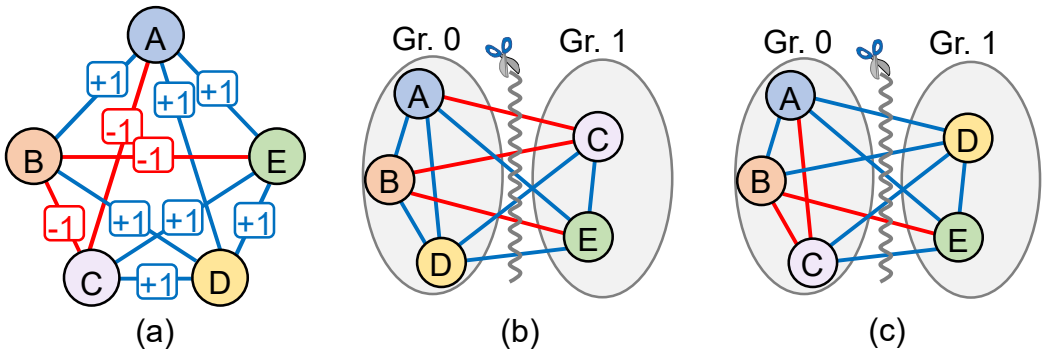

Figure 2.4: (a) Problem example of Max-cut and (b) worst and (c) optimal solutions, where cut sizes are 0 and 4, respectively.

2.3 Amoeba-inspired computing system

2.3.1 Amoeba-based computing system

An amoeboid organism, *Physarum polycephalum*, has highly-sophisticated ability to solving simple optimization problems even though it is a single-celled organism [30, 31, 44]. Aono et al. have developed an amoeba-based computing system to solve more complex optimization problems, the TSP. Figure 2.5 shows a schematic of the amoeba-based computing system. The amoeba-based computing system utilizes amoeboid organism's habits. The amoeboid organism placed on a nutrient-rich agar

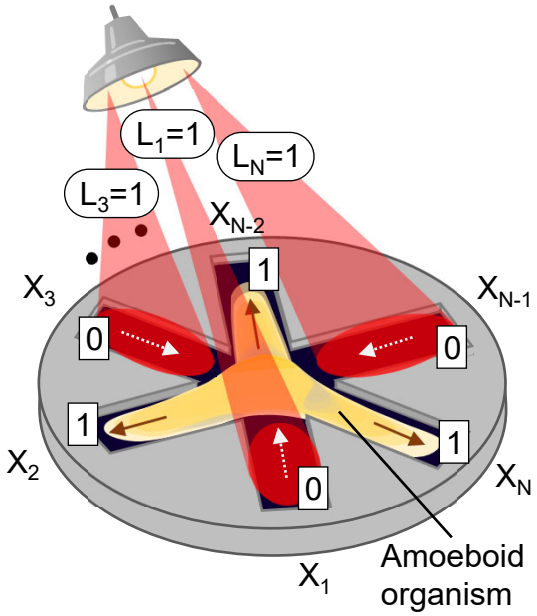

Figure 2.5: Amoeba-based computing system.

plate expands its body to maximize food intake [32, 33, 45]. The plate is divided by several grooves, where the grooves are assigned each variable and the number of grooves correspond to the number of variables of optimization problems to be solved. If the amoeboid organism expands its pseudopod like branches to grooves to absorb nutrients, then a variable state is regarded as 1 in accordance with the area of pseudo pod-like branches, otherwise it is regarded as 0. A monitor and feedback system always watches a state of the amoeba and illuminates a light to the amoeba's pseudo pod-like branches. The amoeboid organism avoids a light because it dislikes a light. Which branches should be illuminated is decided by a specified rule, called "bounce-back rule." The bounce-back rule lists up all combinations of variables that violate constraints to optimization problems to be solved in advance of starting a search for a solution. The deformation and light illumination are repeated for the amoeboid organism to find a solution. The light illumination pattern becomes constant when the amoeboid organism finds a solution because there are no variables that violate constraints. However, the amoeba fluctuates and deforms again; then, state variables change and the light illumination pattern also changes based on the

bounce-back rule. This fluctuation plays an important role in solving optimization problems because variables usually fall into a local optimum without the fluctuation [46, 47].

Aono et al. have demonstrated that the amoeboid organism found a solution to the TSP in a short time [33]. The amoeboid organism reached a solution in a linear growth of time as increasing the number of cities of the TSP. The finding was featured in an article released from an online science news site, *Phys.org*, and it attracted a general audience [33, 48]. The article was shared approximately 15K times, where the number of shares was the order of magnitudes greater that that of other typical articles on the site. Aono et al. have also developed amoeba-inspired algorithms for solving the SAT and TSP. In solving the SAT, the performance of the amoeba-inspired algorithms is superior to the WalkSAT, which is a well-known algorithm for solving the SAT [46, 47, 49]. The algorithm for solving the TSP reproduces the behavior of the amoeba that finds a solution in an amount of time that grows only linearly, where it assumes continuously redistributing its intracellular resource at a constant rate, while processing optical stimuli in parallel. Theses algorithms emulate parallel processing on a serial processing computer: a runtime on a conventional computer grows with the number of variables and estimation time of the bounce-back rule; thus, if we develop a physical system that performs the algorithms concurrently, we could find a solution in a short time.

2.3.2 Amoeba-inspired electronic computing system

Kasai et al. have developed an electronic computing system, called "electronic amoeba," inspired from the ability of the amoeboid organism to solving optimization problems and demonstrated that it could solve a NOR problem that is stated as finding a combination of variables that satisfies $\mathrm{NOR}(x_{i-1}, x_{i+1}) = 1\,(i = 1, 2, ..., N)$, where N is the number of variables [35]. The electronic amoeba implements the

essence of the solution-searching behavior of the amoeboid organism. Figure 2.6 shows a schematic of the electronic amoeba. In the electronic amoeba, an amoeba

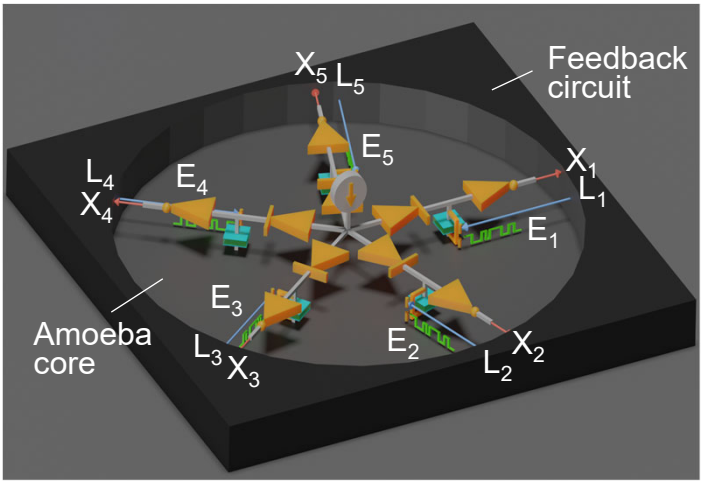

Figure 2.6: Amoeba-inspired electronic computing system, electronic amoeba.

core and a feedback circuit correspond to the amoeboid organism and the monitor and feedback system in the amoeba-based computing system shown in Fig. 2.5. The amoeba core searches a solution and the feedback circuit feeds back the information produced by the bounce-back rule to the amoeba core. Each branch in the amoeba core is connected by the current source, which implements the volume conservation of the amoeboid organism on a chip by the Kirchhoff's current law. The feedback circuit reads variable states from the amoeba core and feeds back to the metal oxide semiconductor field effect transistor (MOSFET) of branches in the amoeba core based on the bounce-back rule. The bounce-back rule consists of logical sums and products for the SAT, and it for the TSP consists of product-sum and thresholding. The fluctuation of the amoeboid organism is implemented with external noise sources that input a random value 0 or 1 to the MOSFET in the branch. As related works, superconductor and FPGA-based amoeba-inspired solvers have been developed and demonstrated that the ability to solve optimization problems [50–56], and only the concept of physical systems using exciton and Brownian ratchet has been reported [46, 47].

2.4 Summary

We reviewed the optimization problem and discussed the hardness of optimization problems. The SAT belongs to class NP-complete, and Max-cut and TSP belong to NP-hard. The number of solution candidates in these problems increases exponentially, which is called combinatorial explosion. We also reviewed an amoeba-based computing system and an amoeba-inspired electronic computing system that solve optimization problems. Related works of electronic amoebae was introduced.

Chapter 3

Previous system overview

3.1 Introduction

Due to the end of the Moore's law and Dennard scaling, the performance improvement of general purpose processors is no longer able to be expected [13–16]. Therefore, a domain-specific architecture (DSA) dedicated to processing special tasks has been developed [17, 18]. One of the DSAs is an Ising machine inspired from the Ising model, which solves optimization problems at low power consumption and a high speed [19, 20, 57, 58]. In this chapter, we review Ising machines dedicated to solving optimization problems as a counterpart of the electronic amoeba, and I point out weak points in Ising machines compared with the electronic amoeba.

3.2 Ising machine

The first physical computing system for searching for a solution to optimization problems is Hopfield-Tank Neural Network (HTNN or HNN): Hopfield and Tank developed a recurrent neural network called Hopfield-Tank neural network [59, 60]. The most important aspect of their works is that they formulated the Lyapunov function of the HNN: the HNN converges at a stable state while reducing the energy,

where the state corresponds to a solution to optimization problems and a memory pattern in associative memories. However, the HNN did not work well unfortunately because the system fall into the local minimum state and sometimes reached an illegal candidate for large-scale problems [61–63]. Moreover, the rapid performance improvement of transistors based on the Moore's law and Dennard scaling led one's effort to the development of a conventional digital computer rather than such an analog computer [13, 14].

Here again, physical computing systems have attracted one's attention because of the end of Moore's law and Dennard scaling [15, 17]. A physically inspired computing system dedicated to solving optimization problems, called "Ising machine," has been drawing attention. Its solution search is based on the Ising model and utilizes natural convergence behavior of physical systems. Several practical applications for Ising machines have been proposed [1–10]. Moreover, softwares that connects the user with the Ising machine have been developed [64, 65].

The Ising model is a statistical model that describes spin dynamics of magnetic materials [66]. Spin states in the Ising model are represented by up and down, $(-1, 1)$. As shown in Fig. 3.1, in the square-lattice Ising model, a spin is connected to other spins by interaction coefficients $\boldsymbol{J}$. The Hamiltonian (energy) is given by

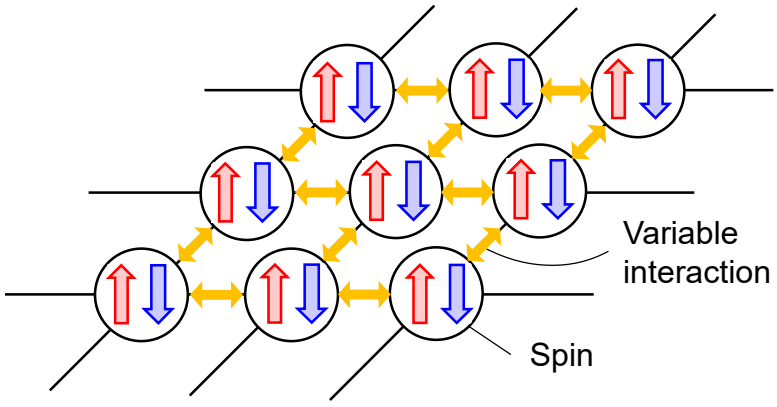

Figure 3.1: Two-dimensional Ising model

$$H = -\sum J_{ij} s_i s_j - \sum h_i s_i, \tag{3.1}$$

where s represents a spin state and h is an external magnetic field. If a spin s_i is connected to another s_j by $J_{ij}(< 0)$, the spin s_i has to be the same value to s_j because the Hamiltonian shown in Eq. 3.1 decreases, else if $J_{ij} > 0$, s_i has to reverse to s_j. When $J_{ij} < 0$, it is called as a ferromagnetic interaction, otherwise an anti-ferromagnetic interation. The exact solution (ground state) of the two-dimentional Ising model without the external magnetic field was derived; however, that with the external magnetic field has not been derived and belongs to the NP-hard. The three-dimentional lattice model also belongs to the NP-hard.

We have to transform optimization problems to the Hamiltonian, i.e., derive spin interaction J_{ij} from the problem to be solved. The ground state of the Hamiltonian corresponds to an optimal solution of optimization problems. The Ising model is equivalent to the quadratic unconstrained binary optimization (QUBO) by $x_i = (s_i + 1)/2$. Formulation of the QUBO for Ising machines have been studied [67].

Simulated annealing (SA) is usually used to solve the Ising model (find a ground state) [68, 69]. The SA utilizes an analogy with the change from disordered arrangement atoms in a metal to ordered ones by gradually cooling from high temperature to low temperature. The atoms are disordered at high temperature; then, they are frozen at low temperature. This process is called *annealing*. The SA utilizes the annealing process by regarding atoms as variables of an optimization problem. A local search algorithm in solving optimization problems successively improves a solution by searching neighborhood; however, it falls into a local minimum as shown in Fig. 3.2(a). The SA probabilistically flips a variable to escape from a local minimum. Initially, the temperature in the SA is set to be high: a variable state becomes random. Then, the temperature gradually decreases; variables freeze. This process enables for the SA to find an optimal solution while escaping from a local optimum as shown in Fig. 3.2(b).

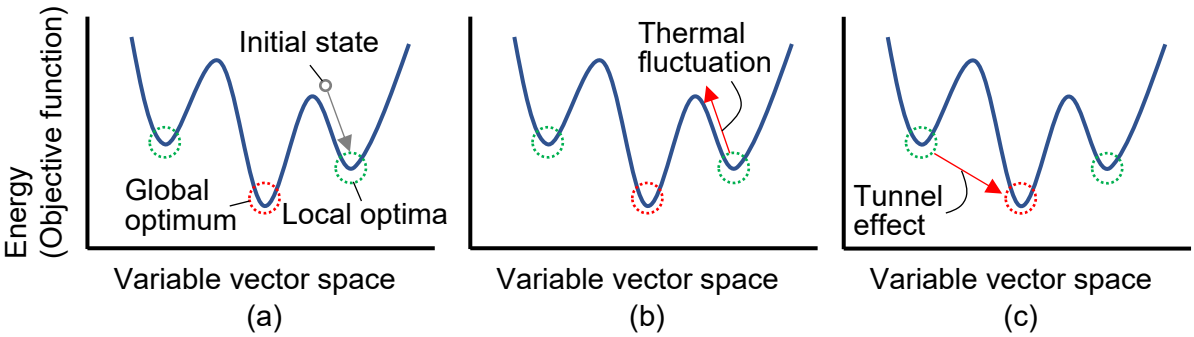

Figure 3.2: (a) Trapping in local minimum and escaping from local minimum using (b) thermal fluctuation in simulated annealing and (c) quantum tunnel effect in quantum annealing

Quantum annealing (QA) is inspired from the SA and utilizes the quantum tunneling effect to escape from local optima as shown in Fig. 3.2(c) [19, 70]. In the QA, the Hamiltonian is deformed by introducing a transverse field into Eq. 3.1,

$$H = -\Lambda(t)(\sum J_{ij} s_i s_j + \sum h_i s_i) - \Gamma(t) \sum s_i, \tag{3.2}$$

where Λ is a value of the Hamiltonian and Γ is a value of the transverse field. The value of the Hamiltonian and the transverse field are initially chosen to be low and high, respectively; therefore the behavior of the system is dominated by the transverse field, which indicates that spin states are in the quantum entanglement. Then Λ gradually increases and Γ gradually decreases with time evolution [19, 70]. When time evolves enough, the Hamiltonian term dominates Eq. 3.2 and the system becomes stable. The freezing state corresponds to a solution of optimization problems: the system finds an optimal solution to optimization problems by increasing Λ and decreasing Γ in sufficient time. However, when the annealing time is insufficient, the system falls into a local optimum or an illegal solution that does not satisfy constraints to optimization problems.

Several Ising machines have been developed to apply them to practical applications in the real world. Especially, major electrical manufacturer in Japan, Hitachi Ltd. (CMOS annealing (CA) [20, 71–76]), Fujitsu Ltd. (Digital Annealer (DA)

[57]), Toshiba Ltd. (Simulated Bifurcation Machine (SBM) [58, 77–79]), and NTT Ltd. (Coherent Ising Machine (CIM) [80–82]) have developed the Ising machine.

Ising machines are divided by how to implement a spin and variable interactions: usually, they are implemented by a fully analog, an analog-digital hybrid, and a fully digital Ising machine. A quantum annealer developed by D-Wave Inc., which is the first commercial quantum computing system and sparked the development of Ising machines, is implemented by a superconducting circuit as a spin and external flux as the variable interaction; therefore, the D-Wave's Ising machine is regarded as an all-analog Ising machine [19]. Other analog Ising machines use a phase as a spin [83–87].

In the CIM, a spin is also implemented by a phase but variable interactions are calculated by an external feedback system that is implemented by field-programmable gate array (FPGA); therefore, the CIM is categorized as the analog-digital hybrid Ising machine. Low-cost implementation for the CIM was reported [88]. Although the spin operates at high frequency, the FPGA calculating product sum between variables and variable interactions, analog-digital converters to readout the states of spins, and digital-analog converters to feed back to the spins limit its performance. Other hybrid systems have been developed [89, 90], which implement a spin by a digital memory but variable interactions by resistors in a crossbar circuit, where the product-sum operation is performed based on the Kirchhoff's current law (this system is similar to the electronic amoeba discussed in Chpt. 5, in terms of using the crossbar circuit). Also, an Ising machine utilizing chaotic dynamics in NbO2 memristor has been developed [91]

A pioneer of the fully digital implemented Ising machines is the CA, which was reported in 2016, and after that, the DA was developed. The CA and DA are implemented by the FPGA or application-specific interated circuit (ASIC). Fully digital Ising machines are easy to implement because the model is compatible with the digital implementation. In academic research institutes, Ising machines implemented by

the FPGA [92, 93] and ASIC [94–97] have been developed. Graphics processing unit (GPU) -based Ising machines have also been developed because the most computing power required in the Ising model is the product-sum operations that is suitable for the GPU [98–100].

3.3 Weak points in Ising machine

In Ref. [67], the QUBO for the Max-cut and TSP that are equivalent to the Ising model was reported. The Ising machine searches for a spin assignment that minimizes the Hamiltonian. The Hamiltonian is generally given by

$$H = H_{cost} + \lambda H_{penalty}, \tag{3.3}$$

where H_{cost} is an objective function that we have to minimize as possible, such as a route length in the TSP, and $H_{penalty}$ is a penalty term that describes constraints of optimization problems, such as prohibition of twice visit to the city and of a visit to the city at the same time. λ is the strength of the penalty term to the objective function [64]. λ should be set as an optimal value: the system falls into an illegal state if λ is small, although the optimality of obtained states is high; on the other hand, if λ is large, the system always finds a legal solution without falling into an illegal state, although the quality of legal solutions is relatively low. Therefore, searching for the optimal λ is needed in advance of starting search for a solution [28, 29]. This weak point is discussed in Appx. A. To repair converged spin alignments in D-Wave's Ising machine when the final states fall at an illegal state, it requires the post-processing to satisfy constraints of optimization problems to be solved and improve the quality of solutions [4, 19, 101]. The similar post-processings are necessary for the Hopfield's neural network to obtain a legal solution.

A graph structure of Ising machines developed by D-Wave Inc. and Hitachi

Ltd. are sparse, which is called chimera graph and King graph, respectively [19, 20]. The structure of the chimera graph is shown in Fig. 3.3. The number of variable interactions per a spin in the chimera graph are four; therefore, we should prepare redundant variables to connect all variables when solving a complete graph problem. The D-Wave's Ising machine needs $O(N^4)$ variables to solve a problem that has complete graph: for example, the TSP. It is difficult for us to implement a large number of variables into the Ising machine due to physical complexity; thus, the Ising machine is hard to solve a large problem. To solve a large problem by

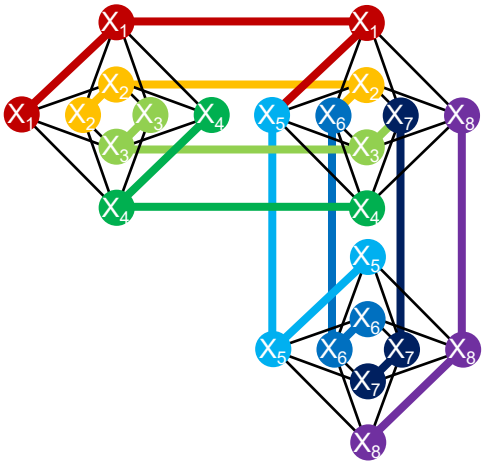

Figure 3.3: Chimera graph structure of D-Wave's Ising machine.

the Ising machine whose connectivity is sparse, embedding methods have been proposed [21–27]. Searching for efficient embedding is also optimization problems, and therefore, we need to solve optimization problem in advance of solving the problem we hope to solve. Moreover, the solution-searching performance of Ising machines degrades due to introduction of the redundant variables [27, 28].

3.4 Summary

In this chapter, we reviewed Ising machines which is based on the Ising model and also discussed weak points in them. Several Ising machines have been developed to solve optimization problems in the real world. The Ising machines require pre-

processing to map the problem onto the Ising machine, and therefore they require to solve the optimization problem to map the problem to be solved. To implement the TSP on the Ising machine, $O(N^4)$ redundant variable are required, and its performance degrades due to introduced redundant variables.

Chapter 4

Amoeba-inspired analog-digital hybrid computing system for solving satisfiability problem

4.1 Introduction

An amoeboid organism shows high intelligence such that it solves a maze and optimize a transportation network [30, 31]. Aono et al. developed an amoeba-based computing system that solves combinatorial optimization problems [33, 45]. Optimization problems can be applied to practical applications in the real world. However, the number of solution candidates of an optimization problem exponentially increases as increasing the problem size; therefore, a conventional computer that operates sequentially cannot solve the problems. Aono et al. showed that the amoeba-based computing system could solve the NOR problem, which is stated as follows: find a combination of variables $\boldsymbol{x} = x_1, x_2, ..., x_N$ that satisfy $x_i = \mathrm{NOR}(x_{i-1}, x_{i+1})$ where $i \in (1, 2, ..., N)$ [45]. They also examined that the system found a high-quality solution to the traveling salesman problem (TSP) only in linear time growth when increasing the number of cities. However, it is difficult to apply the system to prac-

tical applications because the moving speed of the amoeboid organism is 1 cm/h [34].

Kasai et al. have electronically implemented an amoeba-based computing system with analog circuits using charge dynamics in a capacitor network, which is called "electronic amoeba" [35, 45], expecting that we would obtain a solution of optimization problems quickly by implementing its behavior electronically. They demonstrated that the electronic amoeba could find a solution to the NOR problem. However, a conventional computer can solve the NOR problem in a polynomial time, and so far, the solution search capability of the electronic amoeba for intractable problems for the computer has not been evaluated for the electronic amoeba. In addition, although the error property in the amoeba-inspired algorithm influences on its solution-searching performance to the satisfiability problem (SAT) [46, 47], its effect on the performance of the electronic amoeba has not been evaluated.

In this chapter, I propose the electronic amoeba implemented with analog-digital-hybrid circuits and demonstrate that the electronic amoeba can find a solution to the SAT. To evaluate its performance to the SAT, it is needed to change a problem to be solved because some instances can be solved easily but others hardly; thus, I implement the feedback circuit shown in Fig. 2.6 by the reconfigurable microcontroller.

4.2 Electronic amoeba for solving SAT

I implemented the electronic amoeba by commercial electronic devices and a microcontroller (Arduino DUE @ 84 MHz) for solving the SAT as shown in Fig. 4.1. The descriptions of the electronic amoeba and SAT are shown in Chpt. 2. The branches in the amoeba core search a solution and the controller circuit feeds back to the amoeba core based on the bounce-back rule for the SAT. The bounce-back rule is describe in the next section. The error, E, randomly switches the MOSFET of the

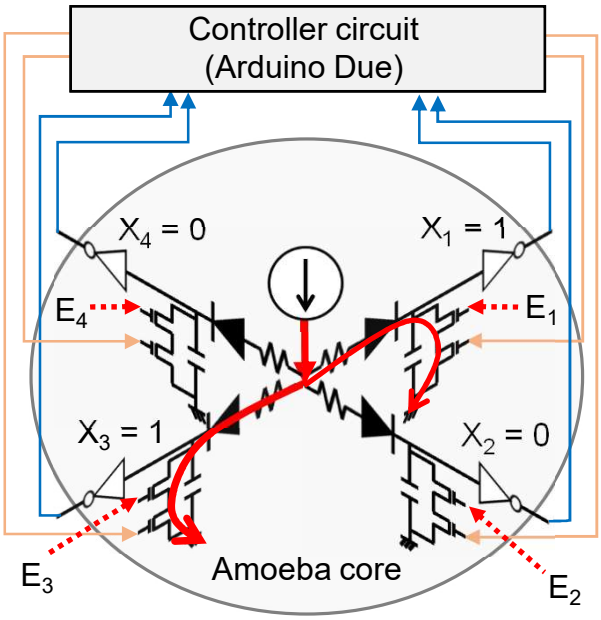

Figure 4.1: Electronic amoeba for solving SAT.

pseudopod regardless of the bounce-back signal.

I defined error parameters (shown in Fig. 4.2) to investigate the effect of an error property on the solution-searching performance of the electronic amoeba. When the

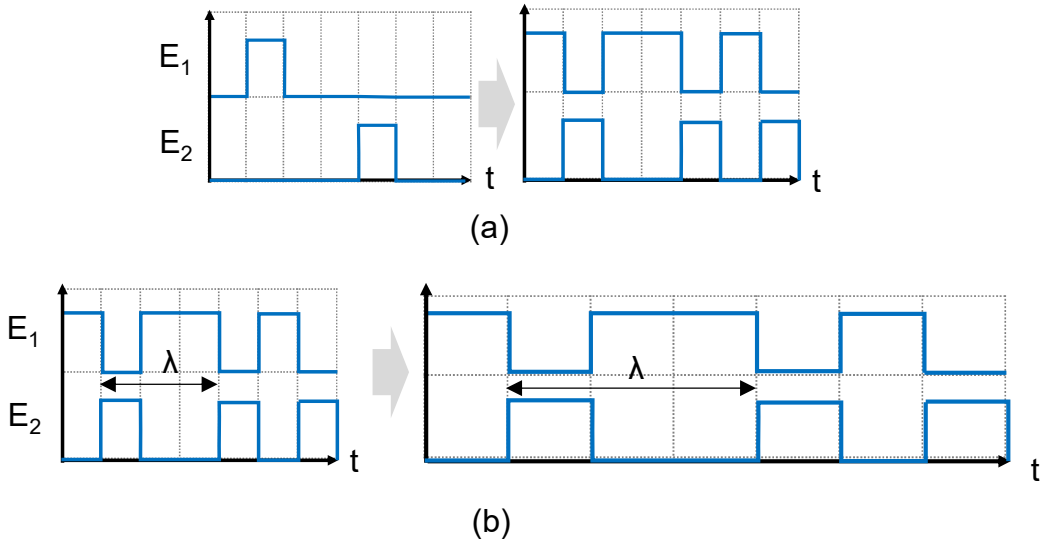

Figure 4.2: Error parameters. (a) Error probability and (b) average error period.

error occurs, the variable is forcibly changed from 1 to 0 or fixed at 0 by stopping current flow via the MOSFET. The error E_i for the variable X_i is independent of other variables. It occurs with a probability E_p produced by the random number

generator and is updated every error duration time t_e because of an asynchronous operation of the amoeba core. I also defined an average error period, λ, as $\lambda = t_e/E_p$, which denotes the period during which errors occur on average.

The error signal was produced by the micro-controller that produces random signals and sends the signals to the MOSFETs in the amoeba core. We implemented Xorshift on the micro-controller, which is the fastest algorithm to produce random numbers [102].

4.3 Bounce-back rule for SAT

Whether to decrease the value of the redundant variable (turn off the MOSFET) is, in accordance with Ref. [46, 47], given by

$$L_{i,v}(t+1) = \begin{cases} 0 & (\text{otherwise}) \\ 1 & (\text{if } B \ni (P,Q) \,\text{s.t.}\, Q \ni (i,v)) \,\text{and}\, \forall (j,u) \ni P(X^b_{j,u}(t+1) = 1) \end{cases}, \tag{4.1}$$

where

$$B = INTRA \cup INTER \cup CONTRA, \tag{4.2}$$

and

$$X^b_{i,v}(t+1) = \begin{cases} 0 & (\text{if } X_{i,v}(t) < T_H) \\ 1 & (\text{if } X_{i,v}(t) \geq T_H) \end{cases}. \tag{4.3}$$

When $L_{i,v} = 1$, $X^b_{i,v}$ is prohibited to become 1 by decreasing the value of $X_{i,v}$. The INTRA is a constraint to avoid from contradiction due to introducing the redundant variables:

$$INTRA \ni ((i,v),(i,1-v)). \tag{4.4}$$

The INTER forbids not to satisfy the clause:

$$INTER \ni \begin{cases} (P,(i,0))\,(\text{if } C_k \ni i) \\ (P,(i,1))\,(\text{if } C_k \ni -i) \end{cases}, \tag{4.5}$$

where

$$P \ni \begin{cases} (j,0)\,(\text{if } C_k \ni j) \\ (j,1)\,(\text{if } C_k \ni -j) \end{cases}. \tag{4.6}$$

The CONTRA resolves a contradiction due to the INTER:

$$CONTRA \ni (P \cup P', P \cup P')\,(\text{if}(P,(i,0) \ni INTER) \text{ and } (P',(i,1) \ni INTER)). \tag{4.7}$$

Now we consider the bounce-back rule for $F(x_1,x_2,x_3) = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$ as an example of SAT instances. The bounce-back rule for the instance is shown in Table 4.1. The INTRA prohibits $X^b_{i,v} = 1$ when $X^b_{i,1-v} = 1$. In the INTER, considering $C_1 = (x_1 \vee x_2 \vee \overline{x_3})$ for instance, when $x_2 \vee \overline{x_3} = 0$, x_1 does not have to become 0 to satisfy C_1; therefore, the bounce-back rule prohibits $X^b_{1,0} = 1$. When $x_2 \vee \overline{x_3} = 0$ in C_1 and $\overline{x_2} \vee x_3 = 1$ in C_2 for instance, the bounce-back rule prohibits $X^b_{1,0} = 1$ and $X^b_{1,1} = 1$ simultaneously due to the INTER: in this simulation, there happens a contradiction that x_1 cannot become 0 or 1. The CONTRA resolves the contradiction by prohibiting factors that promote $X^b_{1,0} = 0$ and $X^b_{1,1} = 0$ by prohibiting $(X^b_{2,0} \wedge X^b_{3,1}) \wedge (X^b_{2,1} \wedge X^b_{3,0}) = 1$.

Table 4.1: Example of bounce-back rule for $F(x_1, x_2, x_3) = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$

$L_{i,v}$	INTRA	INTER	CONTRA
$L_{1,0}$	$X^b_{1,1}$	$X^b_{2,0} \wedge X^b_{3,1}$	$(X^b_{1,0} \wedge X^b_{3,1} \wedge X^b_{1,1} \wedge X^b_{3,0}) \vee (X^b_{1,1} \wedge X^b_{2,1} \wedge X^b_{1,0} \wedge X^b_{2,0})$
$L_{1,1}$	$X^b_{1,0}$	$X^b_{2,1} \wedge X^b_{3,0}$	$(X^b_{1,0} \wedge X^b_{3,1} \wedge X^b_{1,1} \wedge X^b_{3,0}) \vee (X^b_{1,1} \wedge X^b_{2,1} \wedge X^b_{1,0} \wedge X^b_{2,0})$
$L_{2,0}$	$X^b_{2,1}$	$X^b_{1,0} \wedge X^b_{3,1}$	$(X^b_{2,0} \wedge X^b_{3,1} \wedge X^b_{2,1} \wedge X^b_{3,0}) \vee (X^b_{1,1} \wedge X^b_{2,1} \wedge X^b_{1,0} \wedge X^b_{2,0})$
$L_{2,1}$	$X^b_{2,0}$	$X^b_{1,1} \wedge X^b_{3,0}$	$(X^b_{2,0} \wedge X^b_{3,1} \wedge X^b_{2,1} \wedge X^b_{3,0}) \vee (X^b_{1,1} \wedge X^b_{2,1} \wedge X^b_{1,0} \wedge X^b_{2,0})$
$L_{3,0}$	$X^b_{3,1}$	$X^b_{1,1} \wedge X^b_{2,1}$	$(X^b_{2,0} \wedge X^b_{3,1} \wedge X^b_{2,1} \wedge X^b_{3,0}) \vee (X^b_{1,0} \wedge X^b_{3,1} \wedge X^b_{1,1} \wedge X^b_{3,0})$
$L_{3,1}$	$X^b_{3,0}$	$X^b_{1,0} \wedge X^b_{2,1}$	$(X^b_{2,0} \wedge X^b_{3,1} \wedge X^b_{2,1} \wedge X^b_{3,0}) \vee (X^b_{1,0} \wedge X^b_{3,1} \wedge X^b_{1,1} \wedge X^b_{3,0})$

4.4 Experimental results of electronic amoeba

I evaluated the effect of the electronic amoeba on the error parameters. Solved instances were random 3-SAT that has 12 variables and 59 clauses. The solution-searching time was capped at 4.75 s; if the time reached the limit, I regarded the time as 4.75 s. Figures 4.3(a) and 4.3(b) show the photograph of the physically implemented electronic amoeba and time evolutions of variables that show 4 variables out of 24 variables. All variables started from 3 V because of the capacitor

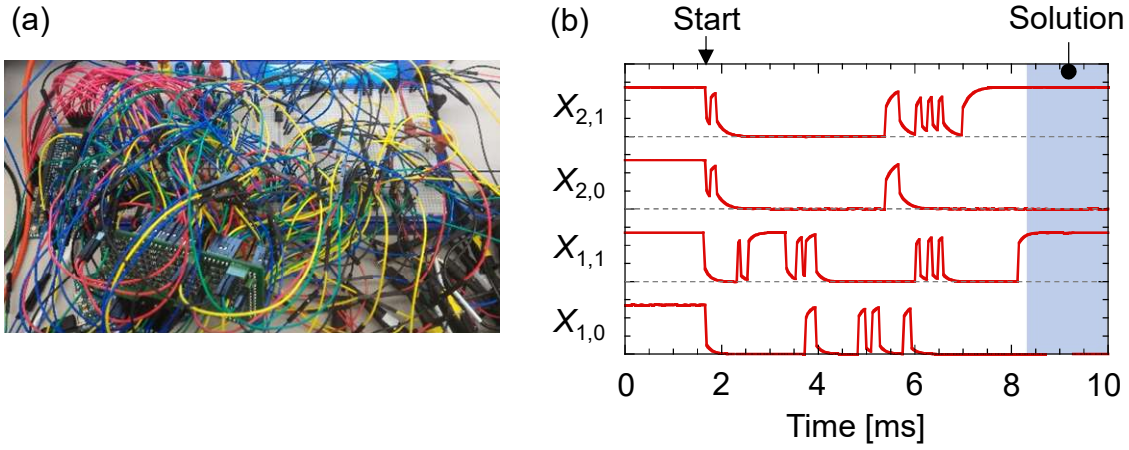

Figure 4.3: (a) Photograph of physically implemented electronic amoeba and (b) time evolutions of variables that show 4 variables out of 24 variables.

charge was initially 0 V, which correspond to all variables taking 1 ($\boldsymbol{X}^b = 1$). The capacitor was charged by injecting the current from the current source. Then, the variables approached to 0 V. The micro-controller that implements the bounce-back rule read the states of variables and estimated the variable to flip in accordance with the bounce-back rule, and it fed back to the amoeba core. The variables were forcibly flipped randomly by the error. These processes were repeated, and finally, the electronic amoeba found a solution.

Figure 4.4 shows the experimental results when changing the error parameters. The number of tested instances was 25, and each instance was tested 30 times. The solution-searching time depended on the error parameters, E_p and λ. We confirmed three things from the results as follows. First the optimal error period for each error probability existed, for example, $\lambda = 3$ ms when $E_p = 0.2$. Second the error period dependence of the solution-searching time depended on the magnitude of the error probability: if the error probability was low, it was unlikely to be influenced by the error period, otherwise it was largely influenced. Third such the optimal error parameters as make the solution-searching time minimum (0.2 s) existed in $E_p = 0.65$ and $\lambda = 0.6$.

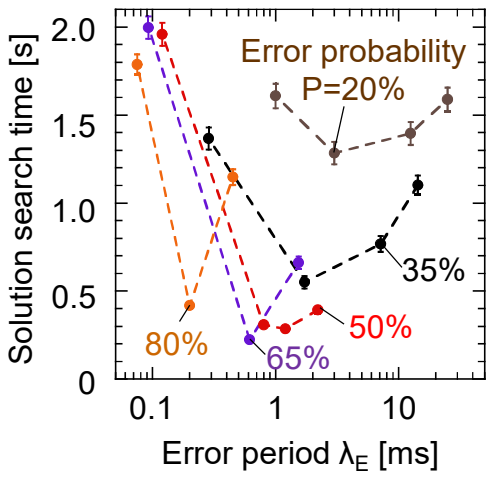

Figure 4.4: Experimental results of electronic amoeba when solving random 3-SAT having 12 variables with 59 clauses.

4.5 Asymmetric dynamics in amoeba-inspired algorithm

4.5.1 Original AmoebaSAT

To reproduce and understand the solution-searching behavior of the electronic amoeba, I used an amoeba-inspired algorithm, called "AmoebaSAT" [47]. There are several versions of amoeba-inspired algorithms for solving SAT [46, 47, 49]. We used the AmoebaSAT in Ref. [47] because the solution-searching behavior of the algorithm is similar to the electronic amoeba. The pseudocode of the AmoebaSAT is shown in Alg. 1.

Algorithm 1 AmoebaSAT algorithm

```
INITIALIZE: t ← 0, c ← 1, X ← 0
while t < t_limit do
    update X^b and L using Eqs. 4.3 and 4.1, respectively
    if X^b_{i,v} ⊻ L_{i,v} = 1 for all i and v then
        return X
    end if
    update X using Eqs. 4.8–4.10
    t ← t + 1
    if c = s_e then
        c ← 1
    else
        c ← c + 1
    end if
end while
```

Following equations are used in Alg. 1:

$$X_{i,v}(t+1) = \begin{cases} X_{i,v}(t)+1 & (\text{if } Y_{i,v}(t+1)=1 \text{ and } X_{i,v}(t)<UL) \\ X_{i,v}(t)-1 & (\text{if } Y_{i,v}(t+1)=0 \text{ and } X_{i,v}(t)>0) \\ X_{i,v}(t) & (\text{otherwise}) \end{cases}, \tag{4.8}$$

$$Y_{i,v}(t+1) = \begin{cases} 1 & (\text{if } L_{i,v}(t+1) = 0 \text{ and } E_{i,v}(t+1) = 0) \\ 0 & (\text{otherwise}) \end{cases}, \tag{4.9}$$

$$E_{i,v}(t+1) = \begin{cases} E_{i,v}(t) & (c < s_e) \\ 1 & (\text{if } c = s_e \text{ and } Z_{i,v}(t) + E_p > 1) \\ 0 & (\text{otherwise}) \end{cases}, \tag{4.10}$$

where $Z_{i,v}$ is a noise source that takes a value between 0 and 1, UL is an upper limit of $X_{i,v}$, s_e is an error duration iteration, and c ($c = 1, 2, ...$) is a counter variable to count the number of occurrences of the same error set. I modified Eq. 4.8 as adding s_e and c, compared with original one. $Y_{i,v}$ determines whether to supply the resource to $X_{i,v}$ based on $L_{i,v}$ and $E_{i,v}$. s_e and UL are set to 1 and 2 in the AmoebaSAT, respectively [47]; then, the error set is updated every iteration and $X_{i,v} \in 0, 1, 2$.

I introduced the error period, λ, to the AmoebaSAT by $s_e > 1$ in Eq. 4.10. The error period λ is defined as $\lambda = s_e/E_p$. We can judge whether the AmoebaSAT finds a solution by checking $X^b_{i,v}(t) \veebar L_{i,v}(t+1) = 1$ for all i and v without checking whether to satisfy all clauses of the given instance [47].

4.5.2 AmoebaSAT with asymmetric dynamics

Now we focus on differences in the solution-searching behavior between the electronic amoeba and the AmoebaSAT. The electronic amoeba has a time constant in a variable transition due to the capacitor and the transconductance of the MOSFET in the branch of the amoeba core that produces asymmetric dynamics for the solution search as shown in Fig. 4.5. When the feedback circuit sends signals to the MOSFETs in the branches to turn on them, the capacitor in the branch immediately discharges and the current from the current source flows into the capacitor. On the other hand, when the feedback circuit sends signals to the MOSFETs to turn off

them, the capacitor gradually charged distributed from the current source. That is, the variable rapidly changes from 0 to 1, but it slowly changes from 1 to 0.

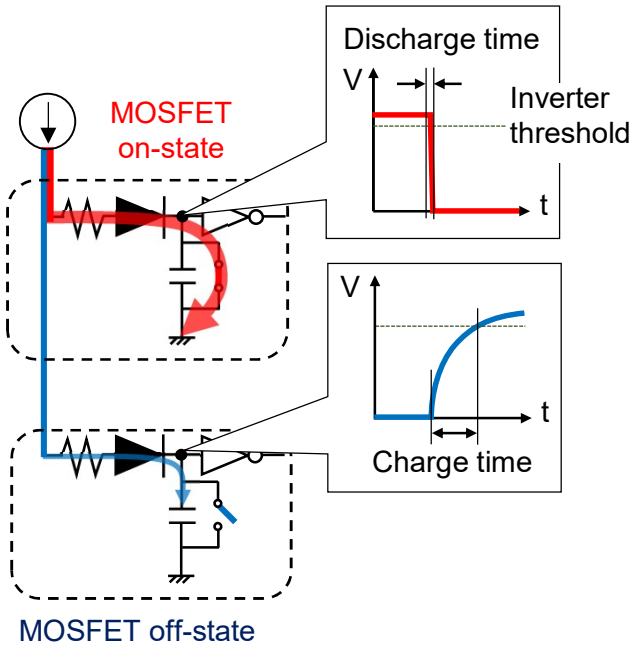

Figure 4.5: Asymmetric dynamics of electronic amoeba.

The variables in the AmoebaSAT immediately change when $L_{i,v} = 1$ or $E_{i,v} = 1$, and the transition speed ratios is symmetric as shown in Eq. 4.8. We reformulate Eq. 4.8 to introduce the asymmetric dynamics to the AmoebaSAT as follows:

$$X_{i,v}(t+1) = \begin{cases} X_{i,v}(t) + T_E & (\text{if } Y_{i,v}(t+1) = 1 \text{ and } X_{i,v}(t) + T_E < UL) \\ UL & (\text{if } Y_{i,v}(t+1) = 1 \text{ and } X_{i,v}(t) + T_E \geq UL) \\ X_{i,v}(t) - T_S & (\text{if } Y_{i,v}(t+1) = 0 \text{ and } X_{i,v}(t) - T_S > 0) \\ 0 & (\text{if } Y_{i,v}(t+1) = 0 \text{ and } X_{i,v}(t) - T_S \leq 0) \\ X_{i,v}(t) & (\text{otherwise}) \end{cases}, \quad (4.11)$$

where T_E and T_S represent the amount of the expansion and shrinkage after one iteration, respectively. The asymmetric dynamics are introduced by setting the value of UL more than 2 and the asymmetric relation between T_E and T_S. The AmoebaSAT

with the asymmetry is equal to Alg. 1 by replacing Eq. 4.8 with Eq. 4.11.

4.5.3 Numerical simulation results of AmoebaSAT with and without asymmetric dynamics

Figure 4.6 shows the numerical simulation results of the AmoebaSAT with the symmetric and asymmetric dynamics. Used instances were the same to the experiments of the electronic amoeba (shown in Fig. 4.4), and we tested 100 times for each instance, where the maximum iteration was set to 10^5. If the iteration exceeds 10^5, the solution search was forcibly aborted; then, the iteration was regarded as 10^5. In the AmoebaSAT with the asymmetric dynamics, T_E, T_S, and UL were set to 10, 1 and 19. In Fig. 4.6, the vertical axis denotes an average iteration to find a solution, where the iteration corresponds to t in Alg. 1. In Fig. 4.6(a), we obtained different results from Fig. 4.4. The iterations were deteriorated over $E_p = 0.2$: they almost reached the iteration limit (10^5) when $E_p = 0.5$ to 0.8. Note that the trend less than $\lambda = 5$ in $E_p = 0.2$ cannot be evaluated because both the AmoebaSAT with the symmetric and asymmetric dynamics consist of the discrete time step; on the other hand, the electronic amoeba operates continuously. Figure 4.6(b) shows the numerical simulation results of the AmoebaSAT with the asymmetric dynamics. As opposed to the results of the symmetry, those of the asymmetry were similar to the experimental results of the electronic amoeba. The AmoebaSAT with the asymmetric dynamics has the robustness to the error property as with the experimental results of the electronic amoeba.

I evaluated the performance of the AmoebaSAT with the symmetric and asymmetric dynamics against the variabilities in the error probability to test a hypothesis that the the asymmetry guarantees that organisms can efficiently perform complex tasks even though they have diversity [103–106]. Figure 4.7(a) shows the example of the error probability distribution for the symmetry and asymmetry. Figure 4.7(b)

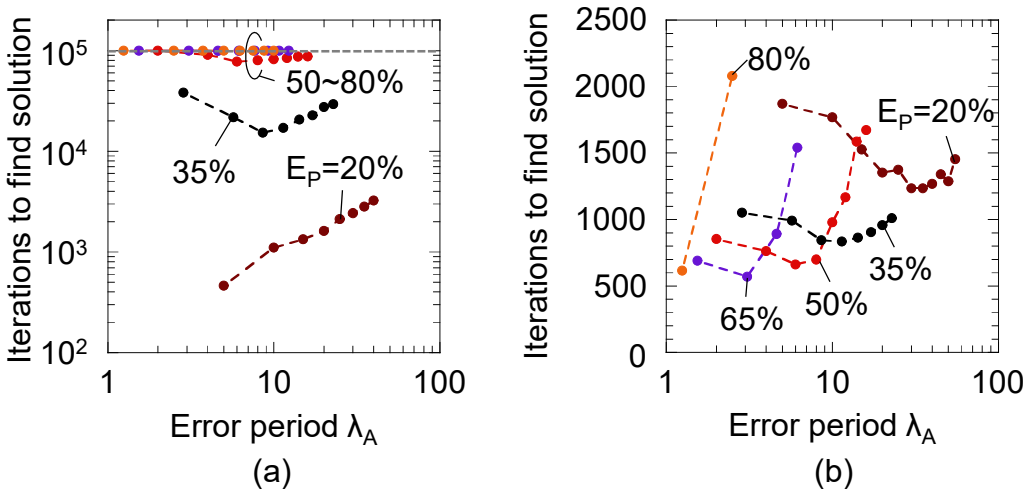

Figure 4.6: Numerical simulation results of AmoebaSAT with (a) symmetric dynamics and (b) asymmetric dynamics.

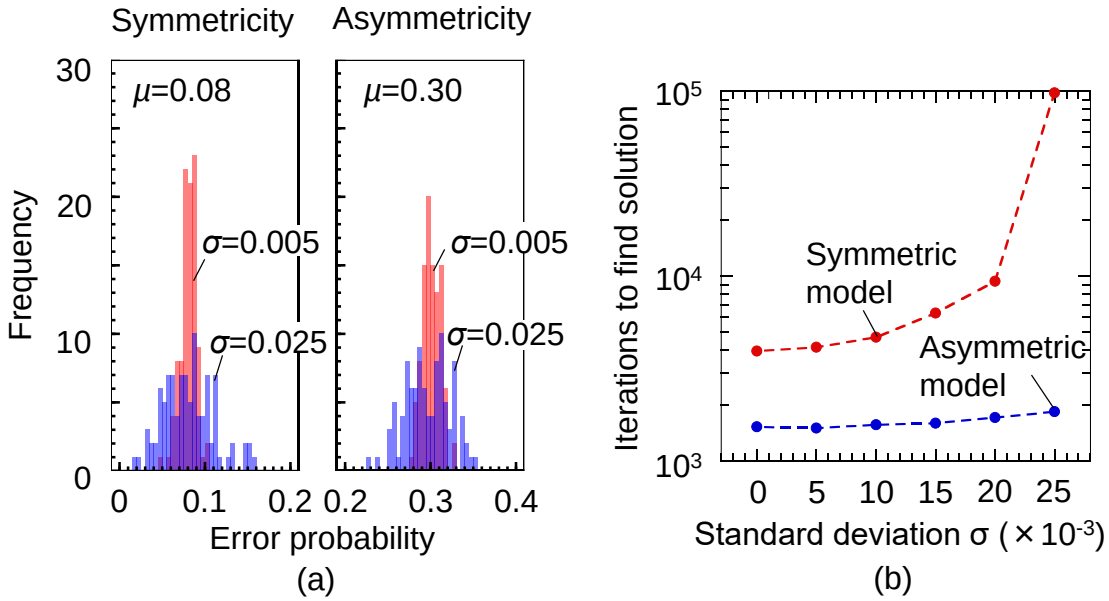

Figure 4.7: (a) Example of error probability distribution when $\sigma = 0.05$ and $\sigma = 0.025$. (b) Numerical simulation results of AmeobaSAT with symmetric and asymmetric dynamics.

shows the numerical simulation results when the average of normaly distributed error probabilities was 0.08 and 0.3 of the AmoebaSAT with the symmetric and asymmetric dynamics, respectively. In the asymmetry, the parameters were set to $s_e = 1$, $UL = 9$, $T_E = 5$, $T_S = 2$, and $T_H = 5$. Used instance was the 50-variable 3-SAT instance with 218 clauses, named uf50-01.cnf, which is a benchmark problem of the SAT and available in Ref. [107]. I produced 100 different types of error probability variations, for each standard deviation σ and tried 100 trials for each type (i.e., the total number of trials was 10000 for each σ). Iterations of the symmetric model increased rapidly when increasing σ; on the other hand, those of the asymmetric model was almost constant to the influence of σ.

My interest is not only the robustness to the error property but also the efficiency; therefore, I evaluated the performance of the symmetric and asymmetric model when increasing the instance size. Figure 4.8 shows the comparison results of the symmetric and asymmetric models when solving random 3-SATs with $N = 50$ (uf50-01–uf50-0100), $N = 75$ (uf75-01–uf75-0100), and $N = 100$ (uf100-01–uf100-0100), which are available in Ref. [107]. The error parameters for the symmetric model were $E_p = 0.08$ and $s_e = 1$. Those for the asymmetric model were $E_p = 0.3$ and $s_e = 1$. Other parameters for the asymmetric model were $UL = 9$, $T_E = 5$, $T_S = 2$, and $T_H = 5$. The solution-searching performance could be improved by introducing the asymmetry for all instances. The average iterations for $N = 50$, 75, and 100 are shown in Fig. 4.9. The iterations of the asymmetric model were 2, 3, and 19 times faster than those of the symmetric model when solving $N = 50$, 75, and 100, respectively. The gaps widened as increasing the number of variables, N, and we confirm that the performance exponentially was improved owing to the asymmetric dynamics.

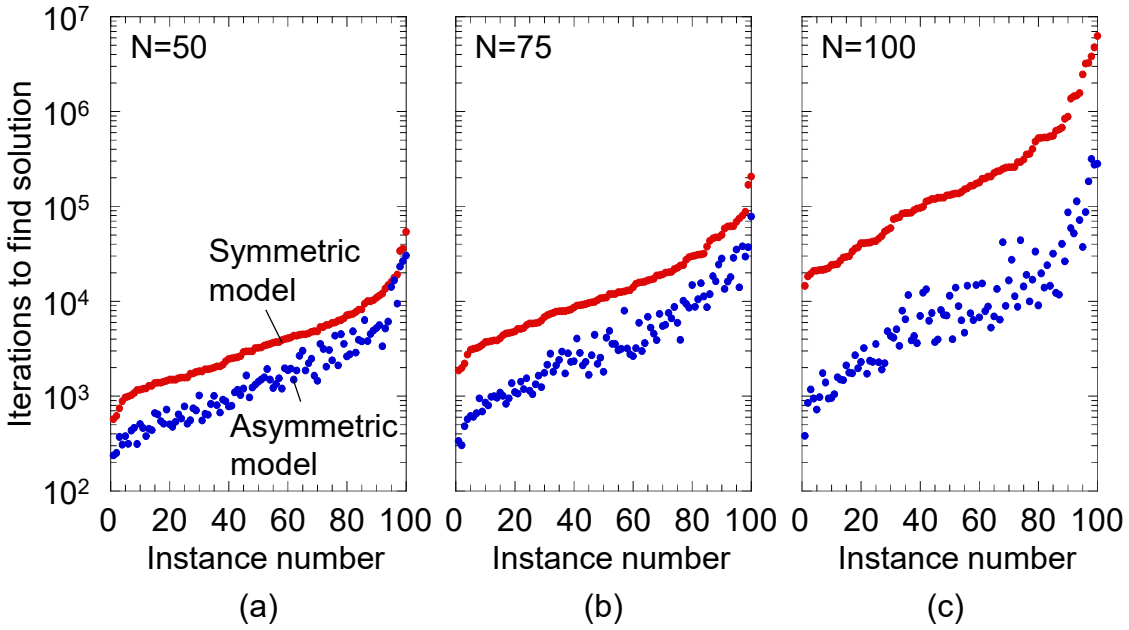

Figure 4.8: Solution-searching performance of symmetric and asymmetric models when solving (a) $N = 50$, (b) $N = 75$, and (c) $N = 100$. Results are sorted in ascending order of iterations about symmetric model. I tested 100 times for each instance.

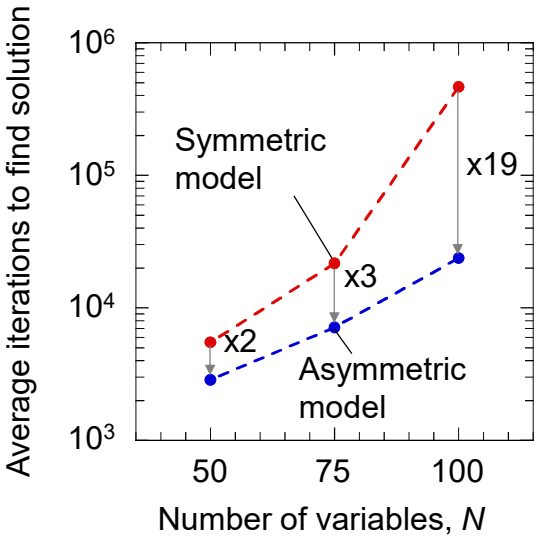

Figure 4.9: Average iterations of symmetric and asymmetric models to find solution when solving $N = 50$, 75, and 100.

4.6 Asymmetric deformation of amoeboid organism

I investigated whether the asymmetric dynamics in the electronic amoeba are in the amoeboid organism. A measurement environment to evaluate the expansion and shrinkage ratio of the amoeboid organism on a nutrient-rich chip is shown in Fig. 4.10(a). The amoeboid organism expands its body to maximize food intake; on the other hand, it shrinks when it is exposed to a light. The temperature was 25 to 27 °C (room temperature) and the humidity was controlled at 85 to 100 % using a humidifier and a humidity sensor. We used a light-emitting diode, VLDB123R-08, to irradiate the amoeboid organism on the nutrient-rich agar plate, where the irradiance was 12.8 mW/cm^2. To evaluate expansion and shrinkage speeds of the amoeboid organism, we measured the moving distance from a neutral position, which was analyzed by time-lapse images.

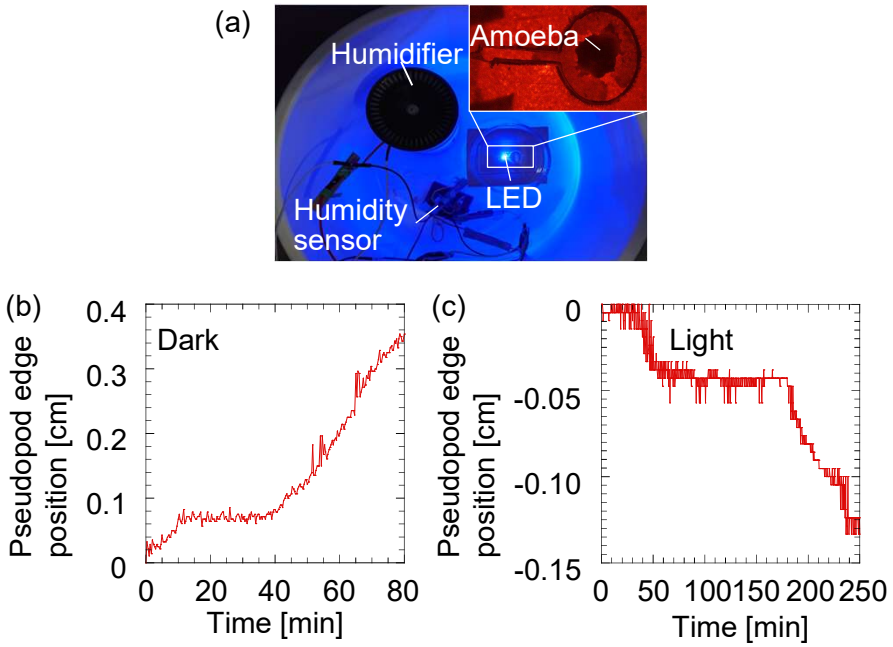

Figure 4.10: (a) Experimental condition to evaluate expansion and shrinkage speed of amoeboid organism. Edge position when (b) unexposed and (c) exposed to light.

Figures 4.10(b) and 4.10(c) show the moving distance of the amoeboid organism

when it was not exposed and exposed to light, respectively. The amoeboid organism fluctuated back and forth, and also it expanded its body when it was not irradiated by the light; on the other hand, it avoided from the light while fluctuating. The amoeboid organism hardly moved during 10 to 40 minutes and 50 to 180 minutes when not exposed to the light and exposed to the light, respectively. The time to decide to move without the light was shorter than that with it. I assumed that this indecisive behavior was that the amoeboid organism was thinking whether to move or not [34]. The average moving distance was 0.146 cm and 0.058 cm per minute without and with light exposure, respectively. The expansion speed of the amoeboid organism was about three times faster than the shrinkage speed.

Incidentally, we assumed that the shrinkage speed, T_S in Eq. 5, was the same when $L_{i,v} = 1$ and $E_{i,v} = 1$ in order to only consider the ratio of the expansion to the shrinkage. As shown in Fig. 4.10, indeed, the speed was different: the amoeboid organism fluctuated on a much faster time scale than it expanded and shrunk. Evaluating the effect of this behavior on the solution search ability remains as a future work.

4.7 Discussion

The obtained results confirm that the asymmetric dynamics are feasible for searching for a solution to the SAT as shown in Figs. 4.7 and 4.8. Here I interpret the roles of the asymmetric dynamics in the solution search as the balanced combination of the local and global searches. In general, the efficiency of the solution search can be controlled by the balance [108, 109]. In the local search, the variables converge at a local (or global) solution; however, they are frequently trapped in a local solution because the optimization problem has many local solutions rather than the global solution. In the global search, on the other hand, the search can avoid sticking in a local solution by using stochastic behavior; however, the stochastic behavior also

makes it difficult for them to converge at a solution since the worst-case complexity becomes $O(2^N)$ if the stochastic behavior is dominant in the solution search.

Above discussions are valid in the asymmetric model: when the error probability E_p is high, the model amoeba can globally search variable vector space while avoiding sticking in a local state. However, in the model with the symmetry, too high E_p leads to vanish an exploration history and the solution search behavior becomes random search, because the variables immediately change from 1 to 0 if the error occurs. On the other hand, in the model with asymmetry, the history does not vanish because they do not immediately change if it occurs; therefore, the model effectively searches a solution even when E_p is too high.

The effect of the asymmetric delays in the response on the solution search is discussed on the analogy of an energy landscape [62, 68]. In the case of the amoeba-inspired computing system for solving the SAT, the global minimum corresponds to the stationary state where the system stabilizes, which means the system finds a solution, and the local minimum corresponds to the steady cyclic state where it moves between several states in a cyclic manner. The barrier height of the potential well is characterized by $(T_E/T_S)UL$ as shown in Fig. 4.11 because increasing T_E or UL (or decreasing T_S) reduces the effect of the error on the solution search. The shape of the well in the model with the asymmetry is relatively deep compared to the model with the symmetry. Here, the depth of the well is proportional to the ratio of the expansion to the shrinkage and the upper bound UL. The error duration time s_e can be considered as a time to climb from the well to escape from a local state. Increasing s_e with low E_p or decreasing s_e with high E_p makes it easier to escape from a local state. By optimizing the balance by the ratio of the expansion to the shrinkage and the magnitude of E_p and s_e, the AmoebaSAT can effectively search the variable vector space globally and locally.

The role of the asymmetry in the solution search is not only to modify the shape of the well but also satisfy a SAT formula. If ignoring the contradiction due to

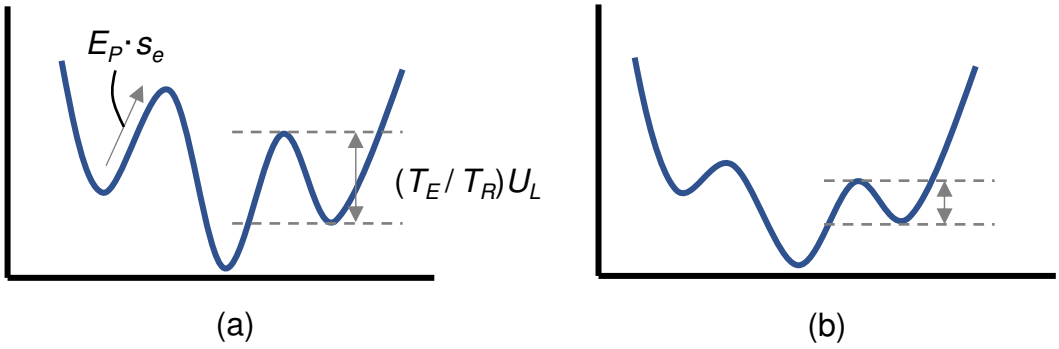

Figure 4.11: Analogy with energy landscape of (a) asymmetric and (b) symmetric models.

introducing the redundant variables, INTRA, the objective function F is satisfied when all redundant variables become 1. The depth of the well as discussed in the above can be controlled by increasing UL and keeping the values of the expansion and shrinkage constant. However, even if adjusting the balance in the symmetric model, the model with the symmetry cannot effectively search variable vector space because redundant variables are biased to 0 by the symmetric dynamics due to the error. Because the asymmetric dynamics bias redundant variables to 1 even when E_p is high, the model with asymmetry is easier to satisfy a SAT formula compared to the symmetric model. We have concluded, from the above discussion, that the AmoebaSAT with the asymmetric dynamics could search a solution compared to original one.

Organisms process information by making their bodies interact with an environment [110–114]. *Caenorhabditis elegans* is a multicellular organism unlike the amoeboid organism asymmetrically behaves when making a decision: if increasing concentration of repulsive odorant 2-nonanonn, *C. elegans* instantly makes the decision to avoid from it, else if decreasing that of 2-nonanonn, it carefully makes a decision [111]. *C. elegans* performs the calculus behavior at a molecular level. The behavior is similar to the amoeboid organism when exposed to light or not. Based on the Darwinism and my results in this chapter, the asymmetric dynamics are considered as an evolution remnant and reasonableness as survival strategy.

The solution-searching performance of the electronic amoeba was low; 0.2 ms was required to solve 12-variable 3-SAT instances even if using the optimal error parameters as shown in Fig. 4.4. There are two solutions to improve the performance. First we should implement the feedback circuit by using an asynchronous circuit. There occurs an operational mismatch between the amoeba core and the feedback circuit: in this chapter, the amoeba core and the feedback circuit operated with an asynchronous and synchronous way, respectively. We can easily implement the bounce-back rule on the feedback circuit by using the FPGA or a crossbar circuit that has a memory at each cross-point because the bounce-back rule consists of AND or OR operations. Second the parameters about the asymmetric dynamics were not optimized. The electronic amoeba is the system that naturally incorporates the asymmetric dynamics because of the capacitor network. Therefore, we can optimize the parameters by adjusting the capacitance and transconductance of the MOSFET in the amoeba core related to the time constant of resistive-capacitive circuits.

From the viewpoint of analog implementation of the electronic amoeba and FPGA implementation of the AmoebaSAT, an overhead occurs in producing a stochastic function [73, 106, 115]. Xorshift, used in the implementation of the electronic amoeba, is a hardware friendly implementation method because it consists of exclusive-OR and bit shift operation [102]. Considering applying the SAT to real-world applications, a numerous number of variables is required; therefore, we should prepare random number generators corresponding to the number of variables. The area of implementing stochastic functions enlarges, leading to reduce the scalability. Therefore, we hope to implement a single stochastic function on a single device. Figure 4.7 indicates that a stochastic analog device, such as a stochastic switching memristor [116–123], is a feasible solution for the electronic amoeba and FPGA-based AmoebaSAT because such a device uses low implementation area and low power consumption although it exhibits finite probability distribution due to the device variation. Random number generators implemented by a stochastic analog

device will enable to reduce the overheads.

4.8 Conclusion

I demonstrated that the electronic amoeba implemented by analog circuits and digital circuits that correspond to the amoeba core and the feedback circuit, respectively, could solve the SAT. I also evaluated the performance of the electronic amoeba to the SAT. The experimental results indicated that there were an optimal error period for each error probability and optimal error parameter sets for the error period and probability that minimized the solution-searching time. In addition, the solution search of the electronic amoeba was uninfluenced by the error period when the error probability was law; on the other hand, it was influenced by the period when the probability was high. We considered these abilities of the electronic amoeba arose from the asymmetric dynamics due to the transconductance of the MOSFET in the branch in the amoeba core and investigated the performance a model amoeba with the asymmetric dynamics by the numerical simulation. The model amoeba showed the robustness to the error property as with the experimental results of the electronic amoeba. Moreover, the solution-searching performance was improved owing to the asymmetry, compared to original one; the gaps when increasing the problem size widened. These results tell us that introducing the asymmetry into FPGA-based amoeba-inspired algorithm is helpful to improve the performance and stochastic hardware random number generators that have the gauss distribution can be used as producing the error in the electronic amoeba. The amoeboid organism and *C. elegans* have such an asymmetry; therefore it is considered to be reasonable way to survive based on the Darwinism.

Chapter 5

Amoeba-inspired all analog computing system integrating resistance crossbar circuit for solving maximum cut problem and traveling salesman problem

5.1 Introduction

The traveling salesman problem (TSP) and maximum cut problem (Max-cut) are one of the optimization problems and belong to nondeterministic polynomial time hard (NP-hard) problems. Their detailed descriptions are shown in Chpt. 2. The TSP and Max-cut have several applications in the real world. In addition, the Max-cut is used as a benchmark problem to compare the performance of the Ising machine [20, 28, 57, 58]; therefore, we can directly compare the performance between the Ising machine and the electronic amoeba by solving the Max-cut by the electronic

amoeba. To solve the TSP and Max-cut by the electronic amoeba, the bounce-back rules for them have to be formulized. So far, the bounce-back rule for the TSP has been formulized by Aono et al. [32, 33], but it for the Max-cut has not been formulized. We can formulize the bounce-back rule for the Max-cut by referring it for TSP.

We have to consider how to implement the bounce-back rule on the electronic amoeba. The bounce-back rule for the TSP needs to operate product-sum operations and thresholding. As with the implementation of the electronic amoeba for solving the satisfiability problem (SAT), we can implement the bounce-back rule on the micro-controller shown in Chpt. 4. However, as we confirmed that the performance of the electronic amoeba to the SAT was low when we implemented its feedback circuit with the micro-controller, the implementation is not feasible for the electronic amoeba that solves the TSP and Max-cut and calculating the product-sum by a central processing unit faces a memory wall problem [124–126]. Therefore, to perform at the electronic amoeba's full potential, we have to implement the bounce-back rule by analog circuits. Then the performance is maximized because the amoeba core and feedback circuit operate in analog domain.

In this chapter, I propose the electronic amoeba integrating the crossbar instance-mapping circuit (crossbar IMC) that operates the product-sum operation and thresholding in parallel based on the bounceback rule. First I formulize the bounce-back rule for Max-cut and implement formulized bounce-back rule on the crossbar IMC, and then I demonstrate the electronic amoeba solves the Max-cut and evaluate the performance. Next I implement the bounce-back rule for the TSP on the crossbar IMC and demonstrate the electronic amoeba solves TSP. Then, I evaluate the performance in solving TSP.

5.2 Bounce-back rule for Max-cut and TSP

5.2.1 Bounce-back signal

The bounce-back signal for the amoeba-inspired computing system that solves the TSP is determined by following equations [32, 33]:

$$L_{ij}(t+\Delta t) = \sigma_{\alpha_1,\beta_1}\left(\sum_{Ul} W_{Vk,Ul}\sigma_{\alpha_2,\beta_2}\left(X_{Ul}(t)\right)\right), \tag{5.1}$$

$$\sigma_{\alpha,\beta}(x) = \frac{1}{1+\exp(-\alpha(x-\beta))}, \tag{5.2}$$

where $W_{Vk,Ul}$ is a variable interaction described in the next section where V and U represent a visit city and k and l represent a visit order. L_{Vk} decides which variables to be decreased; therefore, if L_{Vk} is 1, corresponding variable X_{Vk} is bounced back, i.e., the variable approaches 0. $\sigma_{\alpha,\beta}(x)$ represents a sigmoid function where α decides a slope of the function and β is a threshold value. We can change an instance to be solved by changing $W_{Vk,Ul}$. In the amoeba-based computing system and -inspired algorithm, the parameters are set to $\alpha_1 = 1000$, $\beta_1 = 0.5$, $\alpha_2 = 35$, and $\beta_2 = 0.6$ [32, 33].

5.2.2 Variable interaction for Max-cut

The bounce-back rule for Max-cut has not been formulated. We formulate the variable interaction for the Max-cut by using $2N$ redundant variables for the Max-cut (shown in Appx. B), and it is given by

$$W_{iv,jv'} = \begin{cases} 0.5, & (\text{if } i = j \text{ and } v = 1 - v') \\ \text{edge}(i,j)/\nu, & (\text{if } i \neq j \text{ and } v = v' \text{ and } \text{edge}(i,j) > 0) \\ -\text{edge}(i,j)/\nu, & (\text{if } i \neq j \text{ and } v \neq v' \text{ and } \text{edge}(i,j) < 0) \\ 0, & (\text{otherwise}) \end{cases} \tag{5.3}$$

where $\text{edge}(i,j)$ is the edge between the vertex i and j and ν is a normalization coefficient that makes the amoeba core stable when it finds a solution to the Max-cut. If $X_{i,0} = 1$ and $X_{i,1} = 0$, the vertex i belongs to group 0, else if $X_{i,0} = 0$ and $X_{i,1} = 1$, it belongs to group 1. Therefore if $X_{i,0} = 1$, $X_{i,1}$ is forbidden to become 1, else if $X_{i,1} = 1$, $X_{i,0}$ is forbidden to become 1. The variable interaction to maximize the cut size should be considered: if the edge between the vertexes i and j ($i \neq j$) is positive ($\text{edge}(i,j) > 0$), the total cut size decreases when the vertexes i and j belong to the same set ($v = v'$), else if $\text{edge}(i,j) < 0$, it decreases when the vertexes belong to the set different from each other ($v \neq v'$). A combination of vertexes such as decreasing the total cut size is bounced back as much as possible. As a result, the amoeba-inspired computing system can converge at a solution that the total cut size is relatively large. How to determine the normalization coefficient for Max-cut is given in Appx. B.

5.2.3 Variable interaction for TSP

In Ref. [32, 33], the variable interaction for the TSP is given by

$$W_{Vk,Ul} = \begin{cases} 0.5 & (\text{if } V = U \text{ at } k \neq l \text{ or } V \neq U \text{ at } k = l) \\ \text{dist}(V,U)/\lambda & (\text{if } V \neq U \text{ and } |k-l| = 1) \\ 0 & (\text{otherwise}) \end{cases}, \tag{5.4}$$

where $\text{dist}(V,U)$ is an inter-city distance between the city V and U and ν is a normalization coefficient to make the amoeba core stable when it finds a solution. Eq. 5.4 forbids visiting same city ($V = U$) at the different time ($k \neq l$) and forbids visiting different city ($V \neq U$) at the same time ($k = l$). To minimize the total route, the variable that visits a different city ($V \neq U$) at a time before and after ($|k - l| = 1$) is likely to be bounced back as with the variable interaction of the Max-cut. How to determine the normalization coefficient is given in Appx. B.

5.3 Electronic amoeba integrating crossbar circuit that performs product-sum and thresholding

To evaluate the bounce-back rule for the Max-cut and TSP in the electronic amoeba, the product-sum and thresholding is calculated by analog electronic circuits. Although using the micro-controller such as the electronic amoeba for solving the SAT shown in Chpt. 4 is considered to calculate the product-sum and thresholding, the intrinsic ability of the electronic amoeba will not appear and the feedback circuit faces a memory wall problem [124]. The feedback circuit of the electronic amoeba has to be implemented with analog circuits.

Figure 5.1 shows the electronic amoeba that integrates a crossbar circuit for

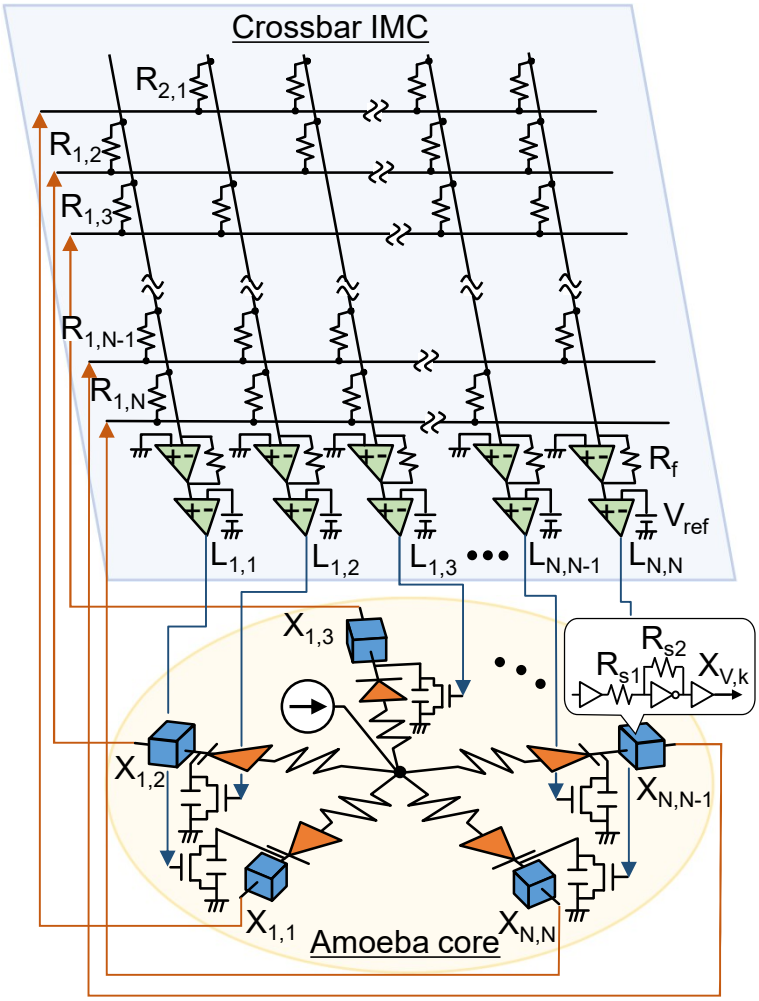

Figure 5.1: Schematic of electronic amoeba with crossbar instance-mapping circuit (IMC). Amoeba core searches solution and operates with asynchronous way. Crossbar IMC performs product-sum operations and thresholding and feeds back to amoeba core. Blue boxes in amoeba core outputs sigmoid-like function.

calculating the product sum by using the Kirchhoff's current law. $L_{ij}(t+\Delta t)$ given by Eq. 5.1 typically takes 0 or 1 due to the sigmoid function when $\alpha_1 = 1000$. Then, the bounce-back signal for the Max-cut and TSP from the crossbar circuit is defined by reformulating Eq. 5.1 as follows:

$$L_{i,j} = \theta\left(\sum_j \frac{R_f}{R_{i,j}} X_j - T\right), \tag{5.5}$$

where $\theta(\cdot)$ is a threshold function that can be implemented with a comparator, R_f is

a feedback resistance of an operational amplifier, $R_{i,j}$ is a resistor at each cross-point in the crossbar, and T is a threshold value representing β_1 in Eq. 5.1. $W_{i,j}$ in Eq. 5.1 corresponds to $R_f/R_{i,j}$, which is derived from $W_{i,j} = R_f/R_{i,j}$. T is a threshold value and controlled by V_{ref} of the comparator. We can change an instance to be solved by changing the resistances $R_{i,j}$ at each cross-point in the crossbar IMC based on Eq. 5.1. We obtain the bounce-back rule for the Max-cut and TSP by changing the subscripts in Eq. 5.5. To implement the bounce-back rule on the crossbar IMC, the variable interaction for the Max-cut and TSP defined by Eqs. 5.3 and 5.4 can be rewritten as $W_{iv,jv'} = R_f/R_{iv,jv'}$ and $W_{Vk,Ul} = R_f/R_{Vk,Ul}$, respectively.

Recently, the crossbar circuit to perform product-sum operations in analog domain has been studied in the area of neural networks [127–133], where memristive switching memory so-called "memristor" is used [134, 135]. Relationship between neural networks and optimization solvers, including Ising machines and the electronic amoeba, is an inverse problem: neural networks learn and optimize synapse weights based on input and output signals [127–133]; on the other hand, in the system dedicated to solving optimization problems, the weights are given by a problem to be solved in advance and they find an optimal input. For example as dedicated systems to solving optimization problems, Hopfield neural networks with the crossbar circuit have been reported [89, 90].

5.4 Results

5.4.1 Circuit simulation results for Max-cut

We conducted circuit simulations by using simulation-program-with-integrated-circuit-emphasis-based analog circuit simulator (LTspice, available in [136]) developed by Analog Devices, Inc. The resistance value of the feedback resistor for the operational amplifier was 10 kΩ. We set R_{s1} and R_{s2} to 10 kΩ and 56 kΩ. Figure 5.2(a)

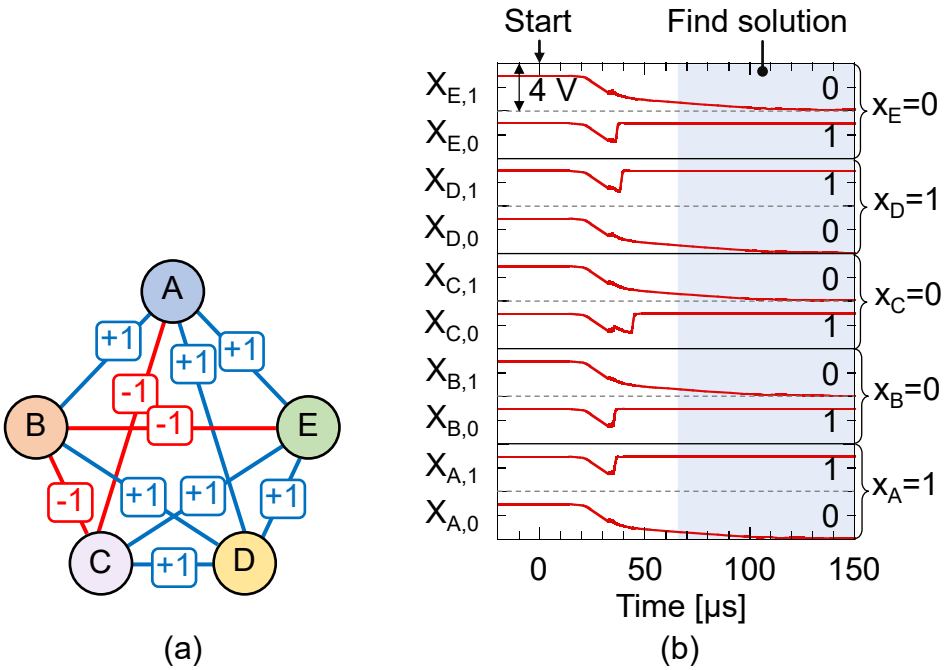

Figure 5.2: (a) Problem to be solved and (b) time evolutions of all variables.

shows the problem to be solved, which is 5-vertex Max-cut and completely connected with +1 or -1. Fig. 5.2(b) shows time evolutions of all variables obtained by the circuit simulation. In the circuit simulation and experiment shown in the next subsection, the amount of the current from the current source and capacitances in the pseudopods in the amoeba core were 25 μA and 100 pF, respectively. All variables started from 3 V because the capacitor charges initially were 0 V, which corresponds to all variables taking 1. These states violate the constraint that the variables should not belong to Gr. 0 and Gr. 1 at the same time; thus, the crossbar IMC sent a signal to turn off the MOSFETs of all branches in the amoeba core. When we injected the current from the current source, each variable approached 0 V, while the crossbar IMC always calculated the product-sum and the thresholding based on resistance values in each cross-point in the crossbar IMC. Then, the crossbar IMC stopped turning off several MOSFETs in accordance with the edge weights and the constraint shown in Eq. 5.3. After the time evolution, the variables reached stationary state, and the electronic amoeba found a solution. Note that if the elec-

tronic amoeba finds a solution to optimization problems, the outputs become stable because there are no variables that violate constraints to optimization problems.

The obtained solution was $(x_A, x_B, x_C, x_D, x_E) = (1, 0, 0, 1, 0)$. That is, the vertexes B, C, and E belong to Gr. 0, and the vertexes A and D belong to Gr. 1. The cut size of the obtained solution was 4, which was one of the optimal solutions to the problem shown in Fig. 5.2(a).

5.4.2 Physical system for solving Max-cut

We implemented the electronic amoeba integrating the crossbar IMC with commercial electronic devices as shown in Fig. 5.3(a). The amoeba core and crossbar IMC were implemented on a breadboard and a surface mount circuit, respectively. The solved problem was the same to Fig. 5.2(a). The feedback resistance of the operational amplifier is the same to the circuit simulation; however, because the accuracy of resistors that are on sale commercially is low, I chose resistance values near to original resistance values.

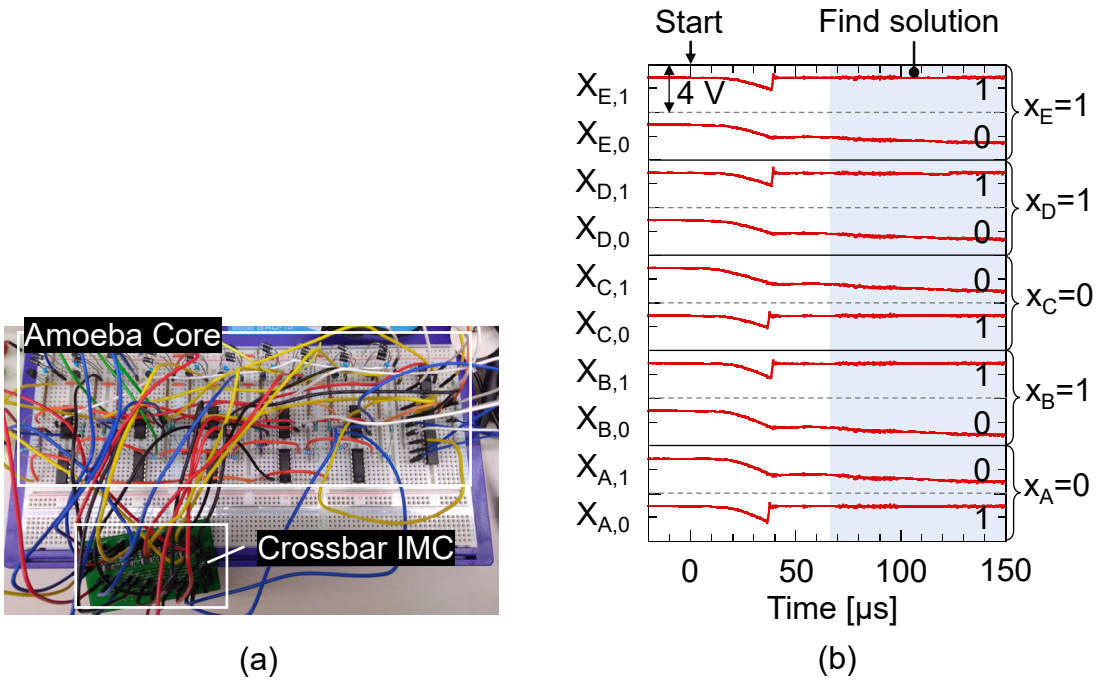

Figure 5.3: (a) Photograph of physical circuit and (b) time evolutions of all variables.

Figure 5.3(b) shows the time evolutions of all variables. The variables started from 3 V as with the circuit simulation. The current from the current source was injected into each branch, and then, the variables approached 0 V. After the time evolution, the variables became stable and the system found a solution to the Max-cut. The solution-searching behavior was the same to the circuit simulation. The obtained solution was $(x_A, x_B, x_C, x_D, x_E) = (0, 1, 0, 1, 1)$. That is, the vertexes A and C belong to Gr. 0 and the vertexes B, D, and E belong to Gr. 1. The cut size of the obtained solution was 4, which was an optimal solution. Although the solution was different one from the circuit simulation as shown in Fig. 5.2(b). We have concluded that the electronic amoeba can solve the Max-cut from the circuit simulation and the experimental result of physically implemented circuits.

After 50 μs, the experimental results showed that the amoeba branches whose output signals became 0 V gradually approached 0 V. This is due to the transconductance of the MOSFETs in the amoeba core and the Kirchhoff's current law. The current from the current source mostly flowed into the MOSFETs whose state was on because the transconductance was relatively large: the current flowed into the branches whose states took 1. Note that the blue square box in Fig. 5.1 outputs inverse sigmoid-like function. Then, in accordance with the Kirchhoff's current law, the current little flows into the MOSFETs whose state is off. Therefore, the variables gradually approached 0 V. We can reproduce the simulation results by decreasing the transconductance of the MOSFET.

5.4.3 Evaluation of solution-searching performance to Max-cut

We investigated the solution-searching performance of the electronic amoeba to the Max-cut by using the circuit simulator. Figure 5.4 shows the simulation results. Tested problems were 5- to 100-vertex Max-cut whose connectivity was complete and

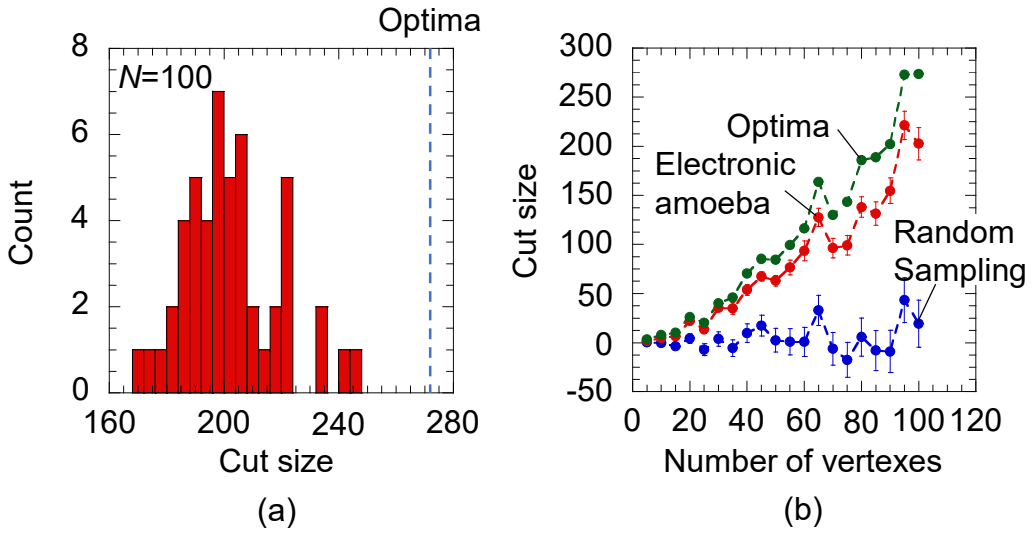

Figure 5.4: (a) Histogram as function of cut size when electronic amoeba solves 100-vertex Max-cut and (b) solution-searching performance, as function of number of vertexes.

the weights were assigned by $-1.0, -0.6, -0.2, 0.2, 0.6$, and 1.0 randomly. I tested 100 trials for each instance. For each instance, the resistance values in the amoeba core were randomly assigned from 1 Ω to 1 kΩ, which changed initial values when the variables bifurcated and therefore it made the system search several solutions. The current from the current source and capacitances in the pseudopods were $5N$ μA and 100 pF, respectively, where N is the number of vertexes. As shown in Fig. 5.4(a), the electronic amoeba implemented on the circuit simulator found several solutions owing to randomly assigning the resistance values. In this circuit simulation, the electronic amoeba did not find an optimal solution.

Figure 5.4(b) shows the cut size of obtained solutions as a function of the number of vertexes. The electronic amoeba found better solutions compared with the average solution obtained by the random sampling from 10000 trials that produces a solution to the Max-cut randomly, and the quality did not degrade against the average results of the random sampling. However, the obtained solutions by the electronic amoeba degraded against the optimal solutions. Methods to improve the quality of solutions obtained by the electronic amoeba are needed (shown in Chpt. 6 and

Appx. C). Figure 5.5 shows a time in the circuit simulation for the electronic amoeba to find a solution. The electronic amoeba found a solution to the Max-cut up to 100 vertexes in nearly constant time. It is needed to evaluate the performance to a large-scale problem and the performance evaluation of the physical implemented circuit is needed to confirm whether the search time is constant.

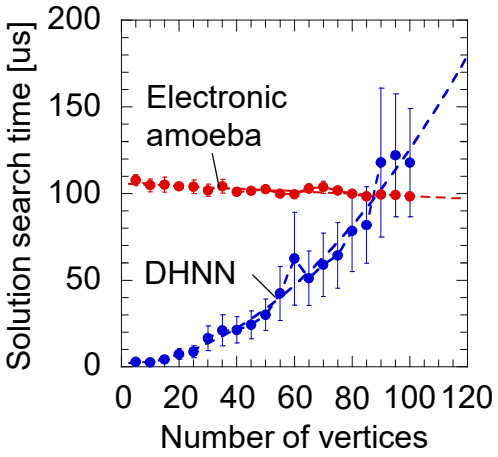

Figure 5.5: Solution-searching time. Red and blue dots are the results of the electronic amoeba and discrete-type Hopfield neural network, respectively.

I examined that the trend whether the electronic amoeba finds a solution to the Max-cut in constant time shown in Fig. 5.5 was universal or not by numerical experiments. I formulized a simple numerical model of the electronic amoeba as follows:

$$X_{i,v}(t+1) = \begin{cases} X_{i,v}(t) + D_{in}N/L_{off} & (\text{if } L_{i,v}(t+1) = 0) \\ X_{i,v}(t) - D_{out}\sigma_{20,0.6}(X(t)) & (\text{if } L_{i,v}(t+1) = 1) \end{cases}, \tag{5.6}$$

where D_{in} and D_{out} are the amount of supply and withdraw resources, respectively, L_{off} emulates the current conservation in the amoeba core and is defined as the total number of $L_{i,v} = 0$, N is the number of vertices, and $\sigma_{\alpha,\beta}$ is a sigmoid function having a slope α and a threshold value β. $X_{i,v}$ represents a state variable of the Max-cut. $L_{i,v}$ determines which the MOSFETs in the amoeba core should be turned

on and is defined as follows:

$$L_{i,v}(t+1) = \sigma_{1000,0.5}\left(\sum_{jv'} W_{iv,jv'}\sigma_{35,0.6}(X_{j,v'})\right). \tag{5.7}$$

Figure 5.6 shows the numerical simulation results when I iteratively solve Eqs. 5.6 and 5.7. In this experiment, the parameters, Δ_{in} and Δ_{out}, were set to 0.0004 and 0.0010, respectively. Initial values of variables were set in the range of $[0, 10^{-3}]$ randomly with the offset 0.59. The number of vertexes of solved instances was from 100 to 5000 and weighted in the range of $-1.0, -0.6, -0.2, 0.2, 0.6$, and 1.0. As shown in Fig. 5.6(b), the cut size was increased when the number of vertices was increased. This trend was the same to the circuit simulation results (shown in Fig. 5.4). On the other hand, solution-searching time (iterations to find a solution) slightly increased when solving over 2000-vertex Max-cut although it decreased when solving under 2000-vertex Max-cut. Therefore, we should conclude that the time to find a solution in the electronic amoeba would grow when solving large vertex instances, although the rate of the time as a function of the number of vertexes was small. In addition,

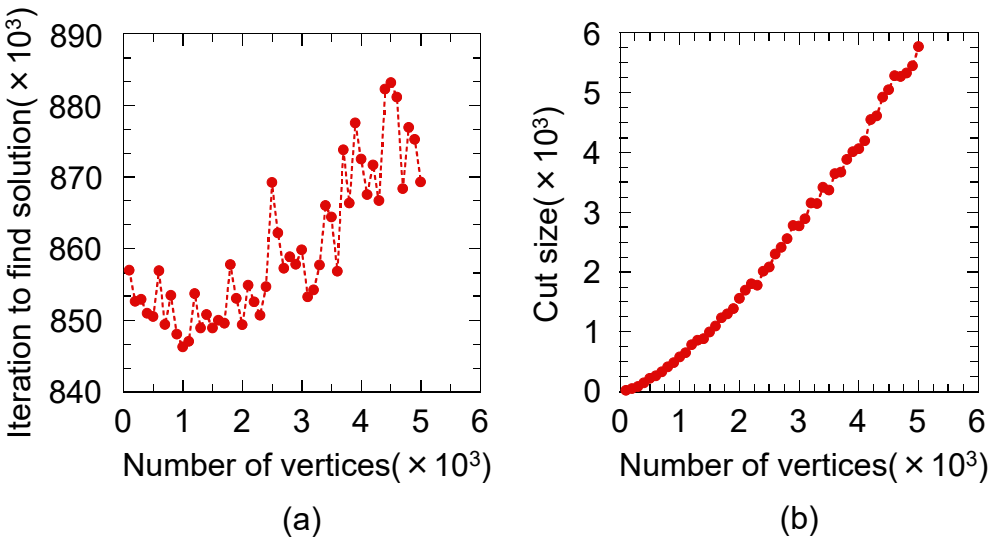

Figure 5.6: Numerical simulation results using formulized behavior of electronic amoeba. (a) Iterations to find solution and (b) cut size, as function of number of vertices.

a physically implemented electronic amoeba has a time delay in its operations as discussed in Chap. 6, the delay makes the search time longer.

We compared the performance of the electronic amoeba with the discrete-type Hopfield neural network (DHNN) [59, 137, 138]. The DNHH is a greedy search method if it does not accompany the simulated annealing (SA). The DHNN with the SA is the solution-searching principle of the digital annealer and the CMOS annealing, which have been developed by Fujitsu Ltd. and Hitachi Ltd., respectively [20, 57]. The DHNN algorithm is shown in Alg. 2, where the energy corresponds to the cut size; if the energy decreases, the cut size increases.

Algorithm 2 Discrete-type Hopfield neural network algorithm

```
1: INITIALIZE: Assign randomly -1 or 1 to X, E ← ReturnEnergy(X)
2: while Cut size of DHNN is smaller than average one of electronic amoeba do
3:    rn ← rand(0, N − 1)
4:    ΔE ← 0, i ← 0
5:    for i < N do
6:       ΔE ← ΔE + edge[rn][i] × X[i]
7:       i ← i + 1
8:    end for
9:    ΔE ← −2 × X[rn] × ΔE
10:   if ΔE ≤ 0 then
11:      X[rn] ← −X[rn]
12:      E ← E + ΔE
13:   end if
14: end while
```

We implemented the DHNN on a conventional computer (Intel Xeon processor E5-1650 v2 @3.5 GHz). In Fig. 5.5, the blue dots show the computation time for the DHNN to find a solution whose quality is comparable to an average quality of solutions obtained by the electronic amoeba. We tested 100 times for each instance. The computation time in the DHNN increased with $O(N^2)$. The computer simulation results indicate that the solution-searching performance of the electronic amoeba is superior to that of the conventional computer when solving more than 90-vertex Max-cut.

There are two reasons that the computation time increased in the quadratic

trend. One is that in the line numbers 5–8 in the DHNN algorithm shown in Alg. 2, the computational cost of the product-sum operation increases in $O(N)$ with increasing the number of vertexes, N. Figure 5.7(a) shows the computation time for the product-sum operations in the DHNN algorithm per iteration. The computation time increased in $O(N)$ when the number of vertexes was larger than 30. This is because the number of the product-sum operations increases as a function of N. In the electronic amoeba, the computational cost about the product-sum operation is constant even when the number of vertexes N increases because the product-sum operations between the variables and the variable interactions are conducted by the crossbar IMC with the operational amplifier and the comparator in a parallel fashion. increasing The second reason is that the number of iterations for the DHNN

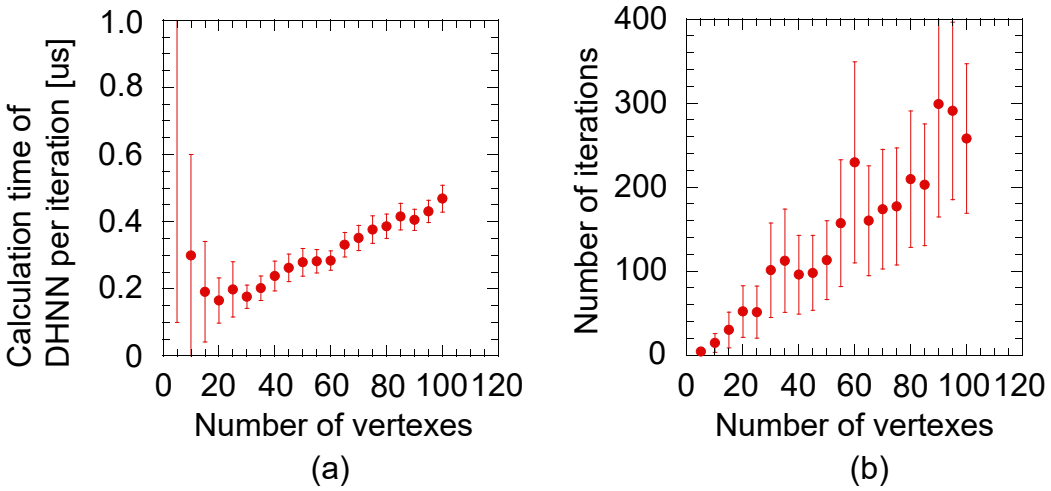

Figure 5.7: (a) Calculation time of DHNN in conventional computer per iteration and (b) number of iterations for DHNN to find solution comparable to average one of electronic amoeba, as function of number of vertexes.

to find a solution comparable to the average one of the electronic amoeba increased in $O(N)$. Figure 5.7(b) shows the iteration number as a function of N. The iteration number increased in $O(N)$. The initial values were randomly assigned by -1 or +1, which was the same to a solution produced by the random sampling. Then the DHNN tried to improve the solution of the random sampling step by step. Therefore, the number of iterations for the DHNN to find a solution comparable to the elec-

tronic amoeba one increased in $O(N)$, while the solutions obtained by the electronic amoeba did not degrade against the random sampling. From the first and second reasons, the computation time of the DHNN increased in $O(N) \times O(N) = O(N^2)$.

5.4.4 Circuit simulation results for TSP

To confirm that the electronic amoeba integrating the crossbar IMC can solve the TSP, I first used the circuit simulator as with the experiment of solving the Max-cut. The overall description of the TSP and the variable encoding for the bounce-back rule for the TSP are shown in Chpt. 2 and Appx. B, respectively. Solved instance is shown in Fig. 5.8(a). Figure 5.8(b) shows time evolutions of all variables when solving the 4-city TSP instance. In the circuit simulation, the capacitances in the amoeba core and the amount of the current from the current source were 500 pF and 80 μA, respectively. The number of required variables to solve the 4-city TSP instance are 16 (4×4). All variables started from 3 V as with the electronic amoeba for solving the Max-cut. These states violated the constraints of TSP that a salesman should visit all cities only once and should not visit the city at the same time. Therefore, the crossbar IMC sends a signal to the MOSFETs in the amoeba core to turn off them. When the solution search started, the variables gradually approached 0 V. Then, the crossbar IMC released the variables that tries to become 0 V, and they immediately reached 3 V. After the feedback process, the variables converged at the stationary state in the same way as the electronic amoeba found a solution to the Max-cut (shown in Fig. 5.2), and the states correspond to a solution to the TSP. The obtained solution was $D \rightarrow A \rightarrow B \rightarrow C \rightarrow D$, the route length was 100, which is the optimal solution to the problem to be solved.

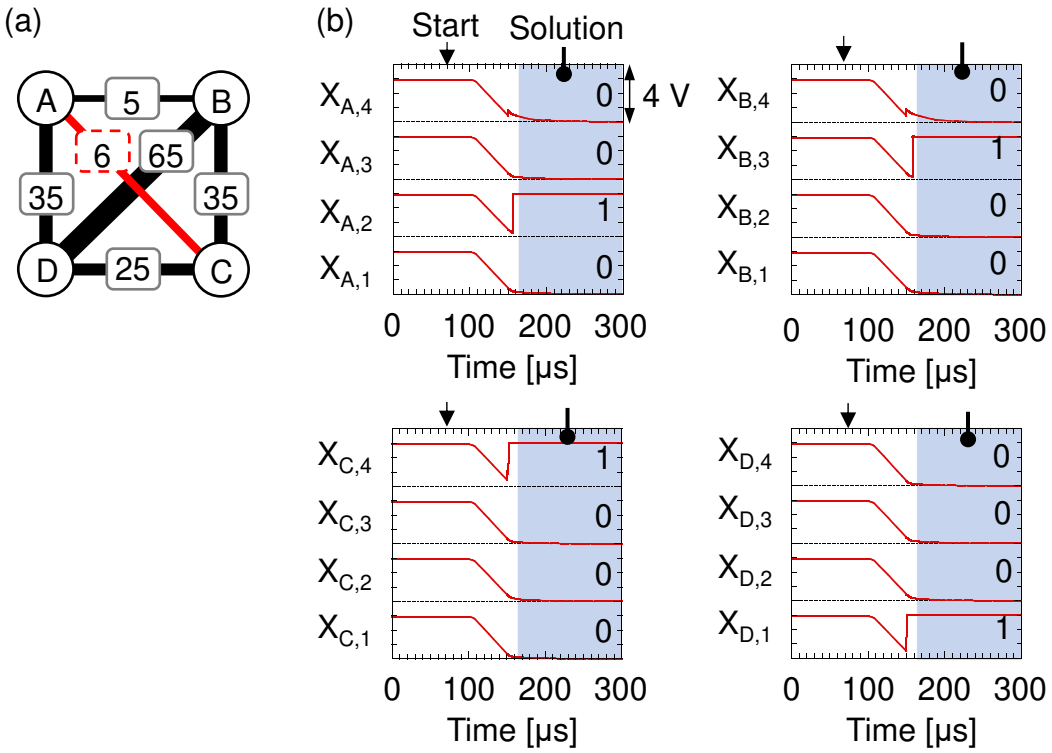

Figure 5.8: (a) TSP instance to be solved and (b) time evolutions of all variables obtained from circuit simulator when solving 4-city TSP instance.

5.4.5 Physical implemented system for TSP

I implemented the electronic amoeba integrating the crossbar IMC on the bread-board as shown in Fig. 5.9(a). In the fabricated system, the capacitance in each pseudopod in the amoeba core were 470 pF, which is the sum of parasitic capacitances and a discrete capacitor. The current of the current source was 80 μA. Figure 5.9(b) shows time evolutions of all variables. The variables started from 3 V, and when the current was injected into the amoeba core, the variables gradually approached 0 V. They became stable after the feedback process in the same way as the electronic amoeba solved Max-cut and TSP by the circuit simulator and the physical-implemented circuit. The obtained solution was $A \rightarrow D \rightarrow C \rightarrow B \rightarrow A$, and the total route was 100 which was the optimal solution.

I conducted that the physically implemented electronic amoeba could solve different problems (shown in Fig. 5.9(a)) by only changing the resistances on each

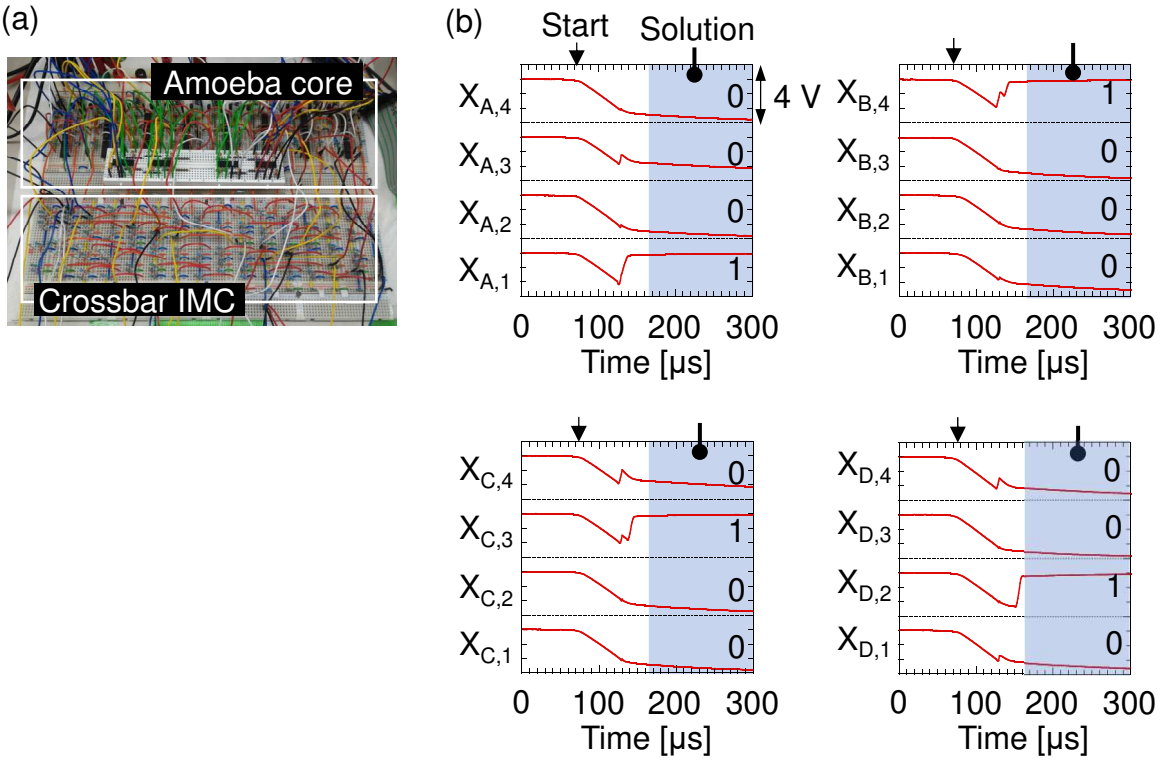

Figure 5.9: (a) Photo of implemented circuit and (b) time evolutions of all variables obtained from physical-implemented circuit when solving 4-city TSP instance.

cross-point in the crossbar IMC. Figure 5.10 shows the experimental results, where the upper of the figure is the problem to be solved and the bottom of the figure is the histograms. The optimal solutions and the worst solutions are summarized in Tab. 5.1. I confirmed that the fabricated electronic amoeba found the optimal solution for the instances A-C and E. However, the system could not find the optimal solution for instance D as shown in Fig. 5.10(d), and the counts of obtained solutions are different for the two optimal solutions, even though the route lengths are in the same (Fig. 5.10(c)). These are attributed to variations in the fabricated circuit: for examples, threshold voltage variation in the CMOS inverter, offset voltage variation in the operational amplifier, and difference in the wiring length in the crossbar IMC. As a result, the electronic amoeba has likely to find the route, $A \rightarrow C \rightarrow D \rightarrow B \rightarrow A$. The electronic amoeba has searched the optimal solution for the instance that the total route length is widely distributed such as the instance

E.

Table 5.1: Summary of TSP instances and route length

	Ins. A	Ins. B	Ins. C	Ins. D	Ins. E
Rt. 1: $A \to B \to C \to D \to A$	100	100	100	100	100
Rt. 2: $A \to C \to D \to B \to A$	96	99	100	101	160
Rt. 3: $A \to D \to B \to C \to A$	136	139	140	141	200

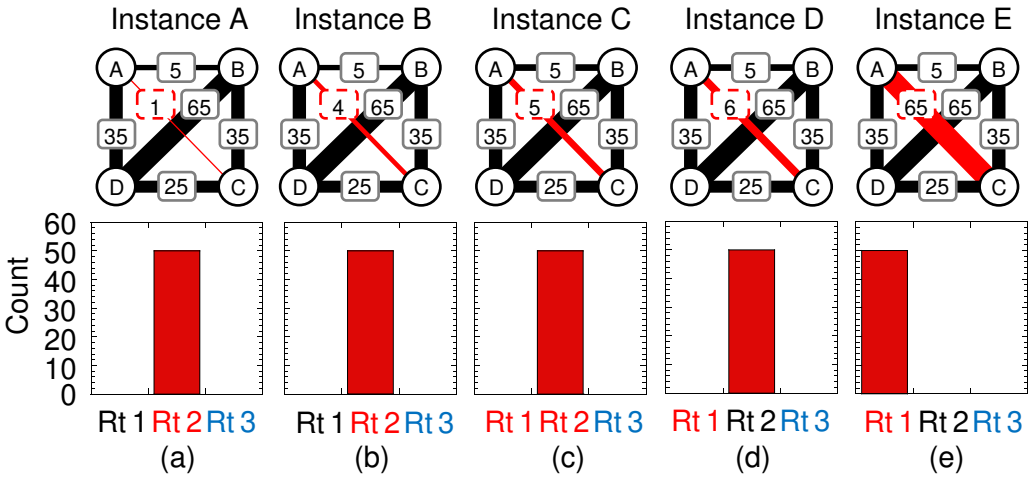

Figure 5.10: Histogram when solving several 4-city TSP instances. (a) Instance A, (b) instance B, (c) instance C, (d) instance D, and (e) instance E.

5.4.6 Evaluation of solution-searching performance to TSP

I investigated the solution-searching performance of the electronic amoeba by the circuit simulation when solving the 10–30 city TSP instances. I tried 50 times for solving the 10–20 city instances and only once for solving more than the 20-city instances because the run time in the simulation increased rapidly: 6 days were spent to simulate the 30-city instance, although 5 hours were spent to do for the 20-city instance. In each trial, resistances in the branches in the amoeba core were randomly assigned from 1 Ω to 10 kΩ to make the electronic amoeba explore the solution space in the same way as we evaluated the performance to Max-cut. Figure 5.11(a) shows

the circuit simulation results when solving 20-city TSP instance. The capability to search the solution space arose in the electronic amoeba by introducing variations in the randomly assigned resistances in the branches, where different solutions were obtained even for the same instance. The randomly assigned resistance values vary the speed of the state transition from 1 to 0 between the variables. This process does not guarantee to reach the optimal solution, but it can lead the electronic amoeba to reach a variety of solutions.

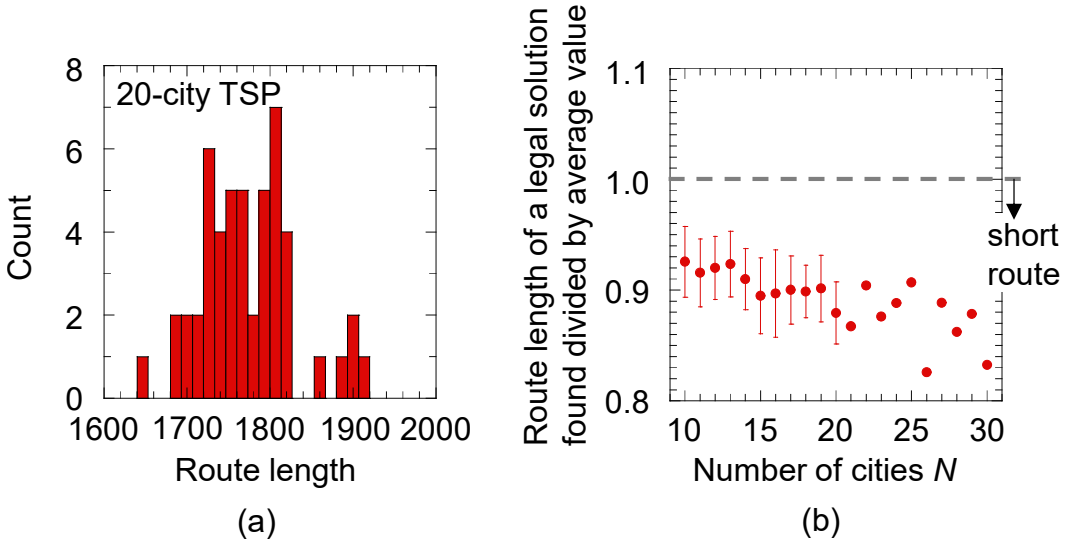

Figure 5.11: (a) Histogram of quality of solutions obtained by electronic amoeba when solving 20-city TSP instance. (b) Solution quality as function of number of cities. Vertical axis denotes route length of electronic amoeba divided by average route length obtained by random sampling from 10000 trials.

Figure 5.11(b) shows the route length obtained by the circuit simulation as a function of the number of cities. The intercity distance of used instances was randomly produced, where the average intercity distance and the standard deviation of used instances were 100 and 17, respectively. In Fig. 5.11(b), the vertical axis is normalized by the average route length obtained from the random sampling in 10000 trials: if a value on the vertical axis is less than 1.0, the solution quality is better than that of the random sampling. The result indicates that the electronic amoeba finds solutions better than the average. The average of solutions was on a

declining trend: the solution quality did not degrade against the average solution when increasing N. We have also confirmed that the electronic amoeba did not reach an illegal solution when it became the steady state such as the hatching area in Fig. 5.8(b).

Figure 5.12 shows the time in the circuit simulation to find a solution. The electronic amoeba could find a solution in linear time growth with the increase in the number of cities. The electronic amoeba will find a better solution in a short time when solving large size problems. A possible interpretation of the linear growth is that the system decides the path one by one, avoiding reaching an illegal candidate. The system searches a relatively shorter path by watching all variables. Once the crossbar IMC determined which variable should be 1 (determine a city that should be visited), the variable is fixed at 1 and the crossbar IMC forbids the variables that violate constraints of the TSP in accordance with the bounce-back rule that is designed so that the constraints are certainly ensured. Therefore, the search space decreases to $N^2 - O(N)$, and the reduction continues until the system finds a solution. The number of reductions is estimated as $O(N)$; therefore, the system finds a solution even when increasing the number of cities. The design of the bounce-back control produced by the crossbar IMC and the amoeba core that operate in parallel are also contributed to the growth because their circuits are implemented by analog circuits.

Figure 5.13 shows a flowchart of solving TSP with linear time growth against N, discussed in the above. We first assign random initial values to all variables in the range of $[-0.005, 0.005]$ with the offset 0.6 to avoid deadlock and reach several kinds of solutions. We define a step that counts the number of repetitions. We compute L_{Vk} for all V and k, where $W_{Vk,Ul}$ is defined by Eq. 5.4. Next, we search the subscriptions i and j about L_{ij} such as the minimum value of $\boldsymbol{L_{Vk}}$, which corresponds to the path that is assumed to be the shortest one under given X_{Vk}. Then, we substitute 1 for the corresponding variable X_{ij}, because the variable whose

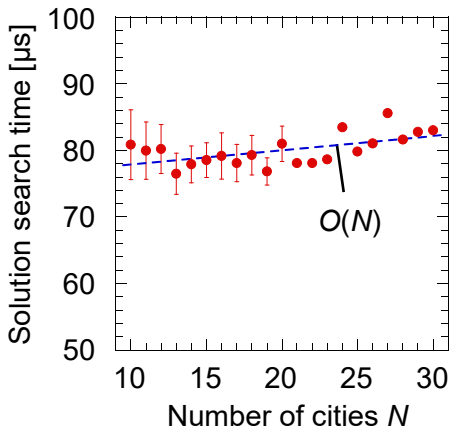

Figure 5.12: Time for electronic amoeba to find solution.

L_{Vk} is minimum is most unlikely to be bounced back, and we substitute 0 for the variables that violate constraints when $X_{ij} = 1$. The variables assigned to 0 or 1 are fixed and never flipped. The solution space that the electronic amoeba is able to search decreases to N^2-2N-1. If we repeat these operations N times, we can assign all N^2 variables and obtain a solution only in linear time as increasing the number of cities. One of the differences between this algorithm and the solution search behavior of the electronic amoeba is that X_{Vk} does not change 1 or 0 immediately.

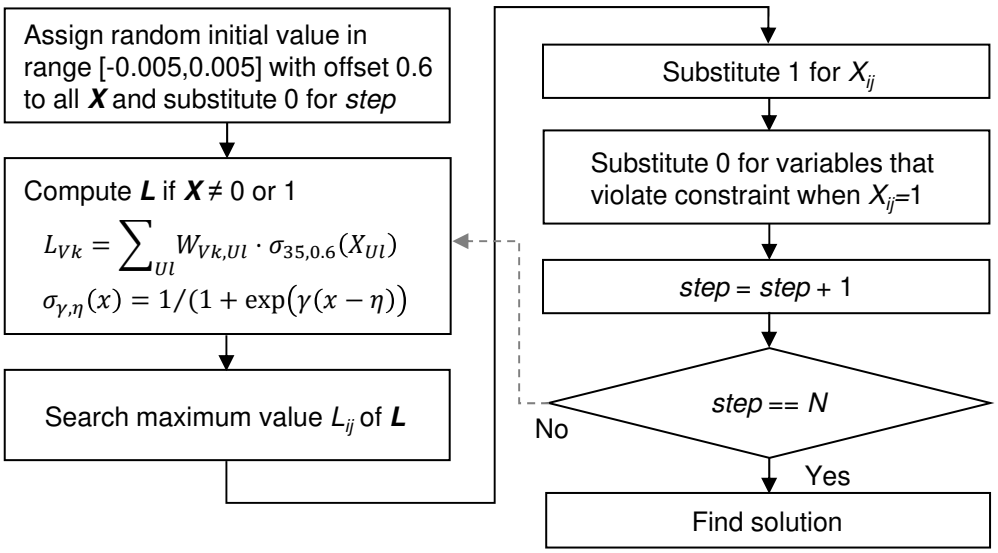

Figure 5.13: Algorithm for solving TSP with linear time growth against N.

Figure 5.14(a) shows the numerical simulation results when solving 10 to 30 cities

instances, which are the same instance to those of the circuit simulation results. The results indicate that the solution quality obtained from the algorithm is comparable to that of the electronic amoeba as shown in Fig. 5.11(b), and the number of steps increases in linear time against N as shown in Fig. 5.14(b). Therefore, we have concluded that the electronic amoeba may have searched the solution such as the algorithm.

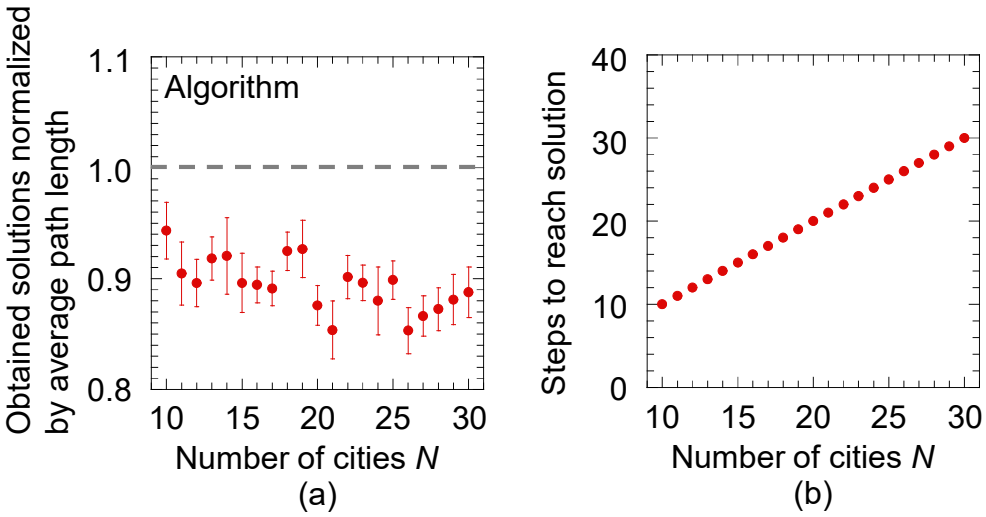

Figure 5.14: (a) Solution quality and (b) steps to find solution obtained from algorithm shown in Fig. 5.13.

To clarify the solution search performance of the electronic amoeba, I compared it with other algorithms. There exist various TSP approximation algorithms, such as the simulated annealing (SA), genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO), and k-opt [139–143]. I chose the probabilistic 2-opt algorithm (shown in Alg. 3) for the comparison because it is a very simple and fast algorithm and complex parameters setting is not necessary. Iteration of the

Algorithm 3 2-opt algorithm

1: Set initial solution $\boldsymbol{X}$ randomly
2: **while** route length of 2-opt is larger than average one of electronic amoeba **do**
3: Choose arbitrary two cities from solution $\boldsymbol{X}$
4: Exchange selected cities and set new solution as $\boldsymbol{X'}$
5: Compare $\boldsymbol{X}$ with $\boldsymbol{X'}$ and set better solution as $\boldsymbol{X}$
6: **end while**

2-opt was made using a commercial computer until the solution quality of the 2-opt reached the average solution quality found by the electronic amoeba. The 2-opt was implemented on a conventional computer (Intel Xeon processor E5-1650 v2 @3.5 GHz). Figure 5.15 shows the simulation results. Approximated curve in Fig. 5.15 is derived by the quadratic function. Figures 5.12 and 5.15 indicate that if the electronic amoeba is implemented physically, it exceeds the 2-opt on a conventional computing system when solving more than 50-city TSP instance.

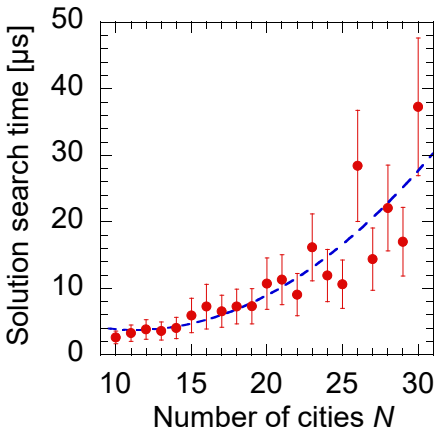

Figure 5.15: Time for 2-opt to find solution comparable to average one of electronic amoeba.

The solution-searching time of the electronic amoeba highly depends on the capacitances in the amoeba core and the current of the current source: we can accelarate the solution search of the electronic amoeba by decreasing the capacitances in the amoeba core or increasing the amount of currents from the current source. In the electronic amoeba, the transition time of the state variable between the two states, 1 and 0, depends on the current injected from the current source and the capacitance value in the pseudopod; therefore, the solution search time depends on these parameters. We confirm these points using the circuit simulation. I used the 10-city instance that is the same instance to the circuit simulation shown in Fig. 5.11 and it was tried 50 times, where the resistance values in the pseudopod were randomly assigned in the range between 1 Ω and 10 kΩ. Figures 5.16(a) and 5.16(b) are

the solution-searching time and the route length of obtained solutions as a function of the current of the current source, respectively. Those of the capacitances in the amoeba core dependences are shown in Figs 5.16(c) and 5.16(d), respectively. The solution-searching time decreased by both increasing the current and decreasing the capacitances. The results suggest that the solution-searching time scales the current of the current source and the capacitance down. On the other hand, the solution quality slightly degraded when increasing the current or decreasing the capacitances as shown in Figs. 5.16(b) and 5.16(d). This is because the effect of the variations of resistances in the amoeba core increases relatively, when the current is increased and/or the capacitance is decreased. Therefore, we can eliminate the degradation of the solution quality by reducing the resistance variation.

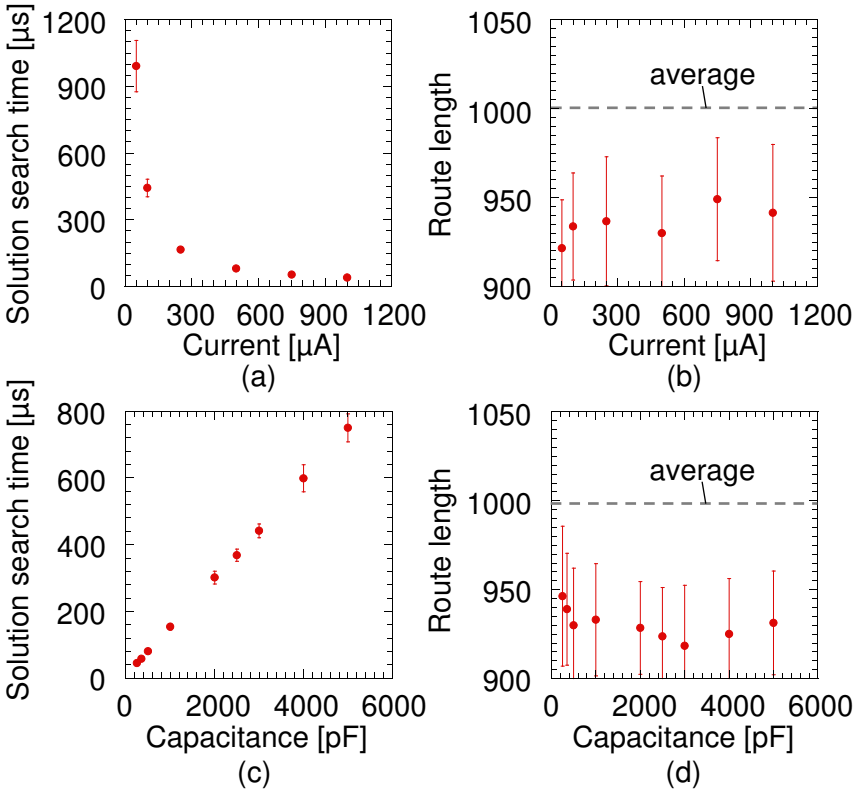

Figure 5.16: Dependence of solution search time on current and capacitance value. (a) and (b) Solution search time and solution quality as function of currents, respectively. (c) and (d) Solution search time and solution quality as function of capacitances in pseudopod, respectively.

5.5 Discussion

We have demonstrated the solution search capability of the electronic amoeba integrating the crossbar circuit to the Max-cut and TSP. The problem to be solved can be changed by changing the resistances in the crossbar IMC. In solving the Max-cut and TSP, the quality of solutions was better than the average quality of solutions obtained by the random sampling, and the time to find a solution was smaller than a conventional computer when the electronic amoeba is physically implemented. The capability of the electronic amoeba exceeds the conventional system when hundred or more cities should be visited.

Our results show that the electronic amoeba is suitable for an application that the user needs the better legal solution in a short time. Users have to select a better legal solution after several trials under the different initial states, since the solution search time of the electronic amoeba growing in $O(1)$ for the Max-cut and $O(N)$ for the TSP makes it possible to retry searching other solutions within a limited time.

Weights for the crossbar IMC are attached to satisfying constraints to optimization problems. Practical applications in the real world require to satisfy constraints. The electronic amoeba takes care of satisfying constraints to optimization problems; on the other hand, Ising machines are not good at solving strictly constrained problems when they have little time because converged spin-alignment sometime represent an illegal candidate that does not satisfy the constraints (detailed discussion is shown in Appx A). Therefore, the electronic amoeba is suitable for such problems.

Owing to the obtained results of the electronic amoeba and the characteristics that the Ising machine converges at an illegal solution and has the laboriousness of the problem mapping due to the sparse connectivity, the electronic amoeba is suitable for the prompt search and has advantages in flexibility and adaptivity to the change of solved instances.

The crossbar IMC implementing the operational amplifiers and comparators has several problems. The operational amplifier which calculates product-sum operation make error due to production tolerance: for example, an offset voltage of the operational amplifier. Summing resistances at each cross-point in the crossbar IMC increases to infinity in mapping Max-cut when the problem size increases. Because the resolution of resistances is limited, we cannot map the large size of the Max-cut on the crossbar IMC. A resistance value (or conductance) in the crossbar IMC cannot be controlled precisely due to intrinsic error when we use the memristor [90, 144]. If the resistances are not precisely controlled, the intercity distance and edge weight are wrongly set. There is a problem of power consumption of operational amplifier because operational amplifier regularly flows current in contrast to the CMOS logic gate. These remain future works.

There is another method for implementing the crossbar IMC without using operational amplifier. A time-domain product-sum operation circuit has been proposed based on spiking neuron model [145–147]. This circuit computes product sum using the slope of resistances: if the resistance in a crossbar circuit is large, the slope is gentle, otherwise it is steep. Low resistance induces frequently-firing and high resistance value induces infrequently-firing based on Ohm's law. Therefore, in the case of implementing the crossbar IMC by the time-domain operations, the crossbar IMC informs the bounce-back signal to the amoeba core at the time domain. Variables that tend to violate constraints or decrease a cost function are frequently bounced back; on the other hand, other variables are infrequently bounced back. As a result, if the amoeba branch is frequently bounced back, its state is likely to become 0, otherwise it is likely to become 1. The bounce-back signal is produced at the frequency domain. Such implementation can remove the problem from the crossbar IMC.

5.6 Conclusion

I successfully demonstrated that the electronic amoeba integrating the crossbar circuit could solve the Max-cut and TSP by both the circuit simulations and experiments. I also evaluated the performance of the electronic amoeba by using the circuit simulator as increasing the problem size: the solution-searching performance of the electronic amoeba exceeds that of a conventional computer when it solves over 90-vertex Max-cut, and the performance in solving TSP exceeds it when solving over 50-city TSP, if the electronic amoeba is implemented by the electronic circuit. The electronic amoeba is superior to Ising machines in terms of finding a legal solution because the bounce-back rule certainly makes the electronic amoeba find a legal solution. Moreover, the crossbar IMC enables the electronic amoeba to have the robustness to the sudden change of problems because the problem can be changed only by changing the resistances at each cross-point. The compact mapping of the instance using the resistor crossbar indicates that the system-on-chip by the semiconductor LSI technology will enhance the potential of our system and the activities of ourselves.

Chapter 6

Exploiting delayed feedback to improve solution quality of traveling salesman problem

6.1 Introduction

One of the challenges of the electronic amoeba is to improve the quality of solutions. One needs a high-quality solution by the sacrifice of time: for example, optimization of transportation network whose construction will not change. I demonstrated that the electronic amoeba could find a solution to the SAT, Max-cut, and TSP shown in Chpts. 4 and 5. To improve the quality of SAT solutions is not necessary because the SAT has only the satisfiable solution: that is, we are not interested in the "quality" in solving the SAT. However, NP-hard problems such as the Max-cut and TSP should improve the quality of obtained solutions. Usually, algorithms for the optimization problem uses a stochastic behavior [139–143]. Local optima freeze the solution search of the algorithms; the stochastic search makes the search escape from local optima. In genetic algorithm, the divergence of individuals (solutions) is increased by the mutation that randomly changes the genetic code [140, 148].

Ising machine utilizes methods to improve the solution quality such as simulated annealing (SA) or quantum annealing shown in Chpt. 3 [68, 70].

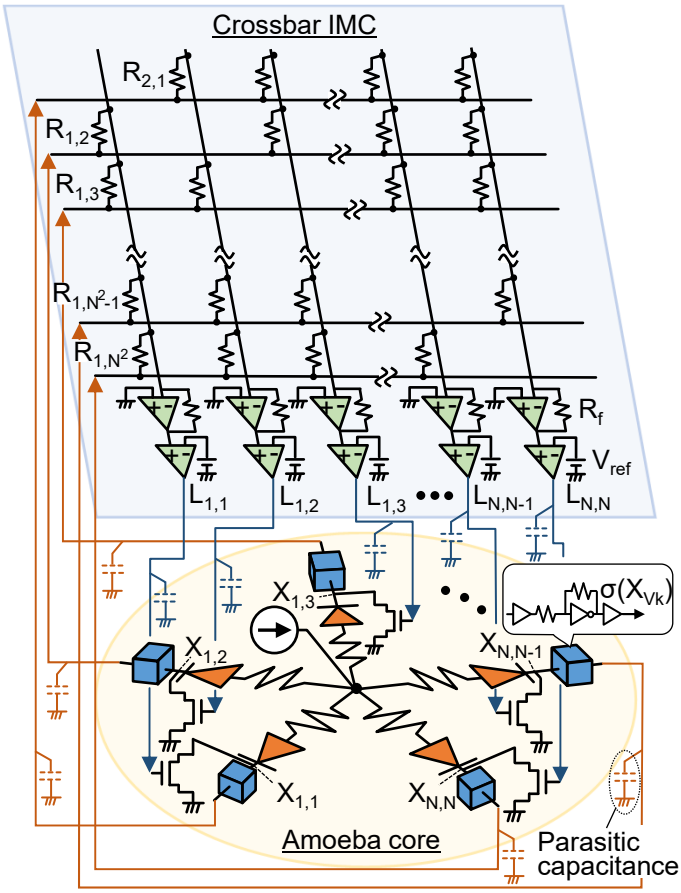

Figure 6.1: Delayed input/output signals in electronic amoeba due to parasitic capacitances.

I demonstrated that genetic-algorithm-based and fluctuation-induced methods improved the quality of solutions obtained by the electronic amoeba (shown in Appx. C). However, their methods require a communication overhead between the electronic amoeba and a conventional computer, or additional circuitry to produce fluctuation, which degrade the solution-searching performance and the scalability. Therefore, we have to develop another method to improve the quality of solutions.

The clue to improve the quality is in the solution-searching behavior of the electronic amoeba. Because the electronic amoeba operates with an asynchronous way, there is an analog-circuit-specific problem. In the electronic amoeba, the output

signals from the amoeba core and feedback circuit are delayed due to parasitic resistor-capacitor time constants in analog circuits as shown in Fig. 6.1; therefore, the state variables of the electronic amoeba oscillate. The experimental results for solving the Max-cut and TSP shown in Chap. 5 were optimized in terms of decreasing the time delay such as resistances in the circuit of the sigmoid function. Figure 6.2 shows the experimental results in solving the Max-cut and TSP when the resistances in the sigmoid function is relatively large. The variables oscillated, and then they became stable, which corresponds to finding a solution. A time for the electronic amoeba with the delay to find a solution grows longer, compared with the case without the delay, because the electronic amoeba reaches a solution when the variables become steady state. Therefore, one may consider hoping to exclude the delay as much as possible from the circuit.

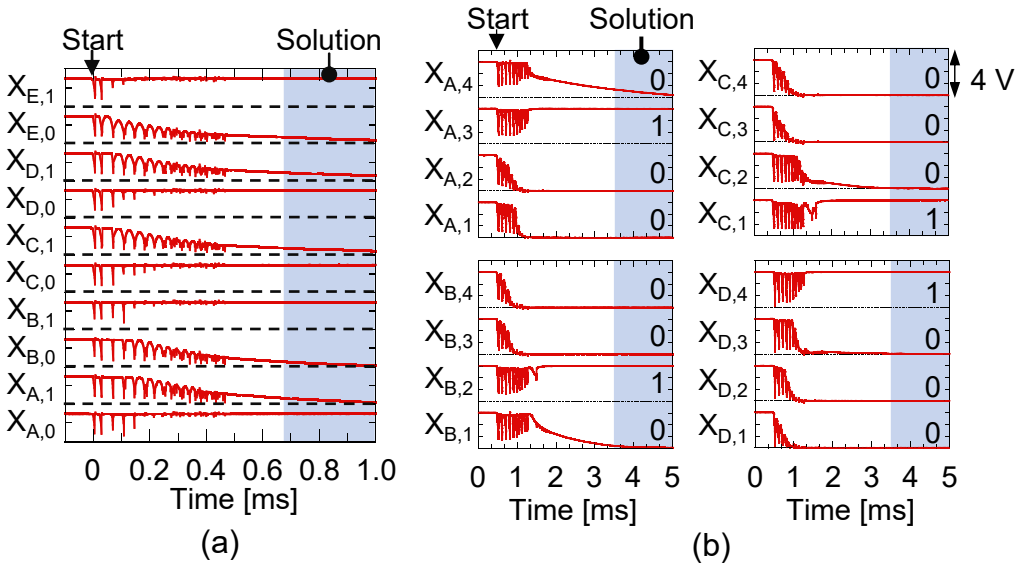

Figure 6.2: Output waveforms of delay-induced electronic amoeba when solving (a) Max-cut and (b) TSP.

The hypothesis in this chapter is that the delayed feedback seems to be troublesome issues but is useful for improving the quality of solutions because the oscillation forcibly causes the electronic amoeba to do trial and error. The trial-and-error behavior improves the solution quality in stead of the stochastic behavior. In this

chapter, we investigate the impact of the delayed feedback on the solution search performance of the electronic amoeba through solving the TSP by using its numerical model. The results show the delayed feedback produces instable states but contributes to improve the solution quality at the expense of time.

6.2 Formulization of solution-searching behavior of electronic amoeba with delayed feedback

I formulated the solution-searching behavior of the electronic amoeba with the delayed feedback for solving the TSP as follows:

$$X_{Vk}(t+1) = \begin{cases} X_{Vk}(t) + \dfrac{D_{in}}{L_{off}(t+1)} & (\text{if } L_{Vk}(t+1) = 0) \\ X_{Vk}(t) - \dfrac{X_{Vk}(t)}{D_{out}} & (\text{if } L_{Vk}(t+1) = 1) \end{cases}, \tag{6.1}$$

$$L_{Vk}(t+1) = \sigma_{\alpha_1,\beta_1}\left(\sum_{Ul} W_{Vk,Ul}\sigma_{\alpha_2,\beta_2}(X_{Ul}(t-\tau))\right), \tag{6.2}$$

$$L_{off}(t+1) = \sum_{iv}(1 - L_{Vk}(t+1)), \tag{6.3}$$

$$\sigma_{\alpha,\beta}(x) = \frac{1}{1+\exp(-\alpha(x-\beta))}, \tag{6.4}$$

where Δ_{in} and Δ_{out} are the amount of resource supply and withdraw, respectively, and τ is a delay time. Note that $\tau = 0$ is that there is no delay in the system. X_{Vk} represents a state variable ($0 \leq X_{Vk} \leq 1$, $V, k = 1, 2, ..., N$), where $X_{Vk} = 1$ corresponds to that a salesman visits V city at kth. $\sigma_{\alpha,\beta}$ is a sigmoid function that has a slope α and threshold value β. L_{Vk} decides to decrease the corresponding variable, which corresponds to whether to turn on the MOSFET in each unit. L_{off} is the total number of non-decreased branches that is the same as the number of

$\boldsymbol{L} = 0$ and represents the current conservation in the amoeba core corresponding to the volume conservation by dividing Δ_{in} by L_{off}. In Eq. 6.1, $X_{Vk}(t) - X_{Vk}(t)/D_{out}$ is derived from applying the Euler method to a resistor-capacitor circuit that discharge the charge of the capacitor. $W_{Vk,Ul}$ is variable interactions given by the bounce-back rule for TSP [32, 33]. The bounce-back rule for the TSP is shown in Chpt. 2. I reformulated the bounce-back rule for the TSP given by Eq. 5.4 as follows:

$$W_{Vk,Ul} = \begin{cases} 0.5 & (\text{if } V = U \text{ at } k \neq l \text{ or } V \neq U \text{ at } k = l) \\ \nu \cdot \text{dist}(V, U) & (\text{if } V \neq U \text{ and } (|k - l| = 1 \text{ or } |k - l| = N - 1)) \\ 0 & (\text{otherwise}) \end{cases}, \quad (6.5)$$

where ν is a normalization coefficient to stabilize the behavior of the system when finding a solution and N is the number of cities. The description of the normalization coefficient is shown in Appx. B. Eq. 6.5 decreases the total route length by only one path, compared to original one (shown in Appx. C). The general description of Eq. 5.4 is shown in Chpt. 5.

We calculate Eqs. 6.1 and 6.2 iteratively successively using Eqs. 6.3–6.5. In the simulation, the parameters Δ_{in}, Δ_{out}, α_1, β_1, α_2, and β_2 were set to 0.1, 700, -1000, 0.5, -35, and 0.5, respectively. ν was set to $\nu = 0.5/\max(\text{dist}(V, V') + \text{dist}(V', V''))\,(V \neq V' \neq V'')$. An initial value of X_{Vk} at $t = 0$ were randomly assigned in range $[-2.5 \times 10^{-3}, 2.5 \times 10^{-3}]$ with the offset 0.99 for each trial. L_{Vk} was set to 1 before $t = 0$ because the values of X_{Vk} should be decreased when X_{Vk} was set to nearly 0.99. An average intercity distance of used 10-city instance was 100, and the standard deviation of intercity distances was 17 (the used instance is the same to the instance in Chpt. 3). I regarded X_{Vk} as 1 if it was less than 0.2; on the other hand, we regarded it as 0 if it was greater than 0.8. Then, I judged whether the system finds a solution, by checking constraints of the TSP in every iteration. In this experiment, ν of the used instance was set to 19.04×10^{-4}.

6.3 Results

6.3.1 Influence of delay on performance

Figure 6.3 shows time evolutions of all variables when changing the delay time, τ. Although all variables (100 variables) are displayed in the figure, they are almost overlapped. The variables did not oscillate when $\tau = 0$, and they immediately became stable and found a solution as shown in Fig. 6.3(a). In contrast, the variables oscillated when $\tau = 100$ and $\tau = 200$ (shown in Figs. 6.3(b) and (c), respectively), and after the time evolution, they reached the stationary state. It was found that initially all the variables oscillated similarly, then gradually phased out from each other and bifurcated. The variables involved in the same city bifurcated almost at the same time and followed the same curve.

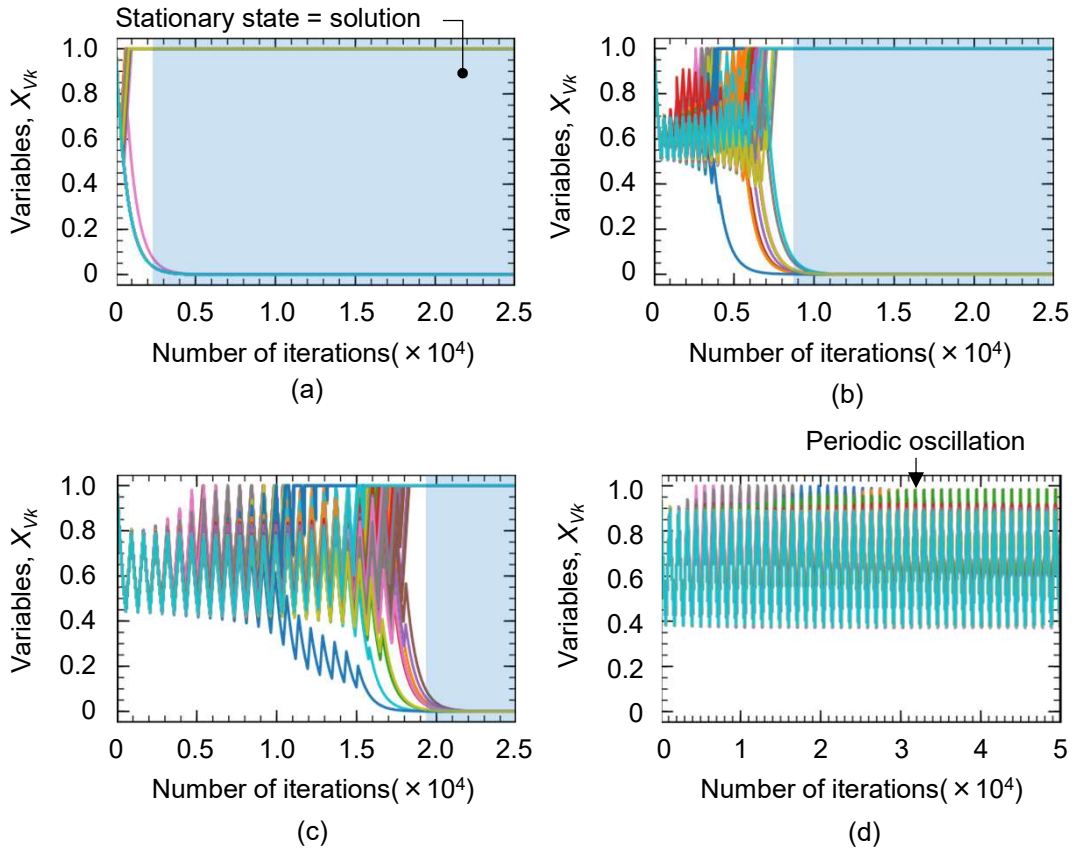

Figure 6.3: Time evolution of X_{Vk} when (a) $\tau = 0$, (b) $\tau = 100$, (c) $\tau = 200$, and (d) $\tau = 300$.

Figure 6.4 shows the number of oscillations and average oscillating period when τ is increased. The average oscillation period and number of oscillations linearly grew with the delay τ. These results indicate that a time to find a solution depends on the magnitude of the delay.

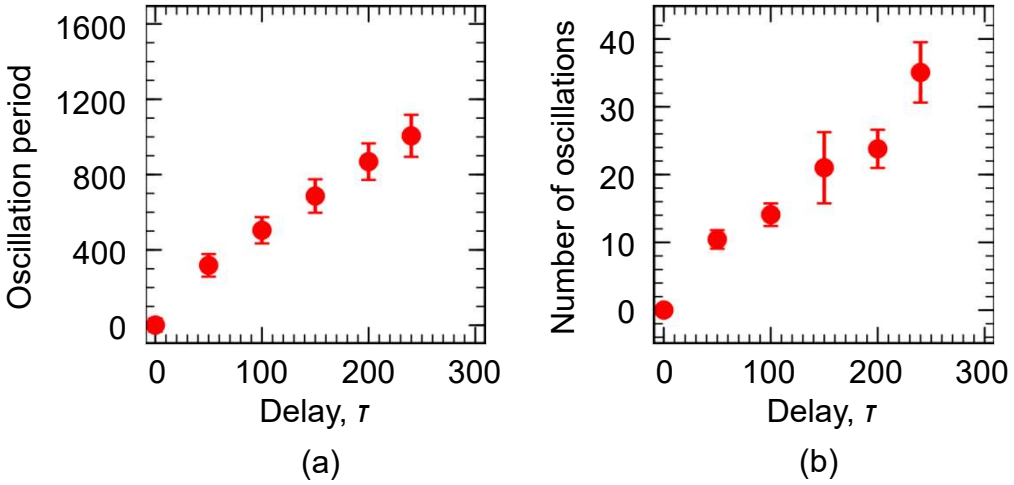

Figure 6.4: (a) Oscillation period and (b) number of oscillations, where they are derived from average of all variables when solving 10-city TSP.

Figure 6.5 shows the dependence of the solution quality and iterations to find a solution on the length of a delay time. We observed that the quality of solutions was improved and the iterations were increased by increasing τ as shown in Figs. 6.5(a) and 6.5(b), respectively. Usually, the quality of the solution obtained by the metaheuristic algorithm for optimization problems is proportion to the solution search time [140–142, 149]. Ising machines using simulated annealing or quantum annealing also take a long time to improve the quality of solutions.

We verified the hypothesis that the delayed feedback made the amoeba-inspired computing system do trial-and-error behavior and the behavior improved the solution quality. Although I hoped that the solution quality was more improved by increasing a delay time, the result was contrary to the expectation: the variables continued oscillating periodically when $\tau = 300$, such as the limit cycle in a nonlinear dynamical system, as shown in Fig. 6.3(d). The variables never became stable

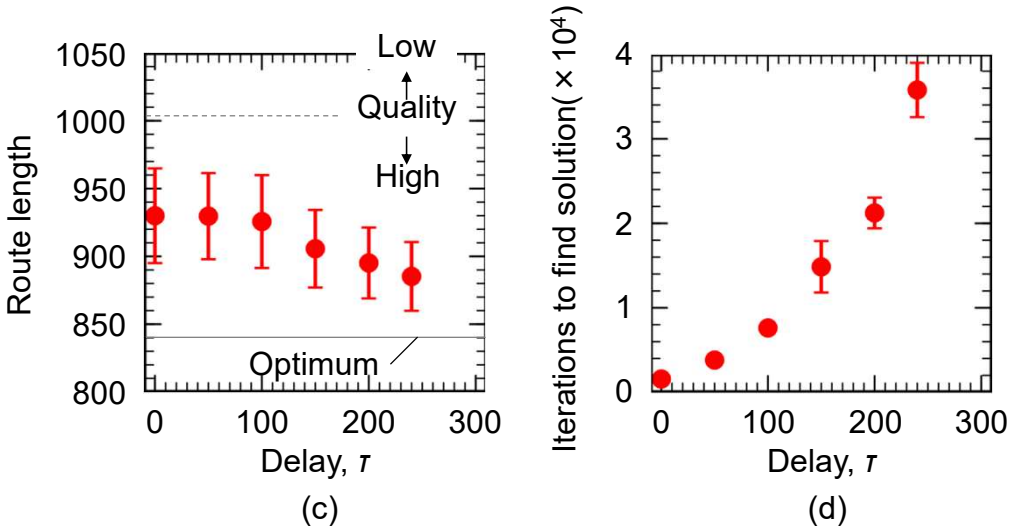

Figure 6.5: Evaluation of solution-search performance as function of τ when solving 10-city instance. (a) Solution quality and (b) iterations to find solution. Broken line is average quality of solutions obtained by random sampling from 10000 trials. Solid line is optimal solution of used instance. Error bars are standard deviation derived from 100 trials for each magnitude of τ.

and could not reached a solution.

6.3.2 Delay time scheduling

The next hypothesis is that if the number of oscillations due to τ depends on the solution quality as shown in Fig. 6.4, more improvement of the solution quality can probably be achieved by introducing a process of becoming the instable states to the stable states. The scheduling of a delay time improves the solution quality, such as the temperature control in the simulated annealing. I defined the delay time scheduling as follows: the magnitude of the delay is initially set to be large such that the variables do not converge at a solution, and then, it is gradually decreased. I defined three parameters in relation to the scheduling: one is the maximum value of the delay, second is the width to continue the same magnitude of the delay (simply called width), and third is the height of decreasing delay at the end of the width (simply called height) as shown in Fig. 6.6(a).

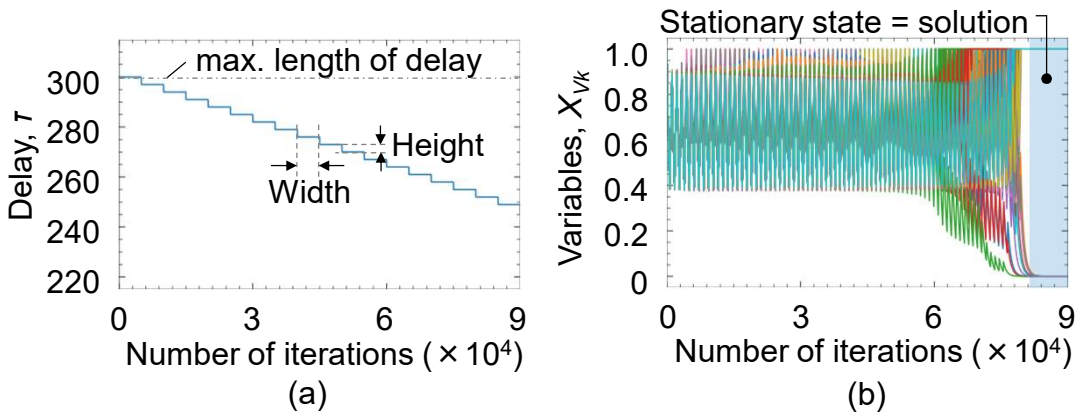

Figure 6.6: (a) Delay scheduling and (b) output waveforms of $\boldsymbol{X}$.

Figure 6.6(b) shows time evolutions of $\boldsymbol{X}$ when applying the delay scheduling, where the maximum delay, width, and height were set to 300, 5000, and 3. The variables firstly oscillated as with Fig. 6.3, and gradually decreasing τ, they became stable and the system found a solution.

Figure 6.7 shows the computer simulation results when changing the width and height at fixed maximum value of the delay. The scheduling parameters in Figs. 6.7(a) and 6.7(b) were the maximum length of the delay of 300 and the height of 3 at fixed width, and those in Figs. 6.7(c) and 6.7(d) were the maximum length of the delay of 300 and the height of 15000 at fixed height. The quality of solutions was improved by increasing the scheduling parameters: longer width and shorter height improve the solution quality. The iterations to find a solution became longer by increasing the scheduling parameters. The obtained results are similar to the temperature scheduling in the SA [69].

6.3.3 Optimizing normalization coefficient

There is a case to improve the quality of solutions by optimizing the parameters such as Δ_{in}, Δ_{out}, α_1, β_1, α_2, and β_2. I modified the normalization coefficient, ν, rather than other parameters because the solution quality strongly depends on it. When increasing ν, the optimality increases but the legality decreases, such as the Ising

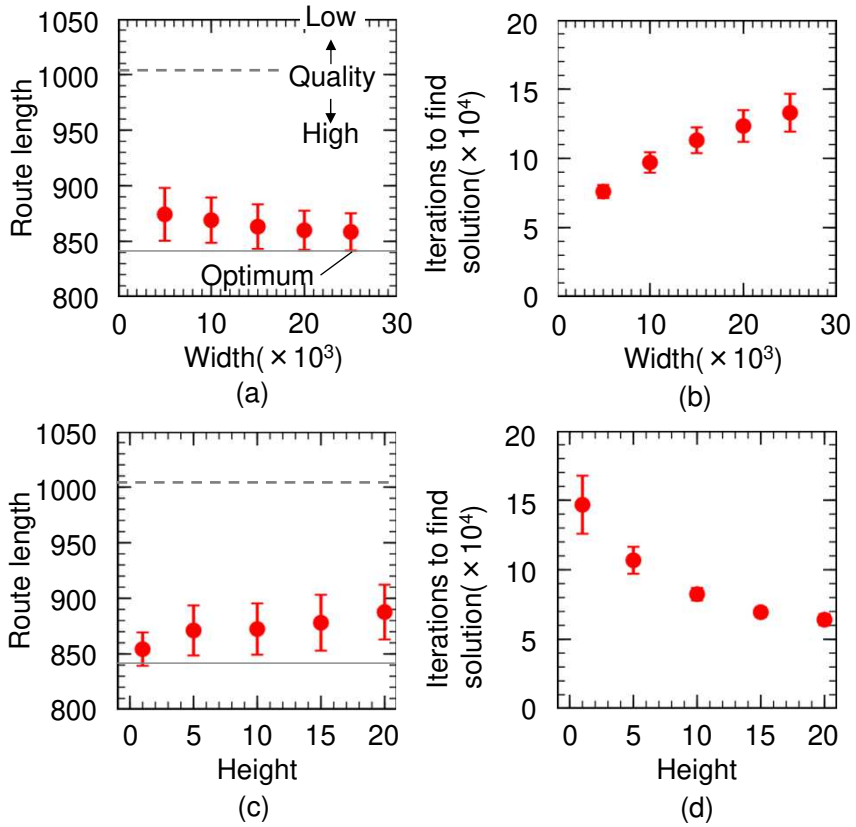

Figure 6.7: Solution-search performance when changing scheduling parameters. (a) Solution quality and (b) iterations to find solution when changing width. (c) Solution quality and (d) iterations to find solution when changing height. ν was set to $19.04 \times 10^{-4} - 5 \times 10^{-6}$. I tested 100 times.

machine: the variables sometimes converges at an illegal candidate when increasing ν.

We set the normalized coefficient as $\nu = 19.04 \times 10^{-4} + \nu_{offset}$. Figure 6.8 shows the computer simulation results when changing ν_{offset}. The quality of solutions was improved by increasing ν_{offset}, although the iterations to find a solution were constant as shown in Figs. 6.8(a) and 6.8(b). On the other hand, as shown in Fig. 6.8(c), the success rate to find a legal solution was degraded with the increase in ν_{offset}. The variables did not converge at a legal solution even once at $\nu_{offset} = 5 \times 10^{-5}$. The rate might have been improved by extending the scheduling time. These results indicate that there is the trade-off between the optimality and the legality when increasing ν_{offset}, such as a penalty term of the Ising machine [29, 64].

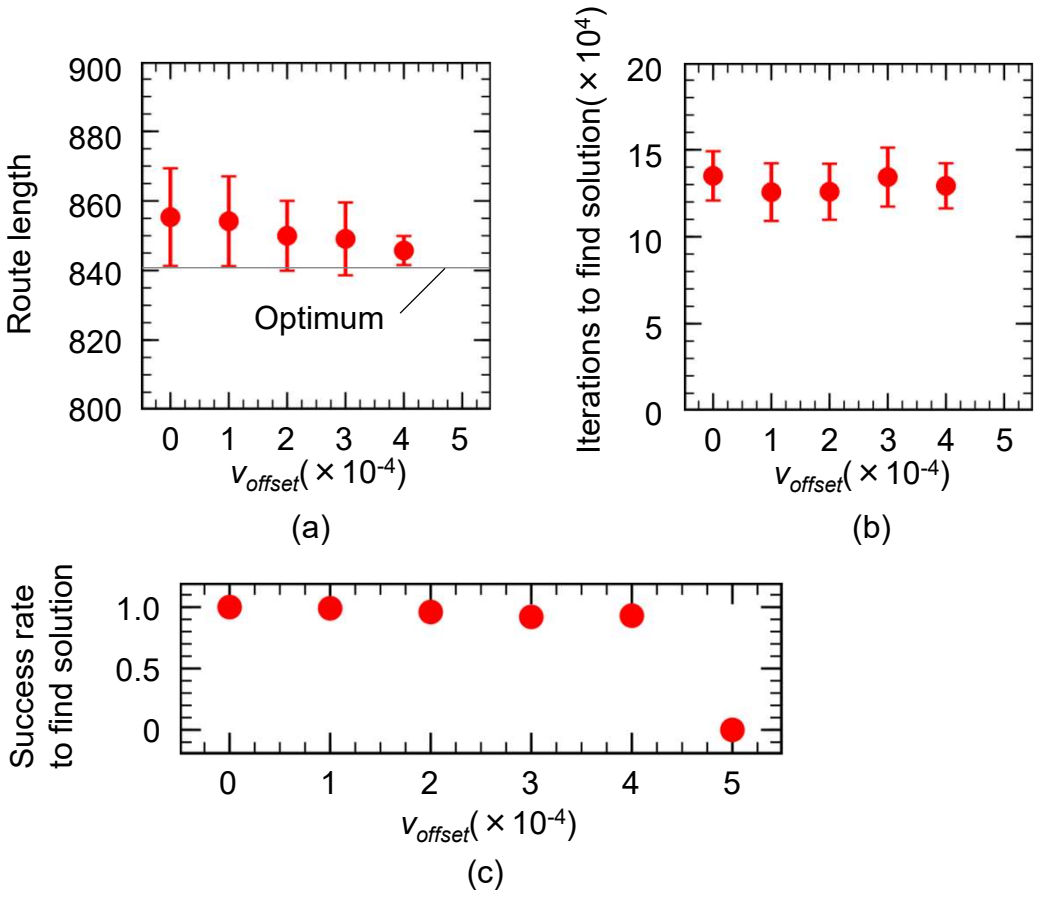

Figure 6.8: Dependence of normalization coefficient on solution-searching performance. (a) Solution quality, (b) iterations to find solution, and (c) success rate of finding legal solutions when changing ν_{offset}. Maximum delay, width, and height were set to 300, 15000, and 3, respectively. Average solution quality was derived from success trials.

6.4 Reason for improving quality of solutions by delayed feedback

We discuss the reason why the system with the delayed feedback improves the quality of solutions when increasing a delay. To come to the point, nonlinear time evolutions of units in the amoeba core as shown in Fig. 6.9 improve the quality. As (1) in Fig. 6.9, the variable decreases exponentially because I formulized the capacitor-

discharging. The differences between two variables are decreased when $L_1 = 0$ and $L_2 = 0$. As (2) in Fig. 6.9, when $L_1 = 0$ and $L_2 = 1$, the differences increase because of the current conservation (the volume conservation). As (3) in Fig. 6.9, the differences are constant.

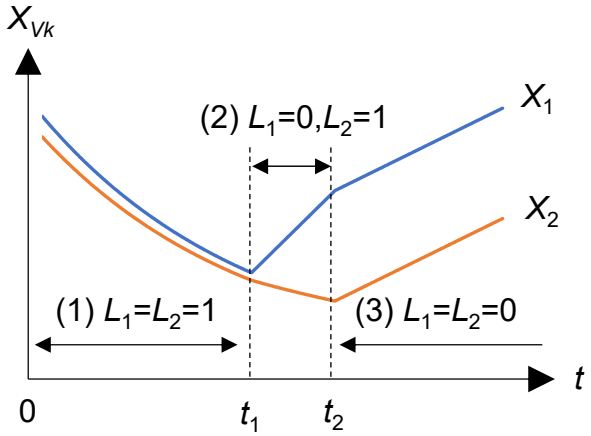

Figure 6.9: Nonlinear time evolutions of units in amoeba core.

Figure 6.10 shows a simple problem example to understand the reason for improving the quality of solutions when increasing a delay, where the state variables are showed by the inverse logic. The variables covered by blue square in Fig. 6.10 are fixed and covered by red square are free to search a solution. In the right side, the solutions are showed, where the solution 2 is better than the solution 1.

Using Eqs. 6.1–6.5, I searched a solution to the simple problem. Figure 6.11 shows time evolutions of $X_{A,1}$ and $X_{A,2}$ when changing τ. In this computer simulation, I set the initial values of $X_{A,1}$ and $X_{A,2}$ such that the variables reached the worse solution when $\tau = 0$. We confirmed that the variables converged at the better solution when increasing a delay, such as the experimental results. If the initial values are the same, the variables reach the better solution owing to the bounce-back rule. In this case, the gap decreases owing to the exponential delay when $L_{A,1} = 1$ and $L_{A,2} = 1$, and then, the variables tend to reach the better solution because of the property of the bounce-back rule and the volume conservation, where the nonlinearity of the volume conservation helps the bifurcation of the variables. When a

time delay increased, the oscillation amplitude became large and the gap decreases. This is the reason why the delayed feedback improves the quality of solutions by increasing the delay.

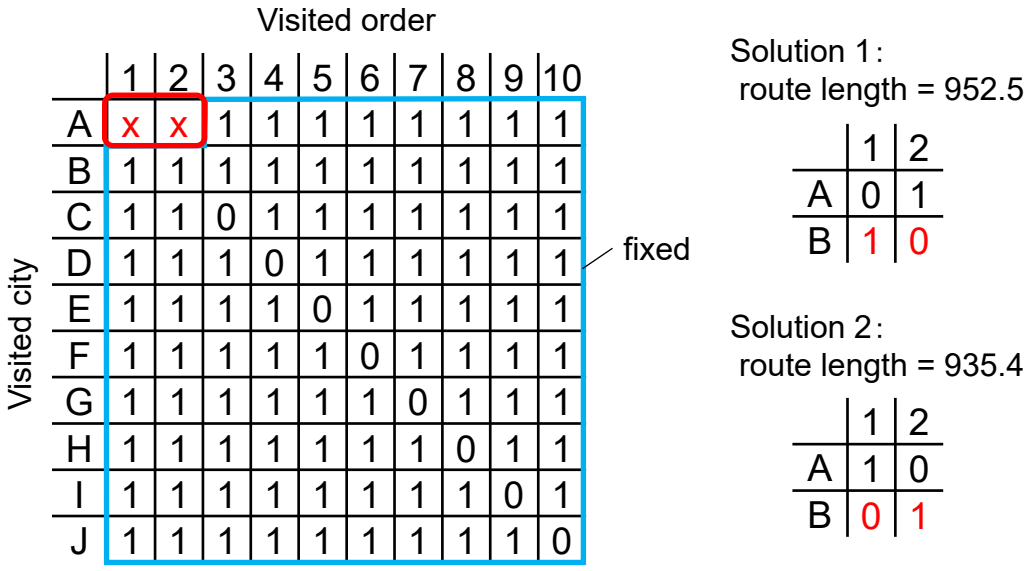

Figure 6.10: Simple problem example for understanding results of delayed feedback system.

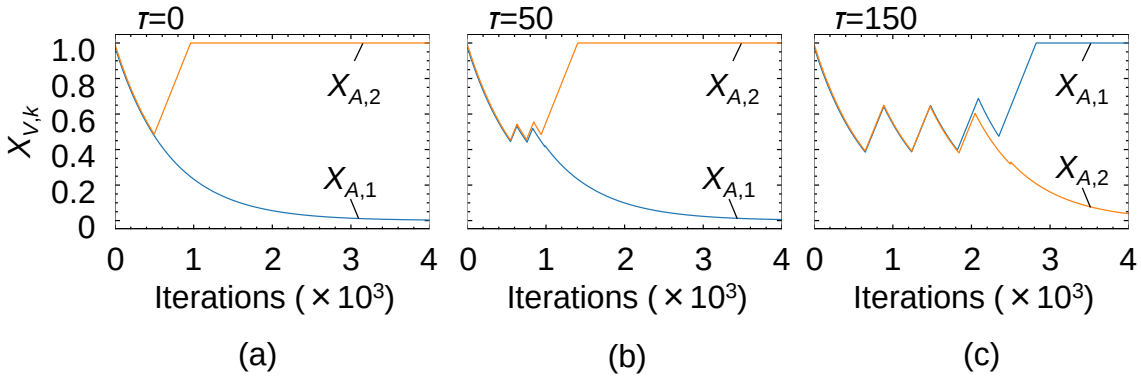

Figure 6.11: Time evolutions of $X_{A,1}$ and $X_{A,2}$ when (a) $\tau = 0$, (b) $\tau = 50$, and (c) $\tau = 150$.

From the above discussion, the delay scheduling is considered to be a reasonable method to improve the quality. When the magnitude of a delay is large, it attracts the variables into a periodic orbit. By decreasing a delay, the amplitude reduces and it is unlikely to attract the variables. Then, the variable that is most likely to

converge at 0 or 1 is determined. As a result, plausible variables, decreasing a total route, gradually converge at a stable state when decreasing a delay continuously.

Linear time evolutions of variables in the amoeba core may not improve the quality, considering the above discussion. To confirm whether not to improve the quality by the linear time evolution, I reformulated 6.1 as follows:

$$X_{Vk}(t+1) = \begin{cases} X_{Vk}(t) + \Delta'_{in} & (\text{if } L_{Vk}(t+1) = 0) \\ X_{Vk}(t) - \Delta'_{out} & (\text{if } L_{Vk}(t+1) = 1) \end{cases}. \tag{6.6}$$

Figure 6.12 shows time evolutions of $X_{A,1}$ and $X_{A,2}$ when changing a time delay. When increasing τ up to 500 (Figs. 6.12(a)–(f)), the number of oscillations and the amplitude of variables increased. However, the convergence to the better solution was not confirmed. As shown in Fig. 6.12(g), when $\tau = 600$, the variables converged to the better solution. This is because there happened the nonlinearity, since the variables approached the upper limit (1.0) and lower limit (0.0): therefore, the variable states turned over and they reached the better solution.

Figure 6.13 shows simulation results when solving the 10-city TSP instance,

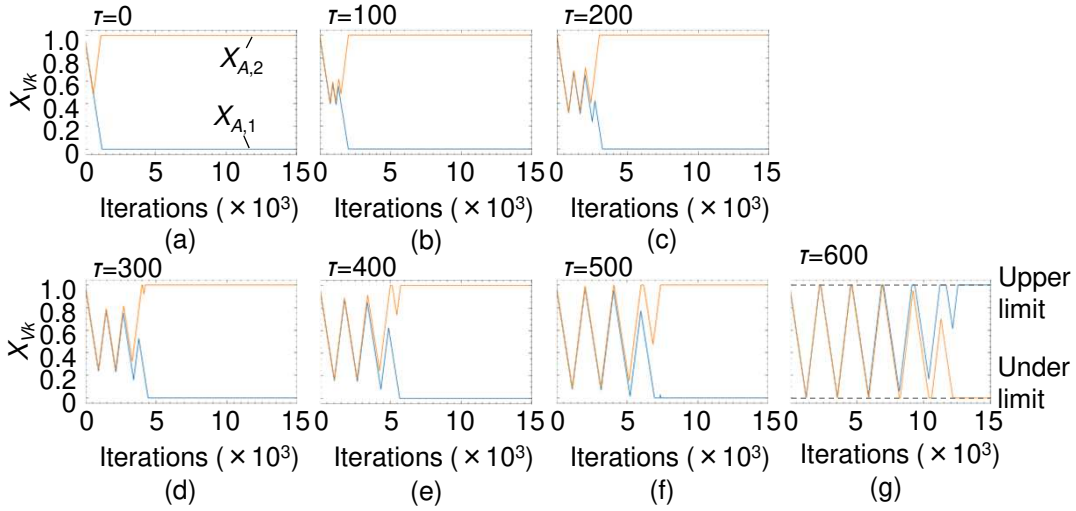

Figure 6.12: Time evolutions of $X_{A,1}$ and $X_{A,2}$ when units evolve linearly. Results of (a) $\tau = 0$, (b) $\tau = 100$, (c) $\tau = 200$, (d) $\tau = 300$, (e) $\tau = 400$, (f) $\tau = 500$, and (g) $\tau = 600$. Δ'_{in} and Δ'_{out} were 0.001 and 0.0008, respectively.

while changing τ. The parameters were the same to Fig. 6.12. From the results, the quality of solutions was not improved by increasing τ except for $\tau = 410$. When $\tau = 410$, the variables approached the upper and lower limits, such as Fig. 6.12(g), as shown in Fig. 6.14. These results indicate that the nonlinearity in the amoeba core help for the system to improve the quality of solutions.

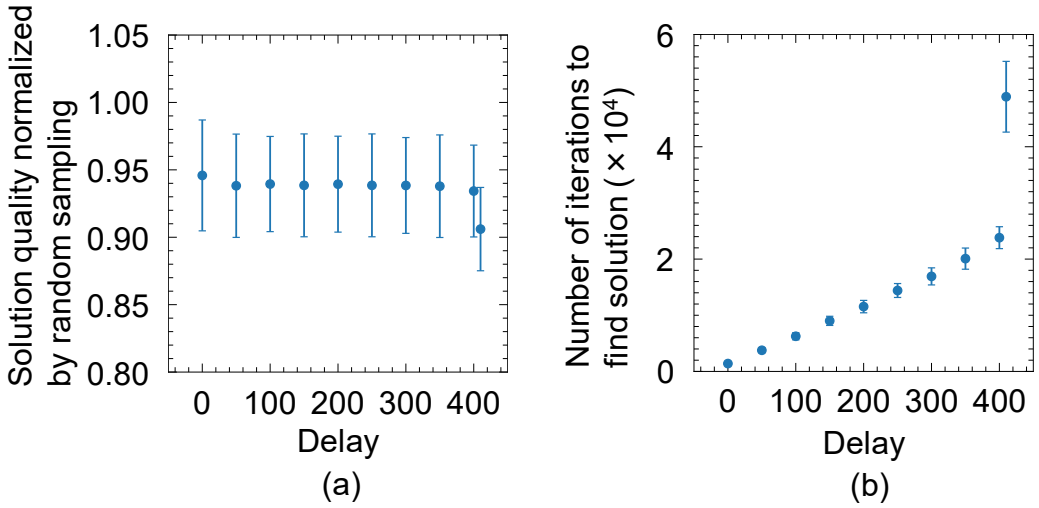

Figure 6.13: (a) Solution quality and (b) number of iterations to find solution when solving 10-city TSP instance.

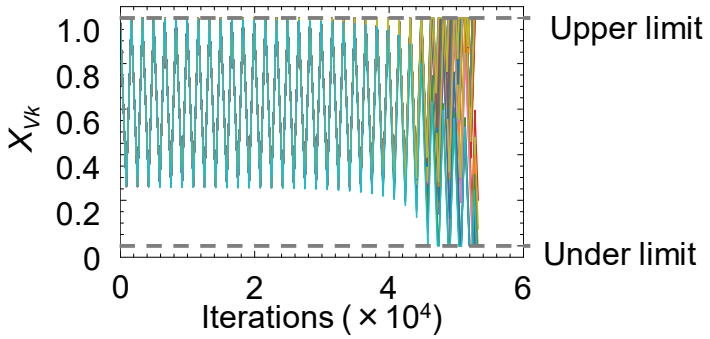

Figure 6.14: Time evolutions of all variables.

6.5 Discussion

In this computer simulation, we regarded the occurrence of a delay in the feedback signal, L_{Vk}. Because whether a delay occurs in the amoeba core or the IMC is a

relative problem, both are possible to apply the delay scheduling. In the implementation of a fabricated system, we consider that a delay in the amoeba core is easy to implement: that is, the transition speed of variables, first, is made to be fast, and the speed gradually decreases as time advances. Such a transition scheduling may be implemented by gradually decreasing the magnitude of the current from the current source in the amoeba core.

Although I proposed methods to improve the solution quality of the electronic amoeba such as GA-based algorithm and stochastic fluctuation shown in Appx. C, they require additional equipment, which decrease the scalability and increase the power consumption. On the other hand, we can easily improve the solution quality utilizing the delay in analog circuits without additional communication and circuits. Therefore, the delay-induced method is superior to the GA-based algorithm and introducing the stochastic behavior.

A delay due to time constants is usually eliminated from digital integrated circuits by miniaturing the size of transistors as much as possible in order to increase the clock frequency, decrease the power consumption, and prevent incorrect actions. Adjusting the clock frequency also reduces the impact of a delay on circuit operations. On the other hand, a delay in analog circuits cannot be avoided because all operations are processed in a continuous time without a waiting time; therefore, we have no choice but should cope with a delay well. In the electronic amoeba, we can utilize a time delay to improve the quality of solutions.

The delay has been widely studied because it produces mathematically fascinating phenomena, such as chaos in the Mackey-Glass equation, which explains production of white blood cell [150, 151]. In the real world, one considers the delay as putting us into troublesome issues because there occurs oscillation in a feedback system due to signal propagation delay, and we sometimes fail to suppress the oscillation. A delay in a feedback system cannot be neglected as long as we treat not ideal but practical problems since it potentially exists all over the place. However, such

as Mackey-Glass equation, several literatures reported that the delay feedback can be useful in several fields: the delay can make an unstable system stable [152, 153], reservoir computing system proposed as one of the bio-inspired computing systems predicts time-series data using the delay in a non-linear dynamical system [154–156], the delayed feedback in the auditory system of a human causes negative influences on utterance but can improve stuttering [157, 158], and multistability in a neuron that produces diverse behavior with it [159]. The obtained results indicate that a delay is also useful, as with the literatures, for the system to improve the solution-search performance without any additional circuit.

6.6 Conclusion

In this chapter, I made a recurrence formula to reproduce electronic amoeba having the delayed feedback and demonstrated a hypothesis that the delayed feedback in analog circuits in the system produced trial-and-error behavior and improved the quality of the solution to the traveling salesman problem by numerical simulations, compared with the results that did not impose the delay. The quality of solutions was improved and the iterations to find a solution was increased by increasing the magnitude of the delay. Although the variables did not become stable and did not find a solution when a time delay is too large, introducing the delay scheduling converged the variables at a solution and the quality of obtained solutions was more improved than those of only imposing the delay. From the results of numerical simulations, the delayed feedback was useful for the solution search in the electronic amoeba in terms of improving the solution quality, and additional circuits to introduce external fluctuation to the circuits is not necessary but only adjusts the magnitude of the delay.

Chapter 7

Conclusion

In this thesis, I summarized electronic implementation of an amoeba-based computing system that solves the satisfiability problem (SAT), maximum cut problem (Max-cut), and traveling salesman problem (TSP), which are intractable problems for a conventional computer.

Chapter 1 introduced the background and purpose of this work.

Chapter 2 described an amoeba-based and -inspired electronic computing system (called electronic amoeba) and explained the SAT, Max-cut, and TSP.

Chapter 3 summarized physical computing system dedicated to solving optimization problems.

In Chap. 4, I showed the analog-digital hybrid electronic amoeba for solving the SAT. The solution-searching time depended on the error property. It was minimized for each error probability by an appropriate error period. Solution search of the electronic amoeba is robust to the error property. There also existed an optimal error probability and period that minimize the solution-searching time. By using a model amoeba, I also showed that asymmetric dynamics in the electronic amoeba due to transconductance of the metal-oxide-semiconductor-field-effect-transistor played an important role in solving SAT. Thanks to introducing the asymmetric dynamics to the model, it obtained the robustness to the error property, such as the electronic

amoeba, compared to original one. Moreover, the model with the asymmetry improved the solution-searching performance in terms of the search time. Such an asymmetry is in the amoeboid organism and also in the Caenorhabditis elegans, and the asymmetric dynamics are considered to be reasonable way to survive based on the Darwinism.

In Chap. 5, I proposed the electronic amoeba integrating a crossbar instance-mapping circuit (IMC) that performs the product-sum operation and thresholding based on the bounce-back rule. The resistors at each cross-point in the crossbar IMC represent variable interactions of the bounce-back rule. I formulized the bounce-back rule for Max-cut and demonstrated that the physically implemented electronic amoeba found a solution to the Max-cut. I also evaluated the performance of the electronic amoeba to Max-cut by using a circuit simulator, and we found that when the electronic amoeba searches a solution to 5–100 vertexes Max-cut, the quality of solutions did not degrade to that of solutions obtained by a random sampling which produces a solution randomly and the solution-searching time of the electronic amoeba was constant. Moreover, we confirmed that the electronic amoeba with the crossbar IMC found a solution to the TSP. The performance was evaluated by the circuit simulator and we found that the quality of solutions did not degrade and the solution search time grew in only linear time when increasing the number of cities. The electronic amoeba is suitable for strictly constrained problems such as the TSP compared to Ising machines because it certainly satisfies constraints owing to the bounce-back rule. The compactness of the crossbar IMC enables the electronic amoeba to implement edge-side computing system with state-of-the-art large-scale-integration technique and provide a better solution at low-power consumption.

In Chap. 6, I demonstrated that delayed feedback in the amoeba-inspired computing system improved the quality of solutions without additional circuits. One of the challenges of the electronic amoeba is to improve the quality. I proposed genetic-algorithm-based algorithm and introducing fluctuation but they require overheads,

which limit the performance of the electronic amoeba. The delayed feedback utilizes the delay in an analog circuit, although improving the quality usually requires external circuits to produce fluctuation or external communication via a conventional computer. The quality depended on the magnitude of the delay. I introduced scheduling of a delay and found that the quality of solutions was further improved. If an appropriate value of a normalization coefficient in the bounce-back rule for the TSP was set, the quality of solutions was also improved, otherwise the system fell into an illegal candidate.

Appendix A

Solution-searching performance of Ising-model-based algorithm

In this chapter, we discuss weak points in the Ising machine. To discuss it, I evaluated the solution-searching performance of Ising-model-based algorithms. The Ising model for the TSP is formulated on the basis of the quadratic unconstrained binary optimization (QUBO) given as follows [29, 67]:

$$E = A\sum_{v}\left(1-\sum_{j}x_{v,j}\right)^{2} + A\sum_{j}\left(1-\sum_{v}x_{v,j}\right)^{2} + B\sum_{(u,v)\in E}W_{u,v}\sum_{j}x_{u,j}x_{v,j+1}. \tag{A.1}$$

The encoding of the variable $x_{v,j}$ ($x_{v,j} \in \{0,1\}$) for the QUBO of the TSP is the same to the Hopfield Neural Network for solving the TSP [60]: $x_{v,j} = 1$ denotes that a salesman visits v-city at jth order. $W_{u,v}$ is a distance between city u and v. A is a coefficient of penalty terms and B is that of an objective function term. We can reformulate Eq. A.1 because A and B are parameters that are not meaningful numerically as follows:

$$E = \alpha\sum_{v}\left(1-\sum_{j}x_{v,j}\right)^{2} + \alpha\sum_{j}\left(1-\sum_{v}x_{v,j}\right)^{2} + \sum_{u,v}W_{u,v}\sum_{j}x_{u,j}x_{v,j+1}, \tag{A.2}$$

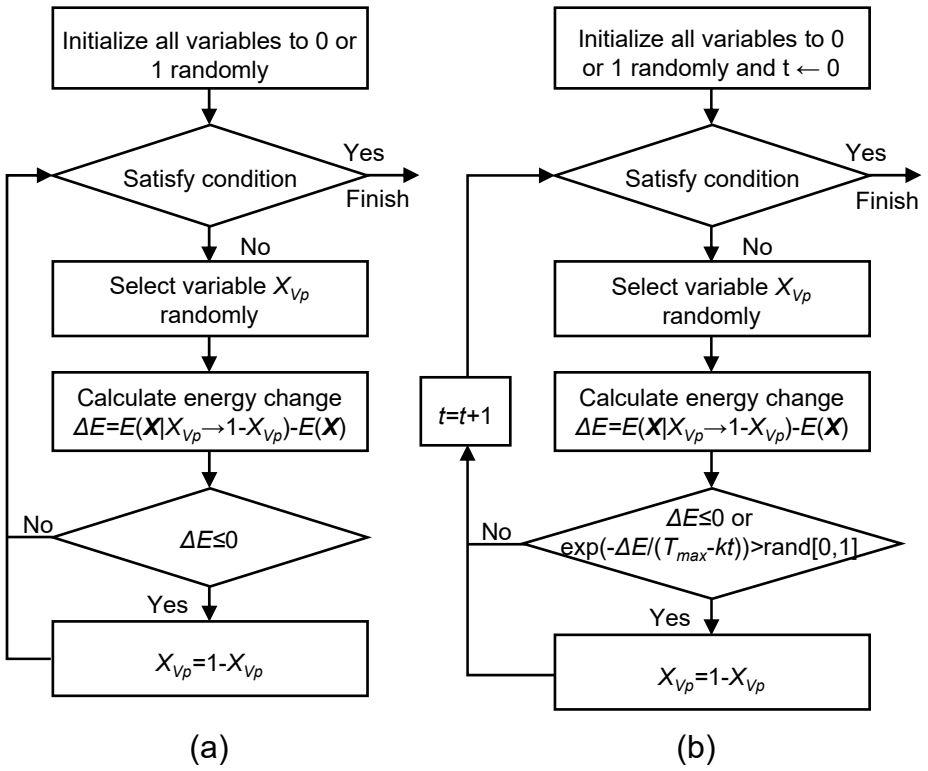

Figure A.1: Ising-model-based algorithm (a) without and (b) with SA.

where $\alpha = A/B$. That is, we have to only tune α. In Eq. A.2, the first and second terms are the constraints of the TSP that forbid visiting different cities at the same time and visiting the same city, and these terms are always positive because of a square number. If a salesman visits all cities at only once, the penalty terms become 0, otherwise E increases. The third term represents the objective function of the TSP that describes a total route. When a salesman finds an optimal route, Eq. A.2 becomes the lowest energy under proper α. Eq. A.2 can be transformed into the Ising model by $s_{i,j} = 2x_{i,j} - 1$, where $s_{i,j}$ represents the spin at lattice site (i, j).

To minimize Eq. A.2, I examined two Ising-model-based algorithms without and with simulated annealing (SA), corresponding to Ising machines which are implemented by a digital circuit, such as the CMOS annealing and Digital Annealer. Figure A.1 shows the algorithm without and with SA (hereinafter called Ising model and Ising model SA, respectively). We first initialize all variables by 0 or 1. Next, we select a variable X_{Vp} randomly and then calculate the energy differ-

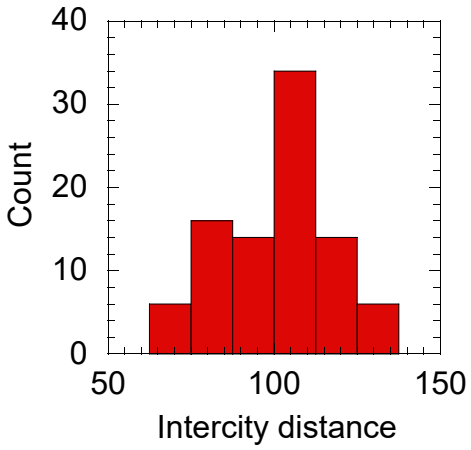

Figure A.2: 10-city TSP instance. Average intercity distance is 100

ence $\Delta E = E(\boldsymbol{X}|X_{Vp} \rightarrow 1 - X_{Vp}) - E(\boldsymbol{X})$). If the energy difference is smaller than or equal to 0, we absolutely flip the selected variable, else if $\Delta E > 0$, the selected variable does not flip. At this time, the Ising model SA probabilistically flips the selected variable under the condition, $\exp(-\Delta E/(T_{max} - kt)) > \text{rand}[0, 1]$, where T_{max} is an initial temperature, k is a temperature gradient, t is the number of iterations, and $\text{rand}[0, 1]$ is the random number between 0 and 1. That is, the proposed flip is accepted even when it increases the energy. The method to reduce the temperature is called cooling schedule [160], and in this experiment, I adopted a linear scheduling. If the temperature decreases sufficiently slow, we can find an optimal solution with the probability 1 that is proven theoretically [161].

Figure A.3 shows the simulation results of the Ising model. The number of cities of used instance was 10. The average intercity distance of the instance was 100 and standard deviation was 17 as shown in Fig. A.2. I tested 100 times while changing initial values. As shown in Fig. A.3(a), when increasing α, the success rate to find a legal solution increased. This is because the constraint coefficient is relatively larger than the constraint of the objective function. The reason that the success rate rapidly increased when $\alpha = 100$ was due to the average of intercity distances used in the experiment. Figure A.3(b) shows the average quality of solutions that is derived from obtained legal solutions. When increasing α, the quality of solutions

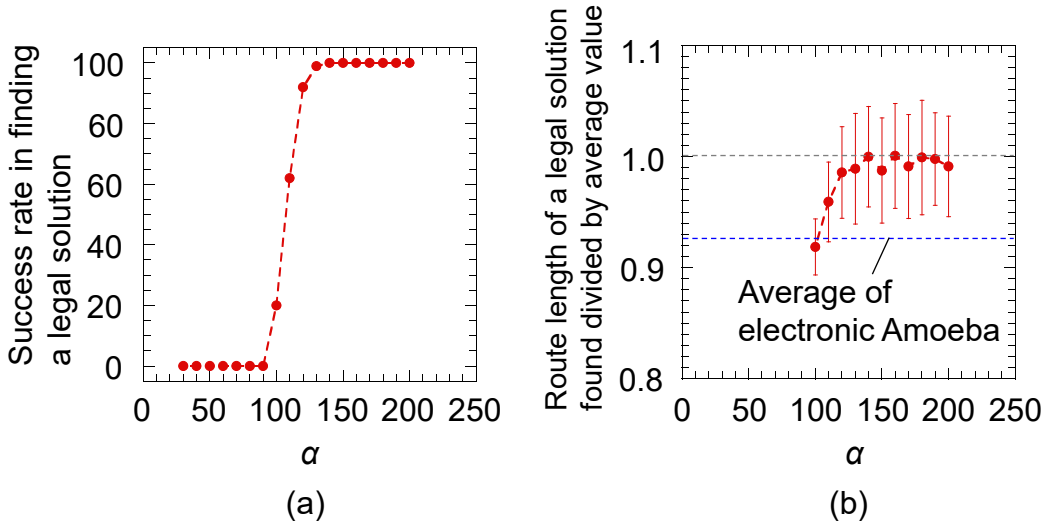

Figure A.3: (a) Success rate in finding legal solution and (b) route length of legal solution found divided by average value obtained from the random sampling as function of α when solving 10-city TSP instance.

obtained by the Ising model degraded; moreover, when $\alpha = 140$, the quality is nearly equal to that of the random sampling even though the success rate was 100%. The quality was superior to that of the electronic amoeba and at that time, the rate was about 20%, although the electronic amoeba found a solution at 100% owing to forbidding the variables that violate the constraints.

Figure A.4 shows the simulation results of the Ising model when changing the number of cities and α. In solving 10–30 cities TSP instances, the trend of the Ising model was the same as solving the 10-city TSP instance.

We derived α so that the Ising model converges to a legal solution certainly. Figure A.5 shows the example of a legal solution and the changes of the solution from the final state. In Fig. A.5(a), the energy difference between the legal state and the flipped state is given from Eq. A.2 as follows:

$$\Delta E = 2\alpha + W_{AC}. \tag{A.3}$$

Because $\alpha > 0$ and $W_{AC} > 0$, Eq. A.3 is always greater than 0; therefore, the proposed flip increases the energy and is not accepted unless we apply the SA. In

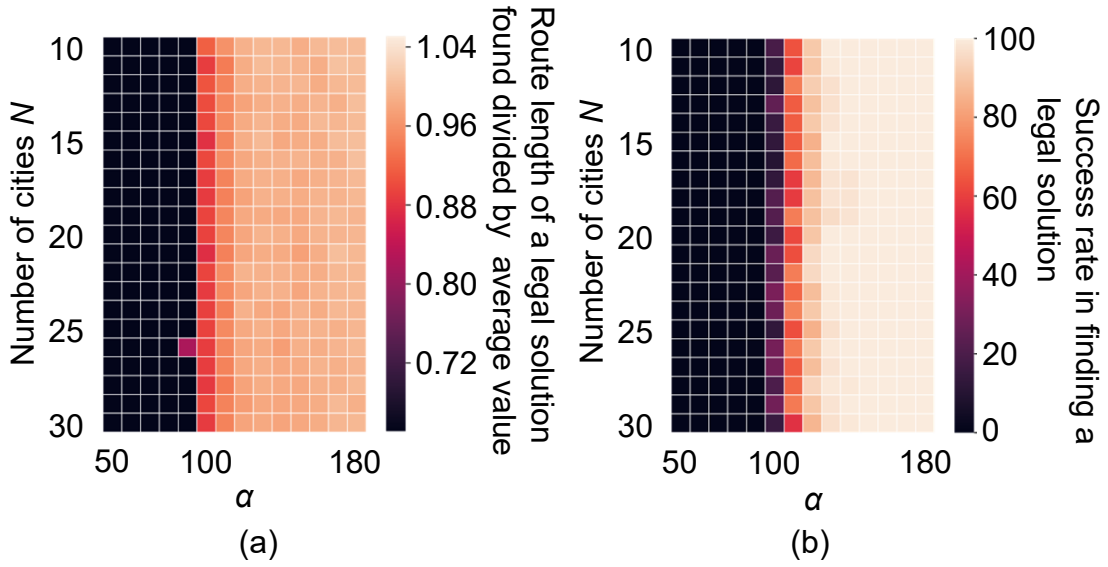

Figure A.4: (a) Quality of solutions and (b) success rate to find legal solution in Ising model.

Fig. A.5(b), the energy difference is given by

$$\Delta E = 2\alpha - W_{DA} - W_{AB}. \tag{A.4}$$

The constraint term increases, but the cost term decreases; therefore, there is the possibility that Eq. A.4 decreases the energy. In order not to decrease the energy, we impose $\Delta E > 0$ on Eq. A.4. Then, using a max() function that returns the maximum value to guarantee generality, we obtain the following equation:

$$2\max(W_{uv}) > W_{DA} + W_{AB}. \tag{A.5}$$

Therefore, the condition that does not change when the Ising model finds a legal solution is given by

$$0 < \max(W_{uv}) < \alpha. \tag{A.6}$$

A similar formulation is given in Ref. [29, 67].

I compared the performance of the Ising model with the electronic amoeba.

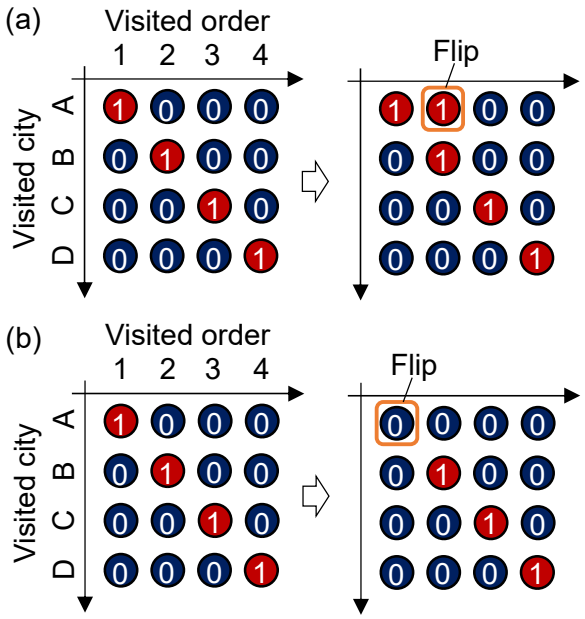

Figure A.5: Flip of (a) $X_{A,2}$ and (b) $X_{A,1}$ when solving 4-city TSP instance.

Figures A.6(a) and A.6(b) show the simulation results of the Ising model when $\alpha = 100$. We confirmed that the Ising model found legal solutions that are comparable to the solution quality of the electronic amoeba as shown in Fig. A.6(a). However, the number of times that it found a legal solution was about 20 % and an illegal candidate was found at 80 %; on the other hand, the electronic amoeba could find a solution 100 % owing to the bounce-back rule as shown in Chpt. 5. As the above discussion, the Ising model converges at a legal solution under the condition, $0 < \max(W_{uv}) < \alpha$. Figures A.6(c) and A.6(d) show the simulation results when $\alpha = \max(W_{uv}) + 1$, which is a proper value because $\max(W_{uv})$ takes a value about 150. As shown in Figs. A.6(c) and (d), although the Ising model certainly found a legal solution, the solution quality was near to that of the random sampling.

I conducted the performance evaluation of the Ising model SA shown in Fig. A.7. Figure A.7(a) indicates that in the case of $\alpha = 100$, the solution quality of the Ising model SA was improved by reducing the temperature gradient, k, and exceeded that of the electronic amoeba; however, when the number of cities increased, the number of times that the Ising model SA found a legal solution decreased as shown in

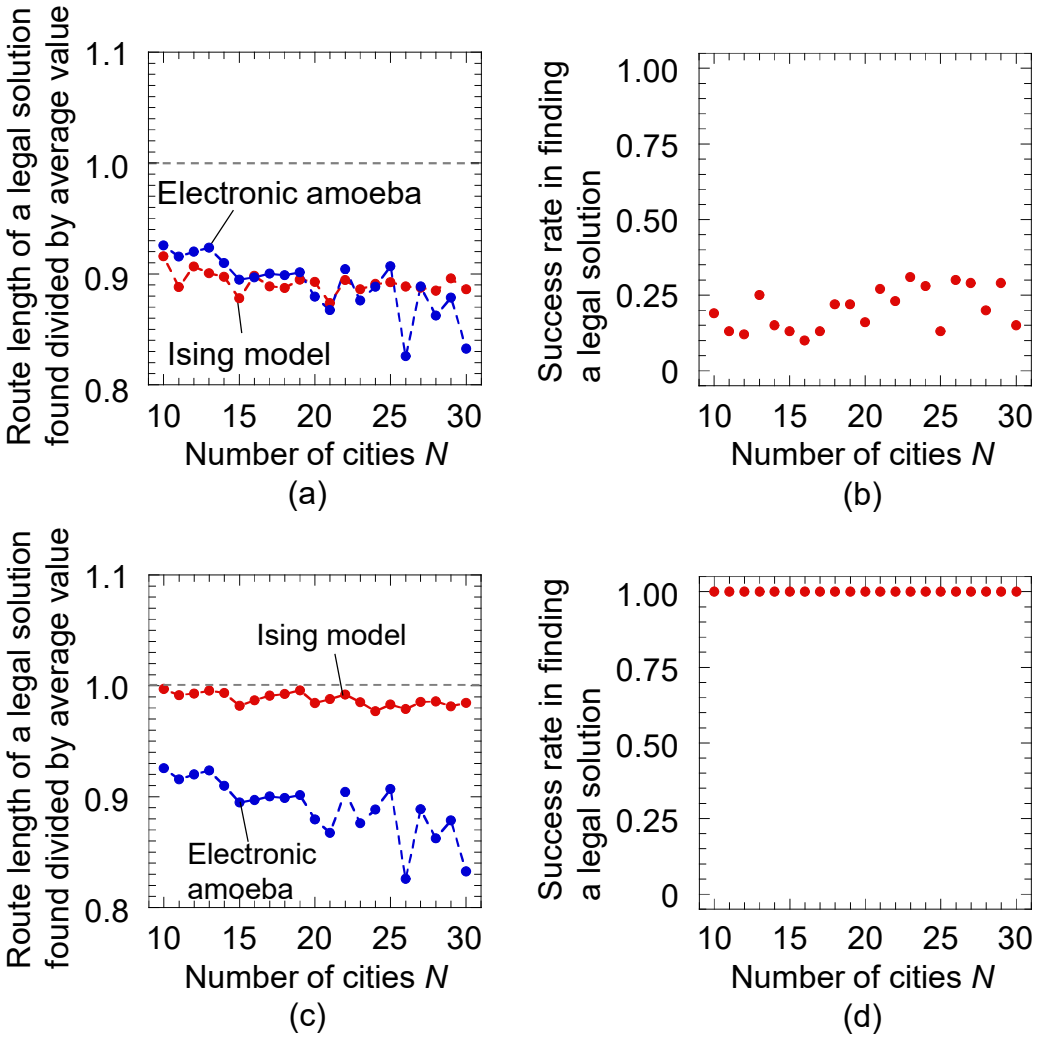

Figure A.6: Simulation results of Ising model. (a) and (b) Solution quality and number of times that Ising model finds a solution, respectively, when $\alpha = 100$. Solution quality is an average value that Ising model finds legal solution. (c) and (d) Those of when $\alpha = \max(W_{uv}) + 1$. Solution search was made until quality of obtained solution is comparable to solution quality of electronic amoeba. I tried 100 times for each instance.

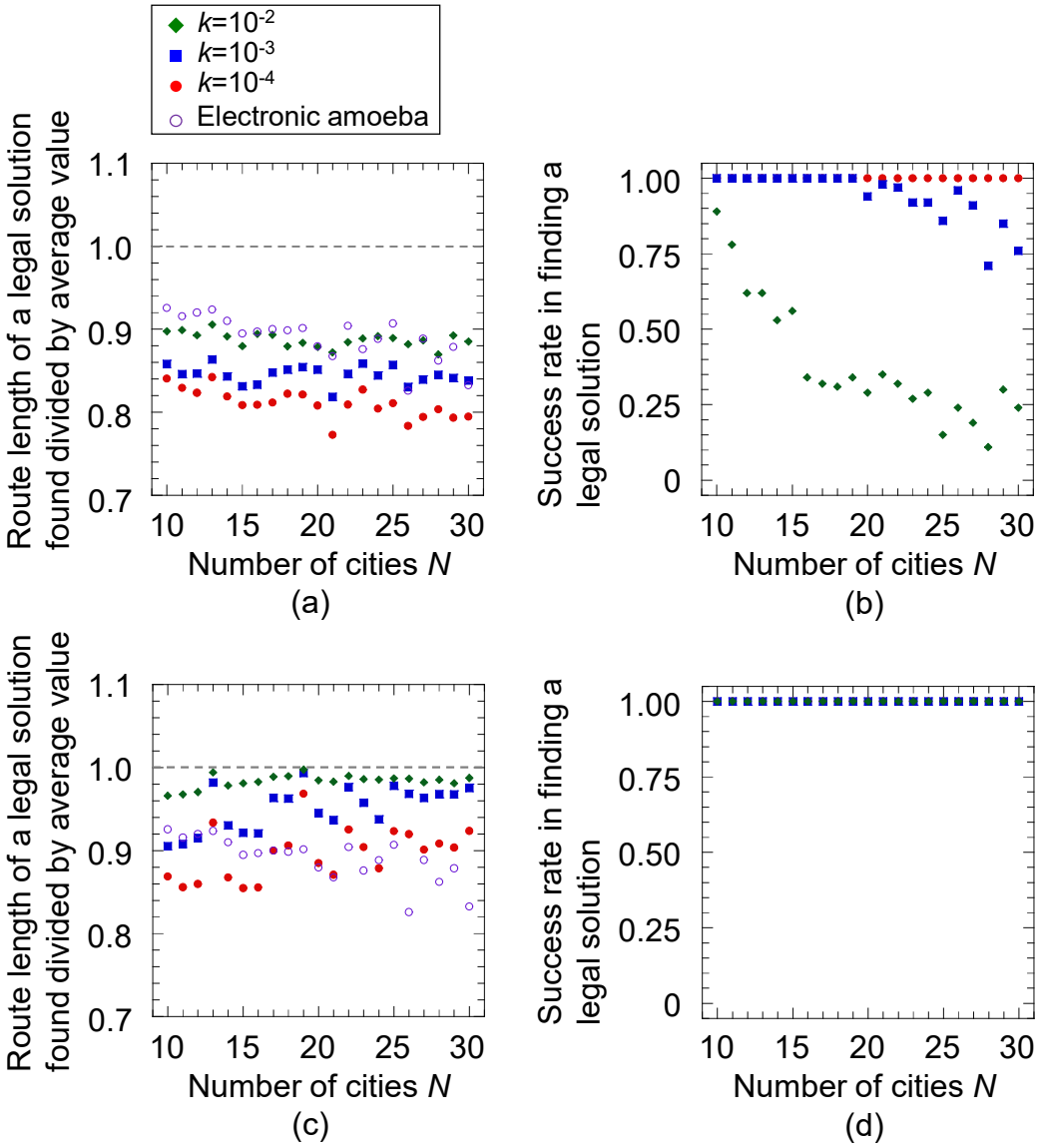

Figure A.7: Simulation results of Ising model SA. (a) and (b) Solution quality and number of times that Ising model SA finds solution, respectively. Parameters were set to $\alpha = 100$ and $T_{max} = 30$. (c) and (d) Those of when parameters are $\alpha = \max(W_{uv}) + 1$ and $T_{max} = 30$. Filled green rhombuses, blue squares, red circles, and empty purple circles are results of $k = 10^{-2}$, 10^{-3}, and 10^{-4}, and electronic amoeba, respectively. Number of iterations was set to $T_{max}/k + 10000$, and I tried 100 times for each instance.

Fig. A.7(b). We have to apply appropriate parameters for the SA, otherwise the Ising model SA cannot find a legal solution. In contrast, when using $\alpha = \max(W_{uv}) + 1$ (shown in Fig. A.7(c)), the solution quality was severely deteriorated, although the Ising model SA found a legal solution at 100 %. These results imply that the electronic amoeba is suitable for an application that requires to obtain a legal solution promptly, whereas the Ising model SA fits well to an application that needs a high-quality solution in a long time.

Appendix B

Variable encoding and stabilization condition for TSP and Max-cut

B.1 Variable encoding and stabilization condition for TSP

In this appendix, I describe the variable encoding and stabilization condition in the amoeba-inspired computing system for the TSP and Max-cut whose descriptions are shown Chap. 2. To solve the TSP, matrix representation is used such as Hopfield neural networks (HNN) [32, 33]: a row and column represent a visit city and order, respectively, as shown in Fig. B.1. Thus, we need N^2 redundant variables to solve N-city TSP. This variable encoding is the same to the Ising machine, where the variables are needed for $O(N^4)$ when the variable connectivity of the Ising machine is sparce [67].

Variable interactions for the TSP are given by Eq. 5.4. ν is given by considering stable condition when the system finds a solution, $A \to B \to C \to D \to A$. Con-

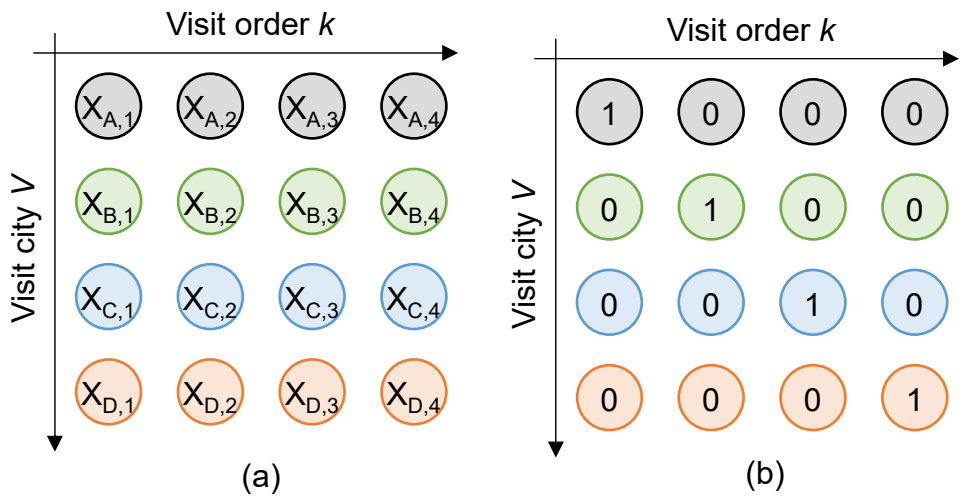

Figure B.1: (a) Variable encoding of TSP for amoeba-inspired computing system and (b) Solution example, which indicates $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$.

sidering the bounce-back rule of $X_{B,2}$, the decision whether to change the variable from 1 to 0 is determined by following equation using Eq. 5.1:

$$\begin{aligned} L_{B2}(t+\Delta t) &= \sigma_{\alpha_1,\beta_1}\left(\sum_{Ul} W_{B2,Ul}\sigma_{\alpha_2,\beta_2}\left(X_{Ul}(t)\right)\right) \\ &= \sigma_{\alpha_1,\beta_1}(W_{B2,B1}\sigma_{\alpha_2,\beta_2}(X_{B1}(t)) + W_{B2,B3}\sigma_{\alpha_2,\beta_2}(X_{B3}(t)) + ... \\ &\quad + W_{B2,C3}\sigma_{\alpha_2,\beta_2}(X_{C3}(t)) + W_{B2,D3}\sigma_{\alpha_2,\beta_2}(X_{D3}(t))). \end{aligned} \tag{B.1}$$

Substituting the values of variables shown in Fig. B.1(b) into Eq. B.1, we obtain

$$L_{B2}(t+\Delta t) = \sigma_{\alpha_1,\beta_1}(W_{B2,A1} + W_{B2,C3}), \tag{B.2}$$

where we use $\sigma_{35,0.6}(0) = 0$ and $\sigma_{35,0.6}(1) = 1$ in Eq. 5.2. To make the variable X_{B2} stable remaining 1, $L_{B2}(t+\Delta t)$ has to be 0:

$$\sigma_{\alpha_1,\beta_1}(W_{B2,A1} + W_{B2,C3}) = 0. \tag{B.3}$$

α_1 has to be large, which means that the light illumination to the amoeboid organism on the chip in the amoeba-based computing system is two selections whether to

illuminate it or not. The inside of the sigmoid function has to be smaller than β_2. Then, the stable condition is derived from Eq. B.3:

$$(W_{B2,A1} + W_{B2,C3})/\lambda < \beta_1 \quad \therefore \lambda > 2(W_{B2,A1} + W_{B2,C3}), \tag{B.4}$$

where $\beta_1 = 0.5$. In general, Eq. B.4 is given by

$$\lambda > 2\text{max}(\text{dist}(V, V') + \text{dist}(V', V'')) \tag{B.5}$$

where max$(\cdot)$ is a function that returns the maximum value from all combinations (V, V', V''). In advance of starting solution search, ν is calculated by using a conventional computer and the calculation time increases $O(N^3)$, where N is the number of cities. If λ is smaller than $2\text{max}(\text{dist}(V, V') + \text{dist}(V', V''))$, the system sometimes converges at an illegal solution; on the other hand, if λ is larger than it, exactly it converges at a legal solution. The value of ν is made as small as possible to amplify the distance between the cities.

B.2 Variable encoding and stabilization condition for Max-cut

In this section, I describe the variable encoding of the Max-cut and the stabilization condition. Because the Max-cut is stated as the vertex set is divided two subsets, U_1 and U_2 whose intersection is an empty set, we use redundant variables such as the SAT formulation: $X_{i,v}\,(i \in 1, 2, ..., N, v = 0, 1)$, where the vertex i belongs to U_1 when $X_{i,0} = 1$ and $X_{i,1} = 0$ and it belongs to U_2 when $X_{i,0} = 0$ and $X_{i,1} = 1$ as shown in Fig. B.2. The variable interactions for the bounce-back rule and Max-cut are given in Eq. 5.3

ν is also defined as with the bounceback rule for the TSP: by setting a proper

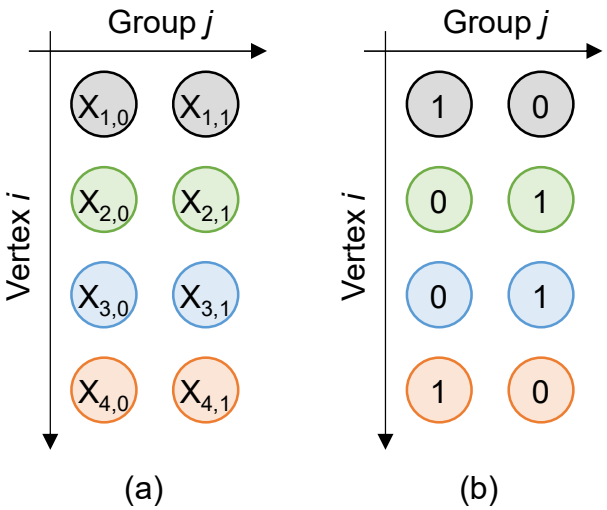

Figure B.2: (a) Variable encoding of Max-cut for amoeba-inspired computing system and (b) Solution example, which indicates $U_1 = \{1, 4\}$ and $U_2 = \{2, 3\}$.

value, the system remains stable when finding a solution. The product-sum operations about the vertex i is less than the threshold value β_1. Such a condition is given as follows:

$$\sum_j W_{iv,jv'}/\nu < \beta_1 \quad \therefore \nu > 2\sum_j W_{iv,jv'}. \tag{B.6}$$

Eq. B.6 is reformulated to generalize it using max$(\cdot)$ function as follows:

$$\nu > 2\text{max}\left(\sum_j W_{iv,jv'}\right), \tag{B.7}$$

for all i. ν can be calculated in $O(N^2)$ in advance of starting a solution search as with deriving λ of the TSP.

Appendix C

Improving solution quality by genetic-algorithm-based and fluctuation-introduced methods

C.1 Genetic-algorithm-based electronic amoeba

Obtained solutions in the electronic amoeba depend on initial values of resistances or device variations in the circuits as shown in Chap. 5. By hybridizing the electronic amoeba and the genetic algorithm (GA, [140, 148]), we can realize the improvement of the solution quality: we regard a combination of initial values as individuals in the GA. The GA is a multi-point search method that takes over superior individuals while crossing them. A method hybridizing the GA and the local search method such as 2-opt has been proposed because the GA is good at the global search while it is not good at the local search, which is called the Memetic Algorithm or Hybrid-GA [162–165]. To improve the quality of solutions in a quantum annealer, hybridizing the GA has been proposed [166]. Therefore, in the electronic amoeba, the GA-hybrid method is also effective for improving the quality.

To confirm that, I performed computer simulations by using the formulized be-

havior of the amoeba core for the Max-cut as follows:

$$X_{iv}(t+1) = \begin{cases} X_{iv}(t) + \dfrac{D_{in}N}{L_{off}(t+1)} & (\text{if } L_{iv}(t+1) = 0) \\ X_{iv}(t) - \dfrac{X_{iv}(t)}{D_{out}} & (\text{if } L_{iv}(t+1) = 1) \end{cases}, \tag{C.1}$$

$$L_{iv}(t+1) = \sigma_{\alpha_1,\beta_1}\left(\sum_{jv'} W_{iv,jv'}\sigma_{\alpha_2,\beta_2}(X_{jv'}(t))\right), \tag{C.2}$$

$$L_{off}(t+1) = \sum_{iv}(1 - L_{iv}(t+1)), \tag{C.3}$$

$$\sigma_{\alpha,\beta}(x) = \frac{1}{1+\exp(-\alpha(x-\beta))}, \tag{C.4}$$

where D_{in} and D_{out} are constant values that define the amount of resource supply and withdrawal, respectively, and L_{off} is the total number of $L_{iv} = 0$ that reproduces volume conservation, $W_{iv,jv'}$ is a variable interaction between X_{iv} and $X_{jv'}$ (shown in Chpt. 5), and $\sigma_{\alpha,\beta}$ is a sigmoid function that has slope α and a threshold value β. From the dependence of the solution quality on initial values, initial values (initial condition for each variable at $t = 0$) are regarded as individuals in the GA, and then, taking over superior individuals is applied. In particular, I performed a following algorithm as shown in Fig. C.1(a). The number of iterations is defined as the number of generations. Crossing is made by crossing probability P_c. As the generation passes, initial values that make the system converge to the better quality of solutions is selectively taken over, crossing initial values themself.

The used instance was a 100-vertex Max-cut. The number of individual was 100, and their initial values in the hybrid algorithm were set as the offset 0.53 with the gauss distributed random number whose average was 0 and standard deviation was 10^{-4}. The individuals to be taken over were selected by the tournament method whose size was 3. In addition to the tournament method, the roulette wheel-based selection method and the ranking based selection method have been known. I used

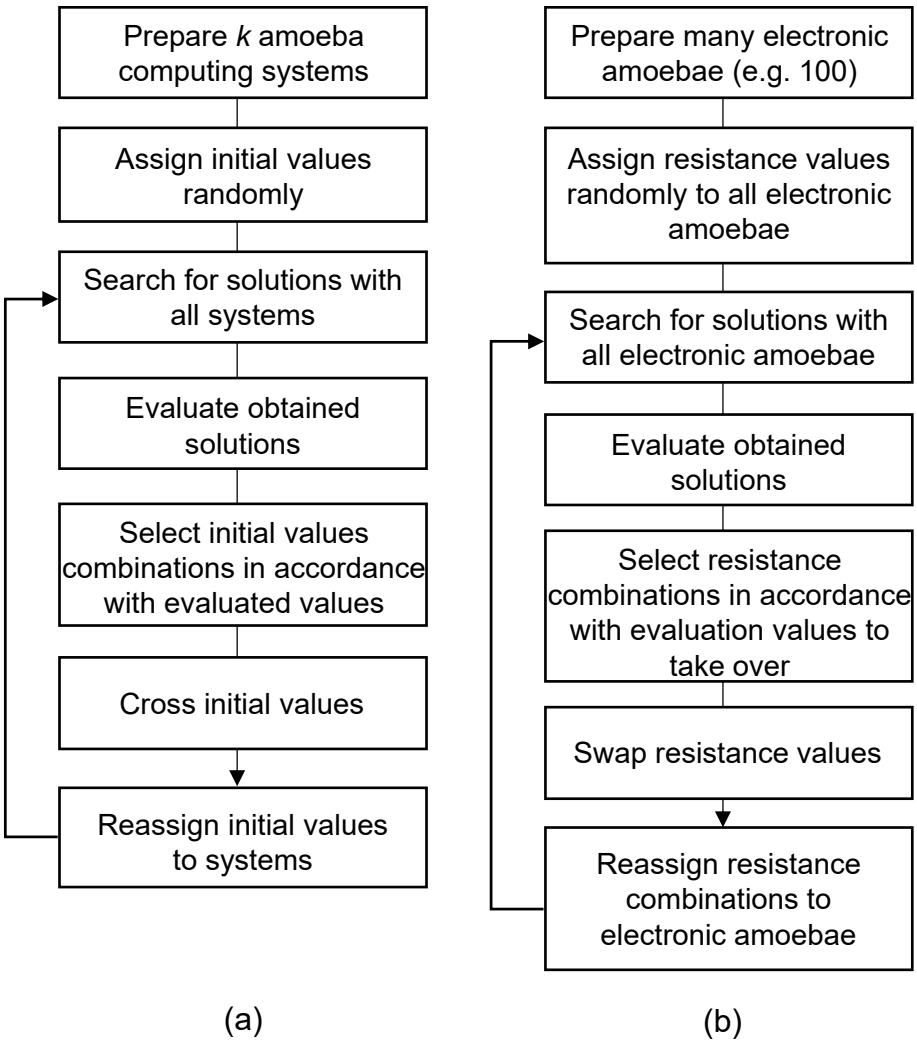

Figure C.1: Genetic-algorithm-based method in case of (a) recurrence formula and (b) electronic amoeba.

two-point crossing, and the crossing probability P_c was 0.9. In this simulation, I did not use the mutation in order to decrease hyper parameters. The parameters in Eqs. C.1 and C.2 were $\alpha_1 = 1000$, $\beta_1 = 0.5$, $\alpha_2 = 35$, $\beta_2 = 0.6$, $D_{in} = 0.03$, and $D_{out} = 500$.

Figure C.2 shows the computer simulation result. We confirmed that the average quality of obtained solutions and the best solution by the systems were improved as the generations passed. The improvement stopped at 30th generation. In this simulation, we could not obtain the optimal solution. This is because the global search was not enough to find the optimal solution: there happens initial convergence. The system falls into a local optimum. The introduction of the mutation increases the divergence and the system may have found the optimal solution.

The advantages of the hybridization of the electronic amoeba and GA are not only to improve the solution quality but also the robustness to dynamic optimization problems whose constraints and cost function change from hour to hour. The properties of practical applications in the real world often change as time passes. The amount of problem changes is smaller than that of original one: the change of an optimal solution (hamming distance) is considered to be small. Because the GA stores individuals that lead the system to find a better solution to the original problem, the GA-hybrid system easily adopts the changes [167, 168]. Therefore, the hybridization supplies the electronic amoeba with the robustness to the dynamic changes of problems. In the case of the electronic amoeba, one of the possible approaches that implement the GA-based method is shown in Fig. C.1(b), where the resistances in the amoeba core are regarded as individuals of the GA.

However, the GA-hybrid method requires the communication between the electronic amoeba and a conventional computer, leading an overhead that limits the performance of the electronic amoeba. Moreover, optimal settings of parameters in the GA have not been known despite the fact that empirical parameters setting has been known, and there is not a general solution. Therefore, we have to tune the

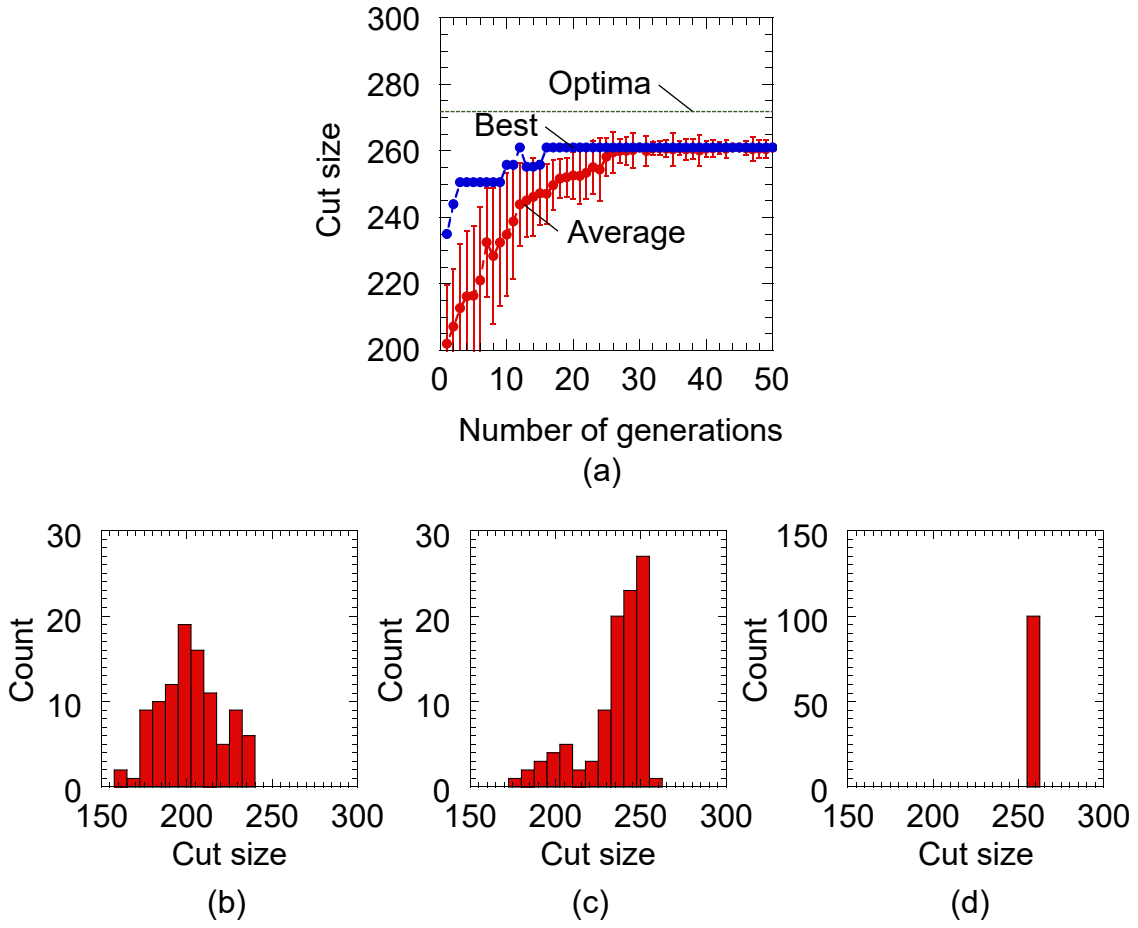

Figure C.2: Simulation results of GA-based algorithm. (a) Cut size as function of number of generations. Red and blue dots are average quality of all individuals and best quality, respectively. Broken green line is optimal solution. Histograms of (b) 1st, (c) 10th, and (d) 30th generations.

parameters that directly affect the quality of solutions and the solution-searching time.

C.2 Fluctuation-induced electronic amoeba

To improve the quality of solutions obtained by the electronic amoeba, another possible approach imposing fluctuation on the threshold of the sigmoid function in the system (changing the pseudopod length by the fluctuation) can be considered. Temporal fluctuation causes trial-and-error in the system. I demonstrated this concept by computer simulation. I used Eq. C.1, C.3, and C.4, although we changed the subscripts to solve the TSP. I reformulated Eq. C.2 as follows:

$$L_{Vk}(t+1) = \sigma_{\alpha_1,\beta_2}\left(\sum_{Ul} W_{Vk,Ul}\sigma_{\alpha_2,\beta_2}(X_{Ul}(t) + \delta_{Ul})\right), \tag{C.5}$$

where δ_{Ul} is a fluctuation that has no correlation between other variables.

Figure C.3 shows the obtained time series of the state variable $X_{1,0}$ and the sigmoid function $\sigma_{15,1.5}(X_{1,0}+\delta_{1,0})$ as an example for different fluctuation range $\delta_{1,0}$. The parameters were set to $\Delta_{in} = 0.00015$, $\Delta_{out} = 650$, $\alpha_1 = 1000$, $\beta_1 = 1.5$, $\alpha_2 = 15$, $\beta_2 = 1.5$, and $T = 1.5$. Initial values of X_{Vk} were all 1.37. $X_{1,0}$ shows a plateau after rapid decreases as shown in Figs. C.3(a)–(c), then it gradually decreased. The width of the plateau increases with the increase of $\delta_{1,0}$. When $X_{1,0}$ decreases, $\sigma_{15,1.5}$ also decreases. At the end of the solution search, $\sigma_{15,1.5}$ converges to 0 and became stable, although the finite fluctuation is still imposed.

Figure C.4 shows the simulation results, where Fig. C.4(a) and C.4(b) show the solution quality and the number of iterations to find a solution, respectively. We found that the solution quality was improved by increasing the amplitude of δ_{Vk}. Especially, the solution quality is improved rapidly when $\delta_{Vk} > 1.2$. On the other hand, the number of iterations to find a solution increases with the increase in δ_{Vk}.

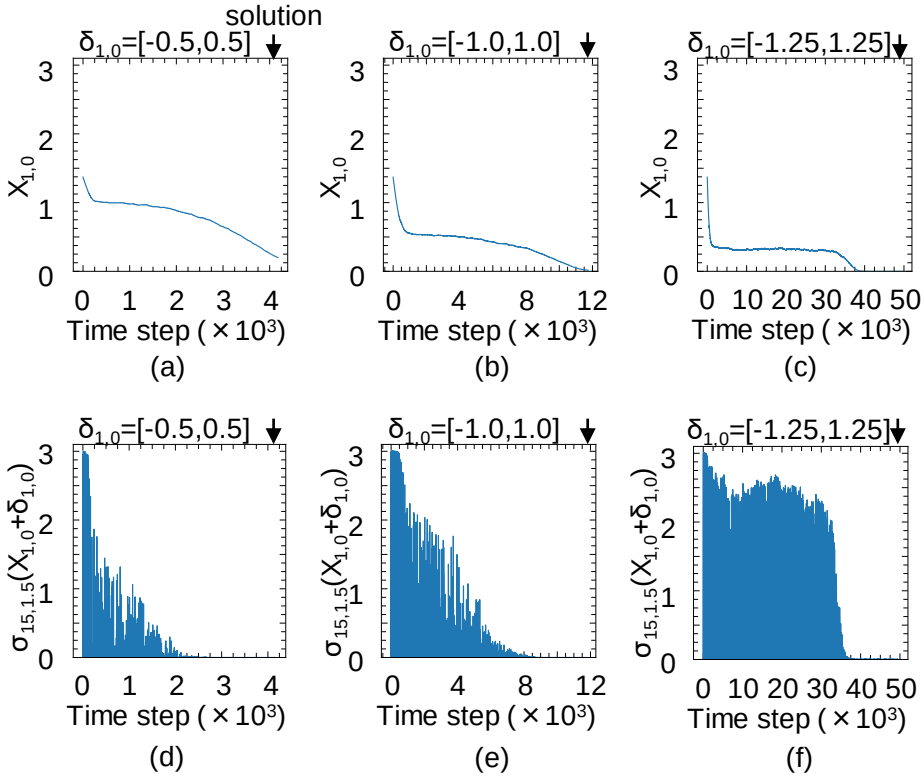

Figure C.3: Example of output waveforms of recurrence formula when solving 10-city instance. (a)–(c) Time evolution of $X_{1,0}$. (d)–(f) $\sigma_{15,1.5}(X_{1,0}+\delta_{1,0})$. Amplitudes of fluctuation (a) and (d) were $\delta_{1,0}=[-0.5,0.5]$, (b) and (e) were $\delta_{1,0}=[-1.0,1.0]$, and (c) and (f) were $\delta_{1,0}=[-1.25,1.25]$.

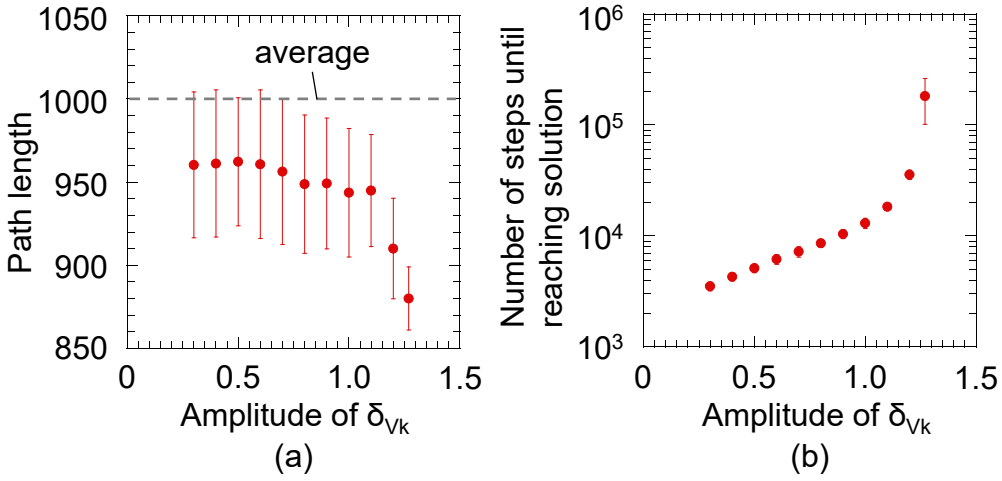

Figure C.4: Solution-searching characteristics of electronic amoeba with fluctuation obtained from computer simulations. (a) Obtained route length when solving 10-city TSP instance. Error bar is derived from 100 trials. (b) Number of steps to reach solution.

The result indicates the trade-off between the solution quality and solution-searching time. It looks like that the amoeba carefully searches for a solution when the large fluctuation is imposed. The minimum route length at large δ_{Vk} of 1.27 is 845.1, which is close to the minimum route length, 840.9, obtained by the brute-force search as shown in Fig. C.5.

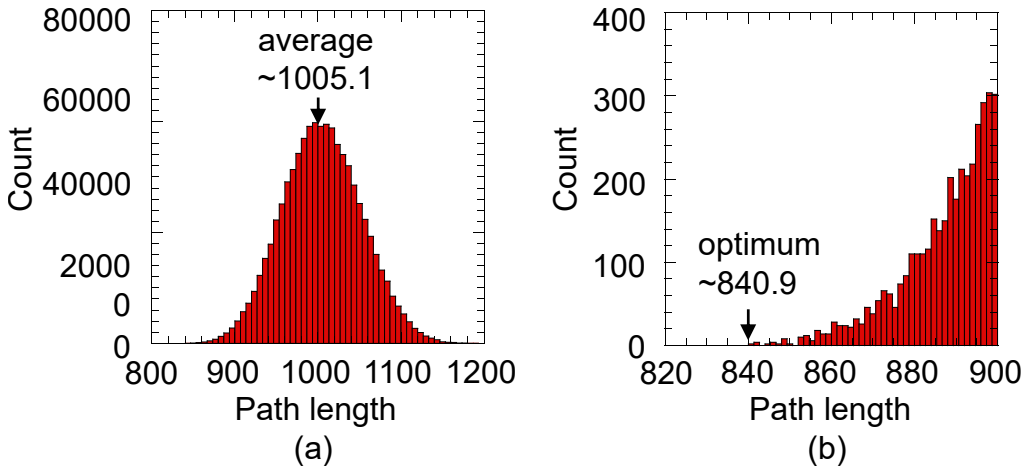

Figure C.5: (a) Histogram of 10-city TSP instance. (b) Enlarged view.

A possible approach of introducing the fluctuation into the electronic amoeba is shown in Fig. C.6. The threshold value of the sigmoid function of the branch can be modulated by the external noise.

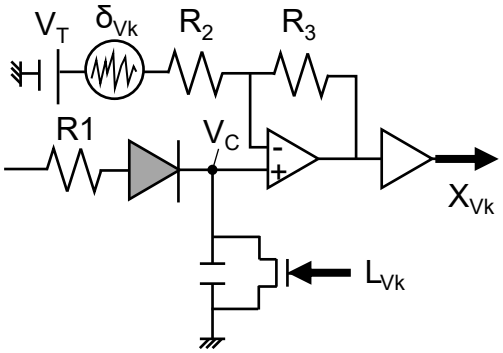

Figure C.6: Pseudopod circuit having sigmoid function whose threshold value is fluctuation by external noise.

The output voltage X_{Vk} is given by

$$X_{Vk} = \left(1 + \frac{R_3}{R_2}\right) V_C - \left(\frac{R_3}{R_2}\right) (V_T + \delta_{Vk}), \tag{C.6}$$

where $V_{ss} \geq X_{Vk} \geq V_{dd}$, V_T is a threshold voltage and δ_{Vk} is the fluctuation. The output voltage of the sigmoid function is similar to the rectified linear unit capped at V_{dd}. In the following simulation, I set the parameters, $V_{ss} = 0$ V, $V_{dd} = 3$ V, $V_T = 1.5$ V, $R_2 = 1$ kΩ, and $R_3 = 3$ kΩ. Figures C.7(d)–(f) show the output waveforms of the sigmoid function shown in Fig. C.6 with an external noise as shown in Figs. C.7(a)–(c), obtained from the circuit simulator.

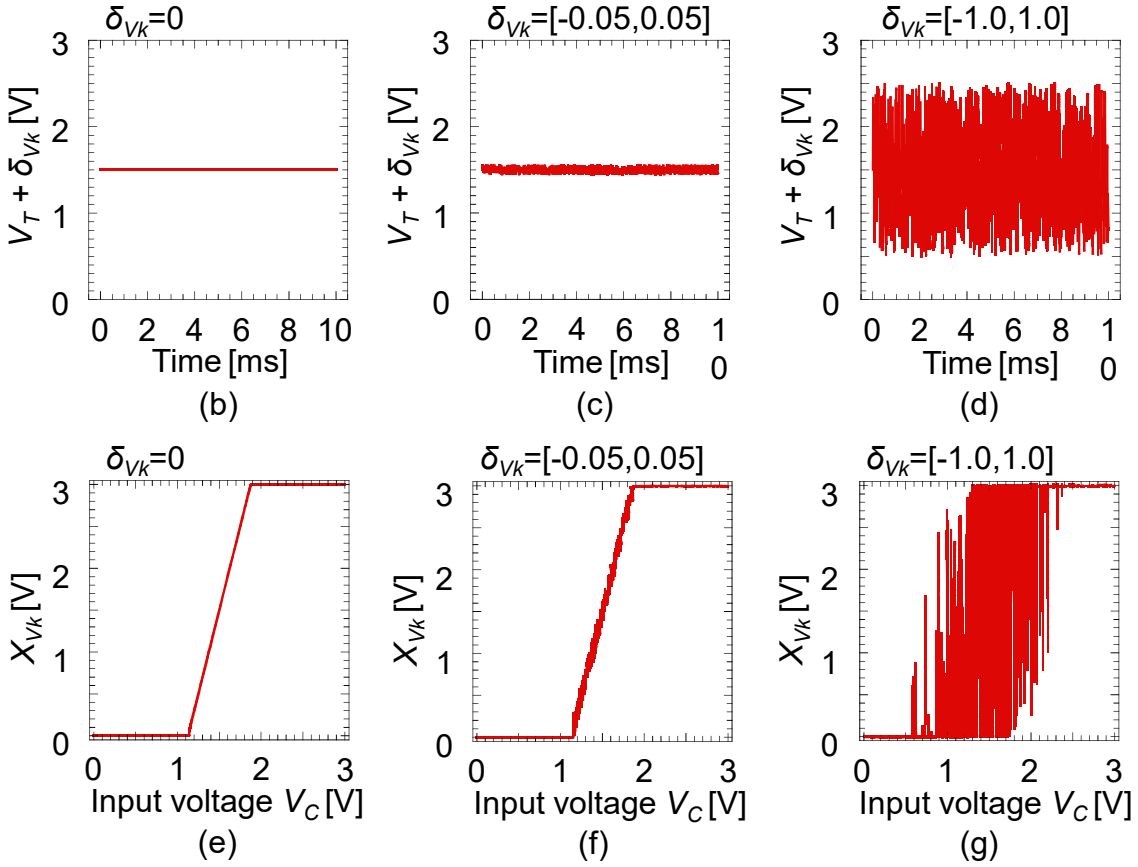

Figure C.7: Waveforms of X_{Vk}. (a)–(c) $V_T + \delta_{Vk}$ and (d)–(f) X_{Vk}. Amplitudes of fluctuation δ_{Vk} for (a) and (d) were 0, (b) and (e) were $[-0.005, 0.005]$, and (c) and (f) were $[-1.0, 1.0]$.

The sigmoid curve fluctuated largely when large noise was imposed. Note that

the dynamic range of the nonlinear output is extended by adding large noise when averaging X_{Vk}, which is sometimes explained by the context of stochastic resonance [169, 170]. At present, I consider that this extends the searchable range of the candidates of the solution for the amoeba core. More discussions are needed to clearly understand why we could improve the solution quality by imposing the fluctuation. One perspective is that the fluctuation produces trial-and-error behavior and the feedback of the IMC also fluctuates, and then the amoeba core integrates its fluctuation. As a result, we can obtain a better solution without trapping a local optimum. So far, an energy landscape such as the Ising machine and the Hopfield neural network have not been defined in the electronic amoeba; however, if such an energy landscape is specified in the electronic amoeba, a better solution has deeper landscape than a coarse solution.

A device that can generate stochastic behavior by itself is preferable to introduce the fluctuation to the electronic amoeba because such a device can easily connect to the electronic amoeba without a conventional computer. A hardware random number generator (HRNG) using intrinsic noise or stochastic behavior is the candidate. HRNGs, such as random switching resistive memories [116–120], have gathered attention in the area of hardware security and stochastic computing. HRNGs are seemed to be suitable for introducing the fluctuation to the electronic amoeba because they are low power consumption and small implementation area. Other noise implementation method is shown in Ref. [171]

Appendix D

Reformulation of bounce-back rule for TSP

The bounce-back rule (variable interactions) for the TSP is given by Refs. [32, 33] as follows:

$$W_{Vk,Ul} = \begin{cases} 0.5 & (\text{if } V = U \text{ at } k \neq l \text{ or } V \neq U \text{ at } k = l) \\ \nu \cdot \text{dist}(V,U) & (\text{if } V \neq U \text{ and } |k-l| = 1) \\ 0 & (\text{otherwise}) \end{cases} . \tag{5.4}$$

The general description for the bounce-back rule is shown in Chpt. 5. Eq. 5.4 does not consider the intercity distance between the first and last visited city (hereinafter referred to as the discontinuous-type bounce-back rule), as shown in Figs. D.1(a) and D.1(b). In Fig. D.1(a), the variable interactions about $X_{A,1}$ for the distance term do not correlate with last visited cities ($X_{B,3}$ and $X_{C,3}$), indicated by the red arrows; on the other hand, those about $X_{A,2}$ for that correlate with first and last visited cities ($X_{B,1}$, $X_{C,1}$, $X_{B,3}$, and $X_{C,3}$). Therefore, when we use the discontinuous bounce-back rule, the first and last visited cities tend not to be bounced back and the total route become longer with the distance between them. I reformulated Eq. 5.4

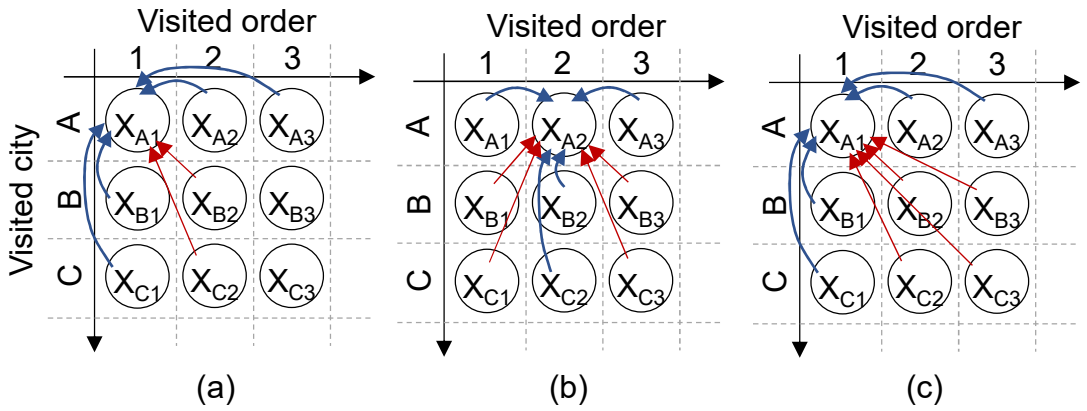

Figure D.1: (a) Variable interactions of 3-city TSP about X_{A1} and (b) $X_{A,2}$ for discontinuous bounce-back rule. Those about (c) $X_{A,1}$ for continuous bounce-back rule. Red and blue arrows indicate distance and constraint term, respectively.

to make them correlate as follows:

$$W_{Vk,Ul} = \begin{cases} 0.5 & (\text{if } V = U \text{ at } k \neq l \text{ or } V \neq U \text{ at } k = l) \\ \nu \cdot \text{dist}(V,U) & (\text{if } V \neq U \text{ and } (|k-l| = 1 \text{ or } |k-l| = N-1)) \cdot \\ 0 & (\text{otherwise}) \end{cases} \tag{6.5}$$

As shown in Fig. D.1(c), by adding $|k - l| = N - 1$, the last and first visited cities are correlated (hereinafter referred as to continuous-type bounce-back rule).

I confirmed the decrease in the distance when using the continuous-type bounce-back rule compared to using the discontinuous-type bounce-back rule. Figure D.2 shows the simulation results of discontinuous- and continuous-type bounce-back rules by using the AmoebaTSP given by Ref. [33], which is a mathematical model that reproduces the solution-searching behavior of the amoeboid organism. I tested 100 times for each instance. Used instances were 10–30 cities TSP whose intercity distance is 100 and standard deviation is 17. As shown in Fig. D.2(a), the total route of the continuous-type bounceback rule decreased, compared with the discontinuous one. The number of iterations of the continuous type was the same to the discontinuous one (shown in Fig. D.2(b)). The difference of the quality between

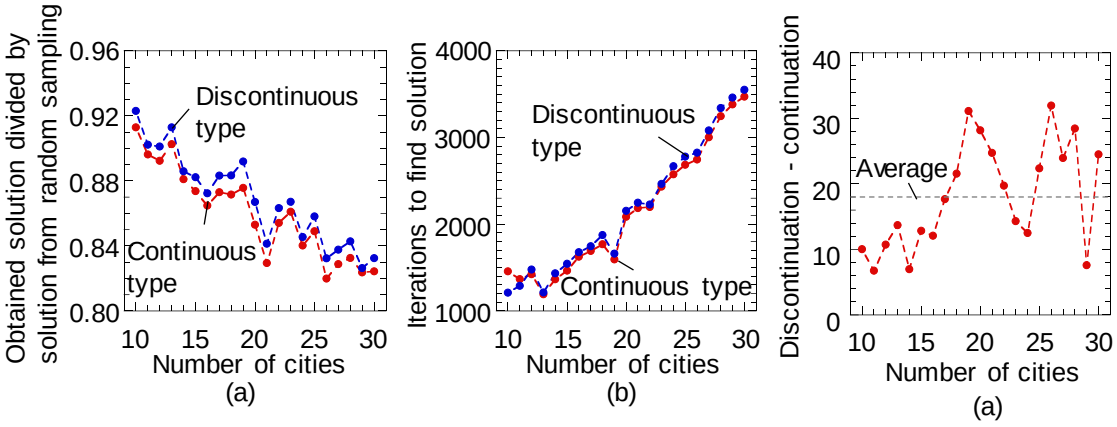

Figure D.2: Performance difference between continuous- and discontinuous-type bounce-back rule. (a) Solution quality, (b) iterations to find solution, and (c) performance differences between continuation and discontinuation.

the discontinuous and continuous type bounce-back rule is shown in Fig. D.2(c). The average difference was 18, which is the almost same as the standard deviation; therefore, we have concluded that the continuous-type bounce-back rule decreases the total route length by only one path.

Figure D.3 shows a histogram of obtained solutions by the distontinuous- and continusous-type bounce-back rule. As shown in Fig. D.3, obtained solutions by the discontinuous-type bounce-back rule were biased. This is due to the discontinuation. On the other hand, those by continuous-type one were not biased.

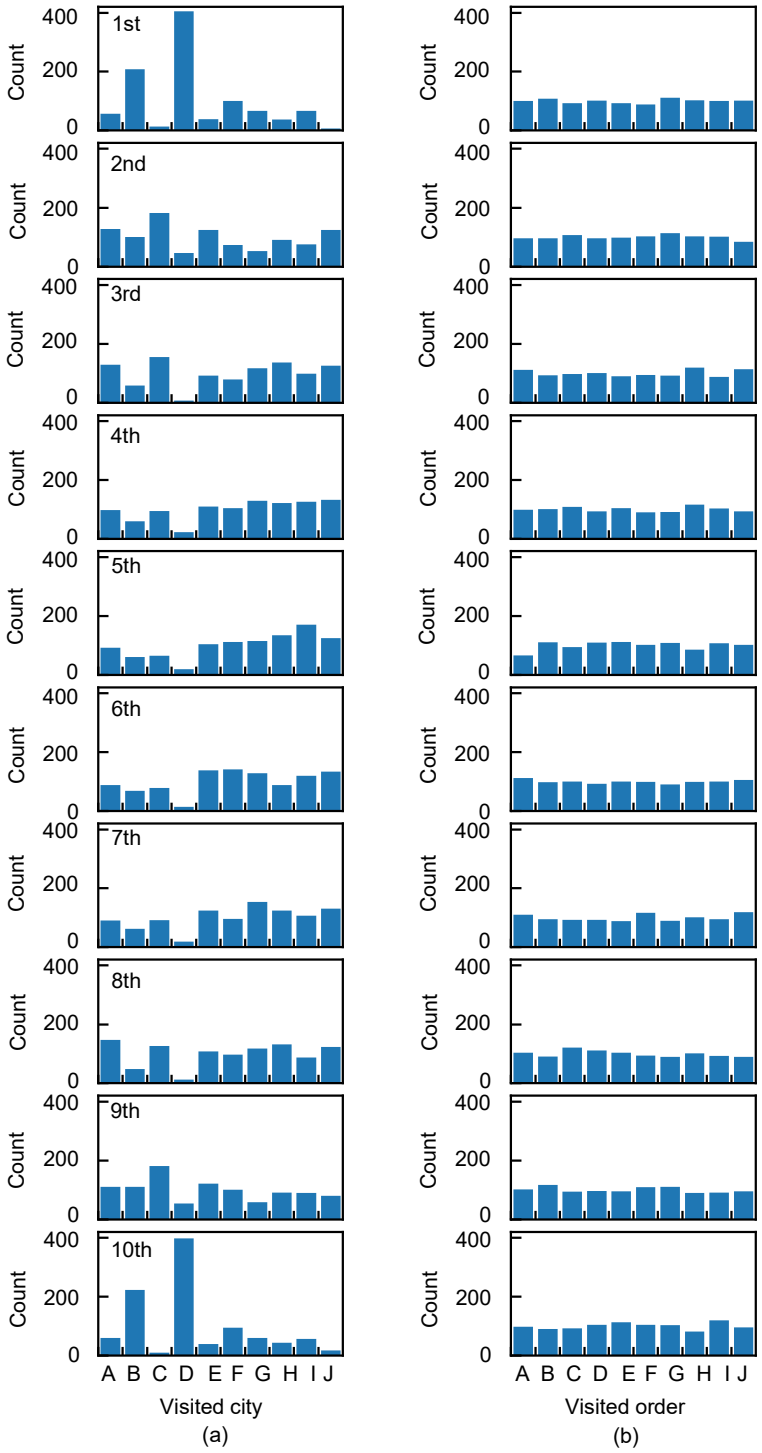

Figure D.3: Histogram of obtained solutions by (a) discontinuous- and (b) continuous-type bounce-back rule.

References

[1] F. Neukart, G. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, and B. Parney, *Front. ICT* **4**, 29 (2017).

[2] B. Wang, F. Hu, H. Yao, and C. Wang, *Sci. Rep.* **10**, 7106 (2020).

[3] N. Nishimura, K. Tanahashi, K. Suganuma, M. J. Miyama, and M. Ohzeki, *Front. Comput. Sci.* **1**, 2 (2019).

[4] M. Ohzeki, A. Miki, M. J. Miyama, and M. Terabe, *Front. Comput. Sci.* **1**, 9 (2019).

[5] K. Ikeda, Y. Nakamura, and T. S. Humble, *Sci. Rep.* **9**, 12837 (2019).

[6] M. Ohzeki, *Sci. Rep.* **10**, 3126 (2020).

[7] D. Venturelli, D. J. J. Marchand, and G. Rojo, "Job Shop Scheduling Solver based on Quantum Annealing," arXiv:1506.08479, 1–15 (2015).

[8] H. Ito, Y. Jiang, H. Yasuda, H. Takesue, K. Aihara, and M. Hasegawa, "High-Speed Optimization Method for Resource Allocation in Wireless Communication Systems by Coherent Ising Machine," in *2020 Int. Conf. Artif. Intell. Inf. Commun.*, 93–97 (2020).

[9] D. Inoue, A. Okada, T. Matsumori, K. Aihara, and H. Yoshida, "Traffic Signal Optimization on a Square Lattice using the D-Wave Quantum Annealer," arXiv:2003.07527, 1–11 (2020).

[10] S. Feld, C. Roch, T. Gabor, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, *Front. ICT* **6**, 13 (2019).

[11] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," in *Proc. Am. Math. Soc.*, 48–50 (1956).

[12] T. Bektas, *Omega* **34**, 209–219 (2006).

[13] G. E. Moore, *Electronics* **38**, 114–117 (1965).

[14] R. H. Dennard, F. H. Gaennsslen, H.-N. Yu, R. V. Leo, E. Bassous, and A. R. Leblanc, "Ion implantation and anneal to produce low resistance metal-diamond contacts," in *IEEE J. Solid-State Circuits*, 256–267 (1974).

[15] F. Peper, *New Gener. Comput.* **35**, 253–269 (2017).

[16] M. Waldrop, "The chips are down for Moore's law," in *Nat. News*, 144–147 (2016).

[17] J. L. Hennessy and D. A. Patterson, *Commun. ACM* **62**, 48–60 (2019).

[18] S. R. Nandakumar, S. R. Kulkarni, A. V. Babu, and B. Rajendran, *IEEE Nanotechnol. Mag.* **12**, 19–35 (2018).

[19] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose, *Nature* **473**, 194–198 (2011).

[20] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, *IEEE J. Solid-State Circuits* **51**, 303–309 (2016).

[21] V. Choi, *Quantum Inf. Process.* **7**, 193–209 (2008).

[22] V. Choi, *Quantum Inf. Process.* **10**, 343–353 (2011).

[23] J. Cai, B. Macready, and A. Roy, "A practical heuristic for finding graph minors," arXiv:1406.2741, 1–16 (2014).

[24] C. Klymko, B. D. Sullivan, and T. S. Humble, *Quantum Inf. Process.* **13**, 709–729 (2014).

[25] S. Okada, M. Ohzeki, M. Terabe, and S. Taguchi, *Sci. Rep.* **9**, 2098 (2019).

[26] Y. Sugie, Y. Yoshida, N. Mertig, T. Takemoto, H. Teramoto, A. Nakamura, I. Takigawa, S. I. Minato, M. Yamaoka, and T. Komatsuzaki, "Graph minors from simulated annealing for annealing machines with sparse connectivity," in *Theory Pract. Nat. Comput.*, 111–123 (2018).

[27] D. Oku, K. Terada, M. Hayashi, M. Yamaoka, S. Tanaka, and N. Togawa, *IEICE Trans. Inf. Syst.* **9**, 1696–1706 (2019).

[28] R. Hamerly, T. Inagaki, P. L. Mcmahon, D. Venturelli, T. Honjo, K. Enbutsu, T. Umeki, R. Kasahara, S. Utsunomiya, D. Englund, E. Rieffel, H. Takesue, and Y. Yamamoto, *Sci. Adv.* **5**, eaau0823 (2019).

[29] K. Takehara, D. Oku, Y. Matsuda, S. Tanaka, and N. Togawa, "A Multiple Coefficients Trial Method to Solve Combinatorial Optimization Problems for Simulated-annealing-based Ising Machines," in *IEEE Int. Conf. Consum. Electron.*, 64–69 (2019).

[30] T. Nakagaki, H. Yamada, and Á. Tóth, *Nature* **407**, 470 (2000).

[31] A. Tero, S. Takagi, T. Saiga, K. Ito, D. P. Bebber, M. D. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki, *Science* **327**, 439–442 (2010).

[32] L. Zhu, M. Aono, S. J. Kim, and M. Hara, *BioSystems* **112**, 1–10 (2013).

[33] L. Zhu, S. J. Kim, M. Hara, and M. Aono, *R. Soc. Open Sci.* **5**, 180396 (2018).

[34] S. Takagi, Y. Nishiura, T. Nakagaki, T. Ueda, and K.-I. Ueda, "Indecisive behavior of amoeba crossing an environmental barrier," in *Int. Symp. Topol. Asp. Crit. Syst. Networks*, 86–93 (2007).

[35] S. Kasai, M. Aono, and M. Naruse, *Appl. Phys. Lett.* **103**, 163703 (2013).

[36] R. M. Karp, *Reducibility among combinatorial problems*, Boston: Springer, 85–103 (1972).

[37] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proc. third Annu. ACM Symp. Theory Comput.*, 151–158 (1971).

[38] M. Davis and H. Putnam, *J. ACM* **7**, 201–215 (1960).

[39] J. P. Marques-Silva and K. A. Sakallah, *IEEE Trans. Comput.* **48**, 506–521 (1999).

[40] U. Schoning, "A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 410–414 (1999).

[41] A. Balint and U. Schoning, "Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break," in *Theory Appl. Satisf. Test.*, 16–29 (2012).

[42] B. Selman, D. G. Mitchell, and H. J. Levesque, *Artif. Intell.* **81**, 17–29 (1996).

[43] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*, NLD: IOS Press (2009).

[44] A. Tero, R. Kobayashi, and T. Nakagaki, *J. Theor. Biol.* **244**, 553–564 (2007).

[45] M. Aono, M. Hara, and K. Aihara, *Commun. ACM* **50**, 69–72 (2007).

[46] M. Aono, M. Naruse, S. J. Kim, M. Wakabayashi, H. Hori, M. Ohtsu, and M. Hara, *Langmuir* **29**, 7557–7564 (2013).

[47] M. Aono, S. Kasai, S. J. Kim, M. Wakabayashi, H. Miwa, and M. Naruse, *Nanotechnology* **26**, 234001 (2015).

[48] "Amoeba finds approximate solutions to NP-hard problem in linear time," https://phys.org/news/2018-12-amoeba-approximate-solutions-np-hard-problem.html.

[49] M. Aono, S.-j. Kim, L. Zhu, M. Naruse, M. Ohtsu, H. Hori, and M. Hara, "Amoeba-inspired SAT Solver," in *Int. Symp. Nonlinear Theory its Appl.*, 586–589 (2012).

[50] M. Aono, *Jpn. J. Appl. Phys.* **59**, 060502 (2020).

[51] Y. Hara-Azumi, N. Takeuchi, K. Hara, and M. Aono, *Jpn. J. Appl. Phys.* **59**, 040603 (2020).

[52] N. Takeuchi, M. Aono, and N. Yoshikawa, *Phys. Rev. Appl.* **11**, 044069 (2019).

[53] K. Hara, N. Takeuchi, M. Aono, and Y. Hara-azumi, "Amoeba-Inspired Stochastic Hardware SAT Solver," in *Int. Symp. Qual. Electron. Des.*, 151–156 (2019).

[54] A. H. N. Nguyen, M. Aono, and Y. Hara-Azumi, *IEEE Access* **4**, 49053–49065 (2020).

[55] A. Hoang and N. Nguyen, "Amoeba-Inspired Hardware SAT Solver with Effective Feedback Control," in *Int. Conf. Field-Programmable Technol.*, 243–246 (2019a).

[56] A. Hoang and N. Nguyen, "FPGA-Based Amoeba-Inspired SAT Solver for Cyber-Physical Systems," in *Int. Conf. Cyber-Physical Syst.*, 316–317 (2019b).

[57] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, *Front. Phys.* **7**, 48 (2019).

[58] H. Goto, K. Tatsumura, and A. R. Dixon, *Sci. Adv.* **5**, eaav2372 (2019).

[59] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Natl. Acad. Sci. USA*, 2554–2558 (1982).

[60] J. J. Hopfield and D. W. Tank, *Biol. Cybern.* **52**, 141–152 (1985).

[61] M. Krivan and B. Budinska, *Math. Appl.* **4**, 109–121 (2015).

[62] G. V. Wilson and G. S. Pawley, *Biol. Cybern.* **58**, 63–70 (1988).

[63] B. Kamgar-Parsi and B. Kamgar-Parsi, *Biol. Cybern.* **62**, 415–423 (1990).

[64] K. Tanahashi, S. Takayanagi, T. Motohashi, and S. Tanaka, *J. Phys. Soc. Japan* **88**, 061010 (2019).

[65] S. Tanaka, Y. Matsuda, and N. Togawa, "Theory of Ising Machines and a Common Software Platform for Ising Machines," in *Proc. Asia South Pacific Des. Autom. Conf. ASP-DAC*, 659–666 (2020).

[66] E. Ising, *Zeitschrift für Phys.* **31**, 253–258 (1925).

[67] A. Lucas, *Front. Pysics* **2**, 1–15 (2014).

[68] S. Kirkpatrick, *J. Stat. Phys.* **34**, 975–986 (1984).

[69] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handb. Metaheuristics*: Springer, 1–35 (2019).

[70] T. Kadowaki and H. Nishimori, *Phys. Rev. E* **58**, 5355–5363 (1998).

[71] C. Yoshimura, M. Hayashi, T. Takemoto, and M. Yamaoka, "CMOS Annealing Machine: A Domain-Specific Architecture for Combinatorial Optimization

Problem," in *Proc. Asia South Pacific Des. Autom. Conf. ASP-DAC*, 673–678 (2020).

[72] T. Okuyama, C. Yoshimura, M. Hayashi, and M. Yamaoka, "Computing architecture to perform approximated simulated annealing for Ising models," in *2016 IEEE Int. Conf. Rebooting Comput.*, 1–8 (2016).

[73] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, *Int. J. Netw.* **7**, 154–172 (2017a).

[74] T. Taketomo, M. Hayashi, C. Yoshimura, and M. Yamaoka, "A 2*30k-Spin Multichip Scalable Annealing Processor Based on a Processing-In-Memory Approach for Solving Large-Scale Combinatorial Optimization Problems," in *2019 IEEE Int. Solid- State Circuits Conf.*, 52–54 (2019).

[75] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "FPGA-based annealing processor for ising model," in *Int. Symp. Comput. Netw.*, 436–442 (2017b).

[76] T. Okuyama, T. Sonobe, K.-i. Kawarabayashi, and M. Yamaoka, *Phys. Rev. E* **100**, 12111 (2019).

[77] H. Goto, *J. Phys. Soc. Japan* **88**, 061015 (2019).

[78] H. Goto, *Sci. Rep.* **6**, 21686 (2016).

[79] K. Tatsumura, A. R. Dixon, and H. Goto, "FPGA-based Simulated Bifurcation Machine," in *Int. Conf. F. Program. Log. Appl.*, 59–66 (2019).

[80] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu, O. Tadanaga, H. Takenouchi, K. Aihara, K. Kawarabayashi, K. Inoue, S. Utsunomiya, and H. Takesue, *Science* **354**, 603–606 (2016).

[81] Y. Yamamoto, K. Aihara, T. Leleu, K.-i. Kawarabayashi, S. Kako, M. Fejer, K. Inoue, and H. Takesue, *npj Quantum Inf.* **3**, 49 (2017).

[82] Y. Yamamoto, T. Leleu, S. Ganguli, and H. Mabuchi, "Coherent Ising machines – Quantum optics and neural network perspectives," arXiv:2006.05649, 1–32 (2020).

[83] T. Wang and J. Roychowdhury, "OIM: Oscillator-based Ising Machines for Solving Combinatorial Optimisation Problems," arXiv:1903.07163, 1–20 (2019).

[84] J. Chou, S. Bramhavar, S. Ghosh, and W. Herzog, *Sci. Rep.* **9**, 14786 (2019).

[85] A. Raychowdhury, A. Parihar, G. H. Smith, V. Narayanan, G. Csaba, M. Jerry, W. Porod, and S. Datta, "Computing with Networks of Oscillatory Dynamical Systems," in *Proc. IEEE*, 73–89 (2019).

[86] R. Afoakwa, Y. Zhang, U. K. R. Vengalam, Z. Ignjatovic, and M. Huang, "CMOS Ising Machines with Coupled Bistable Nodes," arXiv:2007.06665 (2020).

[87] T. Wang and L. Wu, "Late Breaking Results : New Computational Results and Hardware Prototypes for Oscillator-based Ising Machines," arXiv:1302.5843, 1–2 (2019).

[88] M. Babaeian, D. T. Nguyen, V. Demir, M. Akbulut, P. A. Blanche, Y. Kaneda, S. Guha, M. A. Neifeld, and N. Peyghambarian, *Nature Commun.* **10**, 3516 (2019).

[89] J. H. Shin, Y. J. Jeong, M. A. Zidan, Q. Wang, and W. D. Lu, "Hardware Acceleration of Simulated Annealing of Spin Glass by RRAM Crossbar Array," in *Int. Electron Devices Meet.*, 3.3.1–3.3.4 (2018).

[90] F. Cai, S. Kumar, T. Van Vaerenbergh, X. Sheng, R. Liu, C. Li, Z. Liu, M. Foltin, S. Yu, Q. Xia, J. J. Yang, R. Beausoleil, W. D. Lu, and J. P. Strachan, *Nat. Electron.* **3**, 409–418 (2020).

[91] S. Kumar, J. P. Strachan, and R. S. Williams, *Nature* **548**, 318–321 (2017).

[92] A. Minamisawa, R. Iimura, and T. Kawahara, "High-speed Sparse Ising Model on FPGA," in *Midwest Symp. Circuits Syst.*, 670–673 (2019).

[93] H. Gyoten, M. Hiromoto, and T. Sato, "Enhancing the solution quality of hardware ising-model solver via parallel tempering," in *2018 IEEE/ACM Int. Conf. Comput. Des.*, 1–8 (2018).

[94] K. Yamamoto, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki, and M. Motomura, "STATICA: A 512-Spin 0.25M-Weight Full-Digital Annealing Processor with a Near-Memory All-Spin-Updates-at-Once Architecture for Combinatorial Optimization with Complete Spin-Spin Interactions," in *IEEE Int. Solid-State Circuits Conf.*, 138–140 (2020).

[95] R. Ono, K. Someya, and T. Kawahara, *Microprocess. Microsyst.* **78**, 103251 (2020).

[96] K. Someya, R. Ono, and T. Kawahara, "Novel Ising model using dimension-control for high-speed solver for Ising machines," in *14th IEEE Int. NEWCAS Conf. NEWCAS 2016*, 1–4 (2016).

[97] S. Kitamura, R. Iimura, and T. Kawahara, "AI Chips on Things for Sustainable Society: A 28-nm CMOS, Fully Spin-to-spin Connected 512-Spin, Multi-Spin-Thread, Folded Halved-Interaction Circuits Method, Annealing Processing Chip," in *IEEE 18th World Symp. Appl. Mach. Intell. Informatics*, 319–323 (2020).

[98] R. Yasudo, K. Nakano, Y. Ito, M. Tatekawa, R. Katsuki, T. Yazane, and Y. Inaba, "Adaptive Bulk Search: Solving Quadratic Unconstrained Binary Optimization Problems on Multiple GPUs," in *49th Int. Converence Parallel Process.*, 1–11 (2020).

[99] H. M. Waidyasooriya and M. Hariyama, *IEEE Access* **8**, 67929–67939 (2020).

[100] C. Cook, H. Zhao, T. Sato, M. Hiromoto, and S. X. Tan, *Integr. VLSI J.* **69**, 335–344 (2019).

[101] D-Wave Systems Inc., "Postprocessing Methods on D-Wave Systems," technical report (2018), URL: `http://www.dwavesys.com`.

[102] G. Marsaglia, *J. Stat. Softw.* **8**, 1–6 (2003).

[103] A. V. Chervyakov, D. O. Sinitsyn, M. A. Piradov, M. Lebedev, H. Merchant, and Y. Sakurai, *Front. Hum. Neurosci.* **10**, 603 (2016).

[104] I. Dinstein, D. J. Heeger, and M. Behrmann, *Trends Cogn. Sci.* **19**, 322–328 (2015).

[105] T. Masquelier, *Front. Comput. Neurosci.* **7**, 1–7 (2013).

[106] J. Grollier, D. Querlioz, K. Y. Camsari, K. Everschor-Sitte, S. Fukami, and M. D. Stiles, *Nat. Electron.* **3**, 360–370 (2020).

[107] "SATLIB," https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

[108] J. Xu and J. Zhang, "Exploration-exploitation tradeoffs in metaheuristics: Survey and analysis," in *Proc. 33rd Chinese Control Conf.*, 8633–8638 (2014).

[109] C. Blum and A. Roli, *ACM Comput. Surv.* **35**, 268–308 (2003).

[110] J. Palmer, A. C. Huk, and M. N. Shadlen, *J. Vis.* **5**, 376–404 (2005).

[111] Y. Tanimoto, A. Yamazoe-Umemoto, K. Fujita, Y. Kawazoe, Y. Miyanishi, S. J. Yamazaki, X. Fei, K. E. Busch, K. Gengyo-Ando, J. Nakai, Y. Iino, Y. Iwasaki, K. Hashimoto, and K. D. Kimura, *Elife* **6**, e21629 (2017).

[112] S. DasGupta, C. H. Ferreira, and G. Miesenbock, *Science* **344**, 901–904 (2014).

[113] B. W. Brunton, M. M. Botvinick, and C. D. Brody, *Science* **340**, 95–99 (2013).

[114] J. I. Gold and M. N. Shadlen, *Annu. Rev. Neurosci.* **30**, 535–574 (2007).

[115] M. R. Mahmoodi, M. Prezioso, and D. B. Strukov, *Nature Commun.* **10**, 5113 (2019).

[116] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, *Nanoscale* **5**, 5872–5878 (2013).

[117] H. Jiang, D. Belkin, S. E. Savel'Ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, Q. Wu, J. J. Yang, and Q. Xia, *Nature Commun.* **8**, 882 (2017).

[118] S. Balatti, S. Ambrogio, Z. Wang, and D. Ielmini, *IEEE J. Emerg. Sel. Top. Circuits Syst.* **5**, 214–221 (2015).

[119] T. Tuma, A. Pantazi, M. Le Gallo, A. Sebastian, and E. Eleftheriou, *Nat. Nanotechnol.* **11**, 693–699 (2016).

[120] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J. O. Klein, S. Galdin-Retailleau, and D. Querlioz, *IEEE Trans. Biomed. Circuits Syst.* **9**, 166–174 (2015).

[121] S. Balatti, S. Ambrogio, R. Carboni, V. Milo, Z. Wang, A. Calderoni, N. Ramaswamy, and D. Ielmini, *IEEE Trans. Electron Devices* **63**, 2029–2035 (2016).

[122] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, *Nat. Nanotechnol.* **15**, 529–544 (2020).

[123] S. Choi, Y. Yang, and W. Lu, *Nanoscale* **6**, 400–404 (2014).

[124] W. A. Wulf and S. A. McKee, *ACM SIGARCH Comput. Archit. News* **23**, 20–24 (1995).

[125] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, *IEEE Micro* **17**, 34–44 (1997).

[126] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, *IBM J. Res. Dev.* **63**, 1–19 (2019).

[127] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, *Adv. Mater.* **30**, 1705914 (2018).

[128] T. Gokmen, M. J. Rasch, and W. Haensch, *Front. Neurosci.* **12**, 745 (2018).

[129] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, *Nature* **577**, 641–646 (2020).

[130] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, 14–26 (2016).

[131] W. A. Borders, H. Akima, S. Fukami, S. Moriya, S. Kurihara, Y. Horio, S. Sato, and H. Ohno, *Appl. Phys. Express* **10**, 013007 (2017).

[132] C. X. Xue, W. H. Chen, J. S. Liu, J. F. Li, W. Y. Lin, W. E. Lin, J. H. Wang, W. C. Wei, T. W. Chang, T. C. Chang, T. Y. Huang, H. Y. Kao, S. Y. Wei, Y. C. Chiu, C. Y. Lee, C. C. Lo, Y. C. King, C. J. Lin, R. S. Liu, C. C.

Hsieh, K. T. Tang, and M. F. Chang, "A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN-Based AI Edge Processors," in *IEEE Int. Solid-State Circuits Conf.*, 388–390 (2019).

[133] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, and S. Datta, *Nature* **573**, 390–393 (2019).

[134] H. Yeon, P. Lin, C. Choi, S. H. Tan, Y. Park, D. Lee, J. Lee, F. Xu, B. Gao, H. Wu, H. Qian, Y. Nie, S. Kim, and J. Kim, *Nat. Nanotechnol.* **15**, 574–579 (2020).

[135] M. Jerry, P.-Y. Chen, P. Sharma, K. Ni, S. Yu, and S. Datta, "Ferroelectric FET Analog Synapse for Acceleration of Deep Neural Network Training," in *Int. Electron Devices Meet.*, 6.2.1–6.2.4 (2017).

[136] "LTspice," https://www.analog.com/jp/design-center/design-tools-and-calculators/ltspice-simulator.html.

[137] J. Bruck, "On the Convergence Properties of the Hopfield Model," in *Proc. IEEE*, 1579–1585 (1990).

[138] Z. B. Xu, G. Q. Hu, and C. P. Kwong, *Neural Networks* **9**, 483–501 (1996).

[139] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Science* **220**, 671–680 (1983).

[140] D. Whitley, *Stat. Comput.* **4**, 65–85 (1994).

[141] K. Wang, L. Huang, C. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proc. Second Int. Conf. Mach. Learn. Cybern.*, 1583–1585 (2003).

[142] M. Dorigo and L. M. Gambardella, *IEEE Trans. Evol. Comput.* **1**, 53–66 (1997).

[143] S. Lin and B. W. Kernighan, *Oper. Res.* **21**, 498–516 (1973).

[144] L. Xia, B. Li, T. Tang, P. Gu, P. Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **37**, 1009–1022 (2018).

[145] T. Tohara, H. Liang, H. Tanaka, M. Igarashi, S. Samukawa, K. Endo, Y. Takahashi, and T. Morie, *Appl. Phys. Express* **9**, 03420 (2016).

[146] T. Morie, H. Liang, T. Tohara, H. Tanaka, M. Igarashi, S. Samukawa, K. Endo, and Y. Takahashi, "Spike-based time-domain weighted-sum calculation using nanodevices for low power operation," in *Int. Conf. Nanotechnol.*, 390–392 (2016).

[147] Q. Wang, H. Tamukoh, and T. Morie, "Time-Domain Weighted-Sum Calculation for Ultimately Low Power VLSI Neural Networks," in *Int. Conf. Neural Inf. Process.*, 758–759 (2017).

[148] N. M. Razali and J. Geraghty, "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP," in *Proc. World Congr. Eng.*, 1134–1139 (2011).

[149] G. Reinelt, *The Traveling Salesman-Computational Solutions for TSP Applications*, Berlin: Springer, 1–223 (1994).

[150] E. Thomas, *Applied Delay Differential Equations*, New York: Springer, 1–204 (2009).

[151] M. C. Mackey and L. Glass, *Science* **197**, 287–289 (1977).

[152] K. Pyragas, *Phys. Lett. A* **170**, 421–428 (1992).

[153] K. Konishi, H. Kokame, and K. Hirata, *Phys. Rev. E* **60**, 4000–4007 (1999).

[154] L. Appeltant, M. C. Soriano, G. Van Der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, *Nature Commun.* **2**, 468 (2011).

[155] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, *Nature Commun.* **4**, 1364 (2013).

[156] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, *Neural Networks* **115**, 100–123 (2019).

[157] B. S. Lee, *J. Acoust. Soc. Am.* **22**, 824–826 (1950).

[158] T. Toya, D. Ishikawa, R. Miyauchi, K. Nishimoto, and M. Unoki, *J. Signal Process.* **20**, 197–200 (2016).

[159] J. Foss and J. Milton, *J. Neurophysiol.* **84**, 975–985 (2000).

[160] Y. Nourani and B. Andresen, *J. Phys. A: Math. Gen.* **31**, 8373–8385 (1998).

[161] S. Geman and D. Geman, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**, 721–741 (1984).

[162] Y. Watanabe, N. Mizuguchi, and Y. Fujii, *Syst. Comput. Japan* **29**, 68–74 (1998).

[163] S. Salcedo-Sanz and X. Yao, *IEEE Trans. Syst. Man, Cybern. Part B Cybern.* **34**, 2343–2353 (2004).

[164] M. Matayoshi, M. Nakamura, and H. Miyagi, "A Genetic Algorithm with the Improved 2-opt Method," in *IEEE Int. Conf. Syst. Man Cybern.*, 3652–3658 (2004).

[165] F. Neri and C. Cotta, *Swarm Evol. Comput.* **2**, 1–14 (2012).

[166] J. King, M. Mohseni, W. Bernoudy, A. Fréchette, H. Sadeghi, S. V. Isakov, H. Neven, and M. H. Amin, "Quantum-Assisted Genetic Algorithm," arXiv:1907.00707, 1–13 (2019).

[167] A. Zhou, L. Kang, and Z. Yan, "Solving dynamic TSP with evolutionary approach in real time," in *Congr. Evol. Comput.*, 951–957 (2003).

[168] J. Branke and Y. Jin, *IEEE Trans. Evol. Comput.* **9**, 303–317 (2005).

[169] L. Gammaitoni, *Phys. Rev. E* **52**, 4691–4698 (1995).

[170] Y. Tadokoro, S. Kasai, A. Ichiki, and H. Tanaka, *IEEE Trans. Syst. Man, Cybern. Syst.* **46**, 1121–1128 (2016).

[171] S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, and L. Chakrapani, "A probabilistic CMOS switch and its realization by exploiting noise," in *IFIP Intl. Conf. VLSI*, 535–541 (2005).

List of publications/conferences/awards

Publications related to this work

[1] Kenta Saito, Naoki Suefuji, Seiya Kasai, and Masashi Aono, "Amoeba-inspired Electronic Computing System and its Application to Autonomous Walking of a Multi-legged Robot," Journal of Applied Logics - ifCoLoG Journal of Logics and their Applications 5, pp. 1799-1814 (2018).

[2] Kenta Saito and Seiya Kasai, "Effect of feedback delays on solution quality in amoeba-inspired computing system that solves traveling salesman problem," Applied Physics Express 13, 114501 (2020).

[3] Kenta Saito, Masashi Aono, and Seiya Kasai, "Amoeba-inspired Analog Electronic Computing System Integrating Resistance Crossbar for Solving the Traveling Salesman Problem," Scientific Reports 10, 20772 (2020).

[4] Kenta Saito and Seiya Kasai, "Effect of Asymmetric Deformation Dynamics in Amoeboid Organism on Its Search Ability," Bioinspiration & Biomimetics (2021, under review).

[5] 斉藤健太、青野真士、葛西誠也、「抵抗クロスバー回路を備えた粘菌型アナログ電子解探索システムにおける最大カット問題解法」電子情報通信学会論文誌 C・

招待論文（2021、under review）

Presentations related to this work

International conference

[1] Kenta Saito, Seiya Kasai, and Masashi Aono, “Impact of External Fluctuation on Solution Search in Amoeba-inspired Electronic Computing System,” Workshop on Molecular Architectonics - Toward Realization of Neuromorphic Computing by Nanomaterials, Osaka, June 2017.

[2] Kenta Saito, Naoki Suefuji, Seiya Kasai, and Masashi Aono, “Amoeba-Inspired Electronic Solution-Searching System and Its Application to Finding Walking Maneuver of a Multi-legged Robot,” The 48th International Symposium on Multiple-Valued Logic, Linz, Austria, May 2018.

[3] Kenta Saito, Seiya Kasai, and Masashi Aono, “Evaluation of Solution Search Performance of Amoeba-inspired Electronic Computing System for Solving Maximum Cut Problem,” Proceedings of 2020 International Symposium on Nonlinear Theory and Its Applications, Virtual, November 2020.

Domestic conference

[1] 斉藤健太、若宮遼、葛西誠也、「粘菌アメーバ型最適化問題解探索電子システムの再構成・大規模化の検討」電子情報通信学会北海道支部学生会インターネットシンポジウム、2016 年 2 月

[2] 斉藤健太、葛西誠也、青野真士、「電子アメーバにおけるエラーと SAT 解探索効率の関係」、電子情報通信学会ソサイエティ大会、札幌、2016 年 9 月

[3] 斉藤健太、葛西誠也、青野真士、「電子アメーバ最適化問題解探索における外乱の効果」、第 64 回応用物理学会春季学術講演会、横浜、2017 年 3 月

[4] 斉藤健太、葛西誠也、青野真士、「電子アメーバ SAT 解探索システムにおけるエラーと解探索効率の相関」、電子情報通信学会総合大会、金沢、2018 年 9 月

[5] 斉藤健太、末藤直樹、葛西誠也、青野真士、「粘菌に着想を得た TSP 解探索アルゴリズムの電子回路実装」、第 66 回応用物理学会春季学術講演会、東京、2019 年 3 月

[6] 斉藤健太、末藤直樹、葛西誠也、青野真士、「アメーバ電子計算システムにおける最大カット問題のマッピングとその求解」、第 80 回応用物理学会秋季学術講演会、札幌、2019 年 9 月

[7] 斉藤健太、末藤直樹、葛西誠也、青野真士、「巡回セールスマン問題に対するアメーバ電子計算システムの解探索性能」、第 67 回応用物理学会春季学術講演会（新型コロナの影響により講演会中止）

[8] 斉藤健太、葛西誠也、青野真士、「アナログ電子アメーバにおける遅延による不安定状態の解探索性能への影響評価」、第 81 回応用物理学会秋季学術講演会、オンライン、2020 年 9 月

Presentations related to other work

Domestic conference

[1] 末藤直樹、斉藤健太、葛西誠也、青野真士、「粘菌アメーバ型最適化問題解探索の 4 足ロボット自律歩行への応用」、電子情報通信学会ソサイエティ大会、金沢、2018 年 9 月

[2] 末藤直樹、斉藤健太、葛西誠也、青野真士、「非同期 CMOS 論理回路に問題をマッピングしたアメーバ型解探索電子システムの動的挙動」、第 66 回応用物理学会春季学術講演会、東京、2019 年 3 月

[3] 大沼柊、斉藤健太、末藤直樹、葛西誠也、青野真士、「粘菌型自律歩行ロボット行動決定のための動態時系列センシング」、第 80 回応用物理学会秋季学術講演会、札幌、2019 年 9 月

[4] 大沼柊、斉藤健太、末藤直樹、葛西誠也、青野真士、「粘菌型自律歩行ロボットの身体感覚による路面状態センシング」、電子情報通信学会総合大会（新型コロナの影響により講演会中止）

[5] 大沼柊、斉藤健太、末藤直樹、葛西誠也、青野真士、「確率的傾斜法を用いた粘菌型自律歩行ロボットの歩行効率化」、第 81 回応用物理学会秋季学術講演会、オンライン、2020 年 9 月

Awards

[1] 第 66 回応用物理学会春季学術講演会 Poster Award