Amoeba-inspired nanoarchitectonic computing implemented using electrical Brownian ratchets
Amoeba-inspired nanoarchitectonic computing implemented using electrical Brownian ratchets

M Aono¹,², S Kasai³, S-J Kim⁴, M Wakabayashi⁵, H Miwa⁶ and M Naruse⁷

¹Earth-Life Science Institute, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo 152-8550, Japan
²PRESTO, Japan Science and Technology Agency, 4-1-8 Honcho, Kawaguchi-shi, Saitama 332-0012, Japan
³Research Center for Integrated Quantum Electronics, Hokkaido University, North 14, West 9, Sapporo 060-0814, Japan
⁴WPI Center for Materials Nanoarchitectonics, National Institute for Materials Science, 1-1 Namiki, Tsukuba, Ibaraki 305-0044, Japan
⁵Department of Biomolecular Engineering, Tokyo Institute of Technology, 4259 Nagatsuta, Midori-ku, Yokohama 226-8501, Japan
⁶Graduate School of Science and Technology, Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo 669-1337, Japan
⁷Photonic Network Research Institute, National Institute of Information and Communications Technology, 4-2-1 Nukui-kita, Koganei, Tokyo 184-8795, Japan

E-mail: masashi.aono@elsi.jp

Received 13 February 2015, revised 13 April 2015
Accepted for publication 21 April 2015
Published 20 May 2015

Abstract
In this study, we extracted the essential spatiotemporal dynamics that allow an amoeboid organism to solve a computationally demanding problem and adapt to its environment, thereby proposing a nature-inspired nanoarchitectonic computing system, which we implemented using a network of nanowire devices called ‘electrical Brownian ratchets (EBRs)’. By utilizing the fluctuations generated from thermal energy in nanowire devices, we used our system to solve the satisfiability problem, which is a highly complex combinatorial problem related to a wide variety of practical applications. We evaluated the dependency of the solution search speed on its exploration parameter, which characterizes the fluctuation intensity of EBRs, using a simulation model of our system called ‘AmoebaSAT-Brownian’. We found that AmoebaSAT-Brownian enhanced the solution searching speed dramatically when we imposed some constraints on the fluctuations in its time series and it outperformed a well-known stochastic local search method. These results suggest a new computing paradigm, which may allow high-speed problem solving to be implemented by interacting nanoscale devices with low power consumption.

Keywords: electrical Brownian ratchet, nanowire, satisfiability problem, natural computing

(Some figures may appear in colour only in the online journal)

1. Introduction
The twenty-first century has witnessed the rapid development of a new academic discipline known as ‘natural computing’, which aims to develop nature-inspired nanoarchitectonic computers that differ from traditional von Neumann architecture computers by utilizing the physical properties and dynamics of interacting nanoscale devices rather than conventional silicon devices [1]. In fact, rapid progress in the miniaturization of semiconductor transistors has facilitated the imminent arrival of an era where the fluctuations in individual electrons or molecules will be disruptive factors that can no longer be ignored when considering the operation of device elements [2]. The arrival of this new era demands the
establishment of new computing paradigms [3] where rather than attempting to mitigate the impact of fluctuations, we must learn to coexist with them by taking inspiration from natural biological systems, which perform information processing in an energy-efficient and robust manner [4, 5]. Two essential components are required to implement nature-inspired nanoarchitectonic computing in terms of new design principles for the layers of systems and their components. Thus, it is necessary to develop biologically-inspired devices and algorithms simultaneously, where the former are the key components of energy-efficient and robust systems, while the latter determine the assembly of a functioning system from its components to facilitate effective computing.

A molecular motor that converts chemical energy into mechanical motion [6] would be a useful design principle in the device layer because it could produce a coherent force efficiently from low input energy, i.e., as small as thermal energy [6, 7]. A possible mechanism for this molecular motor is a Brownian ratchet [7, 8], which rectifies a particle’s Brownian motion (random motion) using a periodic but asymmetric potential, as shown in figure 1(a) [9–15], where the second law of thermodynamics is not violated if the system is not adiabatic. Previous studies have proposed electrical implementations of Brownian ratchets that can generate direct current from fluctuating electron motions [16–21]. Kasai et al first demonstrated the operation of an electrical Brownian ratchet (EBR) at room temperature using a GaAs-based nanowire device with appropriately designed multiple asymmetric gates, as shown in figure 1(b) [22]. Each of the asymmetric gates implements an asymmetric potential that works as a ratchet, thereby skewing the distribution of the carrier motion in one direction.

In the system layer, a plasmodial slime mold Physarum polycephalum, i.e., a single-celled amoeboid organism, as shown in figure 2(a), provides an intriguing research subject for investigating the competitiveness of the computing principles of biological systems. Despite its lack of a central nervous system, this organism exhibits sophisticated abilities to perform decentralized information processing via its spatiotemporal oscillatory dynamics [23, 24]. Aono et al developed an ‘amoeba-based computer’ [25], which uses dynamical feedback control based on inhibitory stimuli in
response to the deformation of the organism, as shown in figure 2(b). Aono et al fabricated a barrier structure by etching multiple lanes into a plastic chip, which allowed the organism to elongate its pseudopod-like branches into various lanes. If an optical stimulus is not applied to the organism, it will attempt to expand to produce a pattern of concentric rings, thereby maximizing the absorption of nutrients from the agar plate by maximizing its body area. Thus, the organism will aim to allocate its intracellular resource (protoplasm) to all of the branches so they grow (elongate) within a chip. However, the organism exhibits a photoavoidance response (negative phototaxis), where illuminating a branch with an optical stimulus will inhibit growth and the supplied resource will bounce back. Therefore, a feedback system can monitor the shape of the organism and use it to select an illumination pattern (the on/off status of illumination on all the lanes) according to preset rules. Thus, the illumination pattern changes as the organism deforms. In this dynamic environment, the organism tries to deform into its optimal shape by maximizing the body area to ensure the maximum nutrient absorption while minimizing the risk of being exposed to aversive optical stimuli. Using this system, it was demonstrated that if the rules for updating the illumination patterns are determined by a modified Hopfield–Tank neural-net model, the system can be used to search for a solution to the travelling salesman problem, which is a well-known combinatorial optimization problem [26, 27].

By extracting the essential features of the computational dynamics of the amoeba-based computer, Aono et al formulated solution-searching algorithms for solving the satisfiability problem, which is one of the fundamental combinatorial problems in computer science [28–31]. The amoeba-inspired algorithms represent the spatiotemporal dynamics of a coupled system with a number of terminal units, each of which is an abstract representation of a branch of the amoeba. To search for a solution to the problem, the unit must behave in a stochastic manner and make an ‘error’; the error indicates that the resource is not supplied even when the inhibitory stimulus is not applied. These algorithms deliver excellent performance even when implemented as software executed on a conventional digital computer [28, 30, 31]. However, by using hardware devices that operate stochastically to implement the algorithms, we can achieve greater efficiency, miniaturization, and energy consumption reductions [29, 32, 33]. The EBR proposed by Kasai et al [22] is one of the most suitable devices for this purpose because it allows the implementation of stochastic operations with errors, consumes low levels of energy, which are comparable to thermal energy, and it facilitates large-scale integration to solve large problems. In this study, we implemented a simulation model of amoeba-inspired nanoarchitectonic computing based on a network of EBRs to solve the satisfiability (SAT) problem. We compared the performance of our model with that of a conventional algorithm, which showed that our model performed better when we imposed some constraints on the occurrence of probabilistic errors.

2. Methods

2.1. Electrical Brownian ratchet

One of the mechanisms of the Brownian ratchet is represented schematically in figure 1(a), which is called a ‘flashing ratchet’. This process occurs in a biological molecular motor, where myosin runs randomly on an actin filament and the ratcheting motion of myosin emerges from the chemical interaction between them [34, 35]. In the case of our EBR, the ratchet is formed electrically with an asymmetric electrostatic potential. The electrons with Brownian motion accumulate in the potential valleys, as shown at the top of figure 1(a). When the potential hill disappears, the electrons move forward and backward equally due to thermal diffusion. The ratchet potential then forms again and some of the diffusing electrons move to the next valley on the left whereas the others remain trapped in the same valley. A directed electron flow is generated by repeating this process. As shown in figure 1(b), the directed current generation was demonstrated experimentally using a GaAs nanowire with multiple asymmetric gates, where the potential was controlled by an external periodic signal [22]. In addition, a current was also generated by flashing with noise in this device [36].

2.2. The satisfiability problem

The SAT problem is stated as follows. Given a logical formula \( f \) involving \( N \) variables \( x_i \), does an assignment \( x_i \in \{0, 1\} \) (i.e., a combination of \( N \) true/false values) exist that satisfies \( f \), which ensures that the overall formula \( f \) is true? For example, a problem instance \( f = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor x_3 \lor \neg x_4) \land (x_2 \lor x_3 \lor \neg x_4) \land (\neg x_3 \lor x_4 \lor \neg x_1) \land (x_3 \lor x_4 \lor \neg x_1) \land (\neg x_4 \lor x_1 \lor \neg x_2) \land (x_4 \lor x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_3) = 1, 1, 1, 1 \). In general, SAT instances involve logical formulae that have unique satisfying solutions (assignments) or that have no satisfying solution.

As the value of \( N \) increases, the total number of possible assignments grows exponentially as \( 2^N \) and no polynomial-time algorithm is known for finding a solution. SAT belongs to the particularly difficult class of problems known as non-deterministic polynomial time (NP) problems. Moreover, SAT was the first problem that was shown to be NP-complete, i.e., any NP problem may be reduced to SAT in polynomial time [37]. Thus, rapid methods (based on both software and hardware) that are capable of solving SAT may potentially be developed to create general-purpose solution-searching engines, which are applied to an extremely large number of practical applications. These applications span a wide range of fields, including model checking, planning, scheduling, and designing [38]; these fields are closely related to domains such as software/hardware verification for designing computer processors and network protocols, planning for controlling mobile explorers, and reasoning for making optimal decision in electronic commerce.
2.2.1. Amoeba-inspired computing model. Aono et al. formulated several versions of AmoebaSAT algorithms to solve the N-variable SAT problem [28–31], where each utilizes the spatiotemporal dynamics of a coupled system of 2N units that correspond to the branches of an amoeba. Each unit is assigned a variable name \( i \in \{1, 2, \ldots, N\} \), a false/true value \( \nu \in \{0, 1\} \), and it is associated with three variables: \( X_{i,\nu}, Y_{i,\nu}, Z_{i,\nu} \). At a discrete time step \( t \), if a resource is supplied to unit \( (i, \nu) \) (corresponding to the amoeba branch), we denote it by \( Y_{i,\nu}(t) = 1 \), and we interpret this as meaning that the system is considering the assignment \( x_i = \nu \). If no resource is supplied, we write this as \( Y_{i,\nu}(t) = 0 \). We use \( L_{i,\nu}(t) = 1 \) or \( L_{i,\nu}(t) = 0 \) to indicate the application or non-application, respectively, of a stimulus that ‘bounces back’ the supply of resources to \( Y_{i,\nu} \) (corresponding to an optical stimulus inhibiting the elongation of the amoeba branch). In one of the AmoebaSAT algorithms called ‘AmoebaSAT-Brownian’, the state transitions of the values \( Y_{i,\nu} \in \{0, 1\} \) are determined by the dynamics based on the following fluctuations \( Z_{i,\nu} \in [0.0, 1.0] \), which simulate the random motions of the EBRs:

\[
Y_{i,\nu}(t) = \begin{cases} 0 & \text{(if } L_{i,\nu}(t) = 1\text{),} \\ \text{sgn}(1 - \epsilon - Z_{i,\nu}(t)) & \text{(if } L_{i,\nu}(t) = 0\text{),} \\ \end{cases}
\]

where \( \epsilon \) is an exploration parameter and

\[
\text{sgn}(z) = \begin{cases} 1 & \text{(if } z > 0\text{),} \\ 0 & \text{(otherwise),} \\ \end{cases}
\]

where \( Z_{i,\nu}(t) \) represents the time series generated from a random number generator (RNG). We use independent 2N RNGs that select real numbers randomly from the interval \([0.0, 1.0]\) to produce uncorrelated white noise. Briefly, if the bounceback stimulus is applied \( (L_{i,\nu}(t) = 1) \), the resource is not supplied \( (Y_{i,\nu}(t) = 0) \). If the bounceback stimulus is not applied \( (L_{i,\nu}(t) = 0) \), the resource is allocated \( (Y_{i,\nu}(t) = 1) \), unless the value of \( Z_{i,\nu}(t) \) exceeds \( 1 - \epsilon \), in which case the supply of the resource is blocked \( (Y_{i,\nu}(t) = 0) \). The exploration parameter \( \epsilon \) increases the probability of the occurrence of ‘error’ where the resource is not supplied despite the absence of the bounceback stimulus. These operations in the state transitions of \( Y_{i,\nu}(t) \) are implemented by the EBR [22] by adding a DC voltage source in parallel with the flashing signal source, as shown in Figure 3. The two sources are switched depending on the status of the bounceback stimulus, i.e., in the absence of the stimulus, the current in the EBR is directed as usual, whereas in the presence of the stimulus, the current is reversed by keeping the ratchet potential turned off. In addition, the error probability \( \epsilon \) can be adjusted by changing the width of the nanowire to control the number of electrons during charge transfer.

We also define a variable \( X_{i,\nu}(t) \in \{-1, 0, 1\} \) to represent the accumulated value of the resource supply

\[
X_{i,\nu}(t + 1) = \begin{cases} X_{i,\nu}(t) + 1 & \text{(if } Y_{i,\nu}(t + 1) = 1 \text{ and } X_{i,\nu}(t) < 1\text{),} \\ X_{i,\nu}(t) - 1 & \text{(if } Y_{i,\nu}(t + 1) = 0 \text{ and } X_{i,\nu}(t) > -1\text{),} \\ X_{i,\nu}(t) & \text{(otherwise).} \\ \end{cases}
\]

The quantity \( X_{i,\nu} \) may be understood as an abstract representation of the displacement from the equilibrium volume of the amoeba branch with one of the three values \(-1, 0, 1\). In each step, the variables \( X = (X_{1,0}, X_{1,1}, X_{2,0}, X_{2,1}, \ldots, X_{N,0}, X_{N,1}) \) are transformed into the variable assignments \( x = (x_1, x_2, \ldots, x_N) \) according to the following rule:

\[
x_i(t) = \begin{cases} 0 & \text{(if } X_{i,0}(t) = 1 \text{ and } X_{i,1}(t) \leq 0\text{),} \\ 1 & \text{(if } X_{i,1}(t) = 1 \text{ and } X_{i,0}(t) \leq 0\text{),} \\ x_i(t - 1) & \text{(otherwise).} \\ \end{cases}
\]

2.2.2. Bounceback control. Any logical formula that defines a given SAT instance is expressed in set theory by replacing the literals \( x_i \) and \( \neg x_i \) with \( i \) and \( \neg i \), respectively. For example, the formula \( f \) is expressed in the form \( F = \{1, 2, -3, \{2, 3, -4\}, \{2, 3, -4\}, \{3, 4, -1\}, \{3, 4, -1\}, \{4, 1, -2\}, \{4, 1, -2\}, \{1, 2, 3\}\} \), where each ‘clause’ in \( f \) is represented by each element \( C \) in the set \( F \). The SAT where no clause contains more than three literals is known as 3-SAT. It has been proved that 3-SAT is NP-complete [37].

First, we focus on the left-hand clause \( (x_1 \lor x_2 \lor \neg x_3) \) in \( f \). If we have both \( x_1 = 0 \) and \( x_2 = 0 \), then we require \( x_3 = 0 \) in order for this clause to be true; otherwise (i.e., \( x_3 = 1 \)), we find that \( (x_1 = 0 \lor x_2 = 0 \lor \neg x_3 = 0) = 0 \). Thus, if at step \( t \) we have both \( X_{1,0}(t) = 1 \) and \( X_{2,0}(t) = 1 \), then at step \( t + 1 \), we apply a bounceback stimulus to \( Y_{3,1} \) (i.e., we select \( L_{3,1}(t + 1) = 1 \)). We refer to this rule as a ‘bounceback rule’. Similarly, from the left-hand clause, we
can read off the bounceback rules $X_{1,0}(t) = 1 \land X_{3,1}(t) = 1 \Rightarrow L_{2,0}(t+1) = 1$ and $X_{2,0}(t) = 1 \land X_{4,1}(t) = 1 \Rightarrow L_{1,0}(t+1) = 1$. We proceed in a similar manner to investigate all of the clauses in $f$ to analyze the mutual interdependencies between the variables and to determine a set of all bounceback rules, which are formally defined as follows [28-31]:

$$L_{i,v}(t+1) = \begin{cases} 1 & \text{(if } B \ni (P, Q) \text{ such that } Q \ni (i, v) \\ \text{and } \forall (j, u) \in P(X_{j,u}(t) = 1) \],} \\ 0 & \text{(otherwise),} \end{cases}$$

(5)

where

$$B = \text{INTRA} \cup \text{INTER} \cup \text{CONTRA}$$

represents the set of all bounceback rules, where each is a pair $(P, Q)$, which denotes that ‘if $P$ is true, then $Q$ is forbidden’.

The subset of $B$ called INTRA is defined as follows to reflect the fact that each variable $x_i$ is forbidden from taking the values 0 and 1 simultaneously. For all $i \in I$, we have

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

(7)

The set INTER is defined as follows to express interference between the variables in each clause. For each variable $i$ contained in each element $C \in F$, we have

$$\text{INTER} \ni \begin{cases} (P, \{(i, 0)\}) & \text{if } C \ni i, \\ (P, \{(i, 1)\}) & \text{if } C \ni \neg i, \end{cases}$$

(8)

where $P$ includes the following elements for all $j \neq i$:

$$P \ni \begin{cases} (j, 0) & \text{if } C \ni j, \\ (j, 1) & \text{if } C \ni \neg j. \end{cases}$$

(9)

If $f$ contains both clauses involving $x_i$ and clauses involving $\neg x_i$, then some rules in the set INTER will prohibit $x_i$ from being assigned either of the values 0 or 1. For example, the two leftmost clauses in $f$, $(x_1 \lor x_2 \lor \neg x_3)$ and $(\neg x_2 \lor x_3 \lor \neg x_4)$, generate the rules $X_{1,0}(t) = 1 \land X_{3,1}(t) = 1 \Rightarrow L_{2,0}(t+1) = 1$ and $X_{2,0}(t) = 1 \land X_{4,1}(t) = 1 \Rightarrow L_{1,0}(t+1) = 1$, respectively. If we were to operate the system under these rules, a state where $X_{1,0}(t) = X_{3,1}(t) = X_{2,0}(t) = X_{4,1}(t) = 1$ would lead to the application of both bounceback stimuli $L_{2,0}(t+1) = L_{1,0}(t+1) = 1$, thereby yielding an inconsistent state where the variable $x_3$ is prevented from being assigned either 0 or 1. To avoid these inconsistencies, for each $i \in I$, we check the set INTER to generate two sets $P_0 = \{P \mid (P, \{(i, 0)\}) \in \text{INTER}\}$ and $P_1 = \{P \mid (P, \{(i, 1)\}) \in \text{INTER}\}$, and define the set CONTRA by joining each elements $(P_0, P_1)$ in a product set of $P_0$ and $P_1$ as follows:

$$\forall (P_0, P_1) \in P_0 \times P_1 \left( \text{CONTRA} \ni (P_0 \cup P_1, P_0 \cup P_1) \right).$$

(10)

Each element $(P, Q) \in \text{CONTRA}$ (i.e., $P = Q$) gives a ‘multi-targetted’ bounceback rule such that the target set $Q$ contains $q = \#(Q)$ elements, where $\#$ is the number of elements in a finite set; the $q$-targetted bounceback rule can be represented equivalently by $q$ ‘single-targetted’ bounceback rules, each of which is given as $(P, \{(i, v)\})$, where $(i, v) \in Q$.

The sets INTRA, INTER and CONTRA for the instance $f$ are tabulated in tables 1-3, respectively. For any problem instance involving $N$ variables and $M$ clauses, where all clauses have three literals (3-SAT), we have $\#(\text{INTRA}) = 2N$, $\#(\text{INTER}) = 3M$ and $\#(\text{CONTRA}) < M^2$. Consequently, assuming that $M$ typically grows as a linear function of $N$, the time required to generate all the bounceback rules, the memory required to store them, and the time required to implement controls at each step based on these rules will all grow polynomially similar to $\text{poly}(N^2)$.

Table 1. Bounceback rules INTRA of $f$.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{(1, 0)}</td>
<td>{(1, 1)}</td>
</tr>
<tr>
<td>{(1, 1)}</td>
<td>{(1, 0)}</td>
</tr>
<tr>
<td>{(2, 0)}</td>
<td>{(2, 1)}</td>
</tr>
<tr>
<td>{(2, 1)}</td>
<td>{(2, 0)}</td>
</tr>
<tr>
<td>{(3, 0)}</td>
<td>{(3, 1)}</td>
</tr>
<tr>
<td>{(3, 1)}</td>
<td>{(3, 0)}</td>
</tr>
<tr>
<td>{(4, 0)}</td>
<td>{(4, 1)}</td>
</tr>
<tr>
<td>{(4, 1)}</td>
<td>{(4, 0)}</td>
</tr>
</tbody>
</table>

Table 2. Bounceback rules INTER of $f$.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{(2, 0), (3, 1)}</td>
<td>{(1, 0)}</td>
</tr>
<tr>
<td>{(1, 0), (3, 1)}</td>
<td>{(2, 0)}</td>
</tr>
<tr>
<td>{(1, 0), (2, 0)}</td>
<td>{(3, 1)}</td>
</tr>
<tr>
<td>{(3, 0), (4, 1)}</td>
<td>{(2, 1)}</td>
</tr>
<tr>
<td>{(2, 1), (4, 1)}</td>
<td>{(3, 0)}</td>
</tr>
<tr>
<td>{(2, 0), (3, 0)}</td>
<td>{(4, 1)}</td>
</tr>
<tr>
<td>{(4, 0), (1, 1)}</td>
<td>{(3, 1)}</td>
</tr>
<tr>
<td>{(3, 1), (1, 1)}</td>
<td>{(4, 0)}</td>
</tr>
<tr>
<td>{(4, 0), (3, 1)}</td>
<td>{(1, 1)}</td>
</tr>
<tr>
<td>{(4, 0), (1, 1)}</td>
<td>{(3, 0)}</td>
</tr>
<tr>
<td>{(3, 0), (4, 0)}</td>
<td>{(1, 1)}</td>
</tr>
<tr>
<td>{(1, 0), (2, 1)}</td>
<td>{(4, 1)}</td>
</tr>
<tr>
<td>{(4, 1), (2, 1)}</td>
<td>{(1, 0)}</td>
</tr>
<tr>
<td>{(4, 1), (1, 0)}</td>
<td>{(2, 1)}</td>
</tr>
<tr>
<td>{(1, 0), (2, 1)}</td>
<td>{(4, 0)}</td>
</tr>
<tr>
<td>{(4, 0), (2, 1)}</td>
<td>{(1, 0)}</td>
</tr>
<tr>
<td>{(4, 0), (1, 0)}</td>
<td>{(2, 1)}</td>
</tr>
<tr>
<td>{(2, 0), (3, 1)}</td>
<td>{(1, 1)}</td>
</tr>
<tr>
<td>{(1, 1), (3, 1)}</td>
<td>{(2, 0)}</td>
</tr>
<tr>
<td>{(2, 0), (3, 0)}</td>
<td>{(1, 0)}</td>
</tr>
<tr>
<td>{(1, 0), (3, 0)}</td>
<td>{(2, 0)}</td>
</tr>
<tr>
<td>{(1, 0), (2, 0)}</td>
<td>{(3, 0)}</td>
</tr>
</tbody>
</table>
Under the bounceback rules defined above, if a system state \( X = (X_{1,0}, X_{1,1}, X_{2,0}, X_{2,1}, \ldots, X_{N,0}, X_{N,1}) \), for all \((i, v)\), either satisfies the condition that \( X_{i,v}(t + 1) = 1 \Leftrightarrow L_{i,v}(t) = 0 \) or the condition that \( X_{i,v}(t) \leq 0 \Leftrightarrow L_{i,v}(t) = 1 \), then the system is ‘stable’. If these stability criteria are not satisfied, there is a high probability that the sign of \( X_{i,v}(t + 1) \) will differ from that of \( X_{i,v}(t) \) depending on \( L_{i,v}(t) \), and thus the system state \( X \) is unstable. Miwa et al. proved mathematically that the condition that AmoebaSAT-Brownian ‘stabilises’ in a state satisfying the above stability criterion is equivalent to the condition that the state represents a ‘solution’ to the SAT instance [39].
variables to random true or false values. Next, the algorithm selects one clause at random from among the clauses that are not satisfied (i.e., that are false) by the variable assignments at a given time and it then selects a single variable at random from within that clause to flip (changing 0–1, or 1–0). The algorithm then iterates this basic behaviour. As shown in figure 4, WalkSAT found the solution after an average of 30913.8 steps. In the parameter region of $\epsilon \in [0.03, 0.2]$, AmoebaSAT-Brownian was faster than WalkSAT.

There are several different versions of WalkSAT, one of which flips the optimal variable to minimize the deterioration in terms of the number of unsatisfied clauses, while it explores other options by flipping a randomly selected variable in a randomly selected unsatisfied clause (i.e., not the optimal variable) with a certain probability determined by a parameter [45]. We confirmed that some versions of the AmoebaSAT algorithms perform better than these parameterized versions of WalkSAT for larger problems with more than 10 000 variables. These results will be reported elsewhere.

3.3. Performance enhancement by constrained fluctuation

Understanding the origins of the high performance obtained by AmoebaSAT algorithms is a current focus of our investigations. WalkSAT algorithms only update one variable in each step, whereas AmoebaSAT algorithms incorporate many processes that collectively update multiple variables and evolve simultaneously, while they interfere with each other through the bounceback control mechanism. This unique ‘concurrent search’ feature of AmoebaSAT algorithms might be the source of their high performance. Previous analytical results suggest that the statistical properties of the fluctuations used in AmoebaSAT algorithms significantly affect their performance [30].

To demonstrate the influence of the statistical properties of the fluctuations on the performance of AmoebaSAT-Brownian, we modified the fluctuation generator $Z_{iv}(t)$ (uncorrelated random noise) by imposing two constraints on the time series of the occurrence of errors: (i) an error could not occur twice consecutively, i.e., the time series could not contain $(Y_{iv}(t), Y_{iv}(t+1)) = (0, 0)$ while $(L_{iv}(t), L_{iv}(t+1)) = (0, 0)$; and (ii) the time interval between error events should be more than one, i.e., the time series could not contain $(Y_{iv}(t), Y_{iv}(t+1), Y_{iv}(t+2)) = (0, 1, 0)$ while $(L_{iv}(t), L_{iv}(t+1), L_{iv}(t+2)) = (0, 0, 0)$. This constrained version called ‘AmoebaSAT-ConstrainedBrownian’ enhanced the performance dramatically, as shown in figure 4. By setting the parameter $\epsilon$ at the optimal value 0.26, AmoebaSAT-ConstrainedBrownian found the solution after an average of 1759.13 steps, thereby performing significantly better than both AmoebaSAT-Brownian and WalkSAT.

4. Discussion

Figure 3 shows a star network of EBRs, which gives equal resistance to all branches because the lengths of all the nanowires are configured uniformly. However, a star network is not always necessary and other network topologies such as bus networks are suitable in cases where the resistances of the interconnecting wires are negligible. In general, implementations of heuristic models such as neural networks or similar systems require a huge number of interconnected wires, which is a problem that has a deleterious impact on the scalability of such systems. Our amoeba-inspired architecture can avoid this problem. Indeed, the number of interconnections between units required for bounceback control grows only linearly as a function of the problem size $N$ when we introduce two clock frequencies for the operations at each step $t$, i.e., a higher frequency for asynchronously updating each status of the stimuli $L_{iv}$ by checking whether the current system state matches with each of the bounceback rules, and a lower frequency for synchronously updating the variables $X_{iv}$, $Y_{iv}$ and $Z_{iv}$ for all of the units $(i, v)$. Moreover, as mentioned above, the computational resources, including time and memory, required for operating bounceback control do not grow excessively rapidly with $N$, because their growth is limited to order $N^2$.

The EBR network is expected to consume low levels of energy because it uses thermal fluctuations to generate the currents, whereas conventional electronic systems consume considerable amounts of power to eliminate various fluctuations, such as noise and variation. The effective power consumption would be reduced further if we could make use of the noise in the environment to flash the electrostatic potential.

AmoebaSAT-ConstrainedBrownian can also be implemented using the EBR network by imposing constraints (i) and (ii) (i.e., the interval between the occurrences of errors will be constrained to more than one step) using the charge transfer process in multiple ratcheting due to the Coulomb interaction between the charge packets in the adjacent potential valleys. This suggests that the high performance level of AmoebaSAT-ConstrainedBrownian will be boosted further at the nanoscale due to the very low energy consumption when implemented using an EBR network.

It would be possible to compare the ‘concurrent search dynamics’ of AmoebaSAT algorithms in which a number of units (i.e., counterparts of pseudopod-like branches of the amoeba) interact with each other via the bounceback control and explore a satisfying solution (i.e., energy minimum state) with some physical processes that can be examined by laboratory experiments. For example, one may intuitively find some analogies in the dynamics of conducting metal balls driven by electric forces to move within a medium of very low electrical conductivity onto which charges are sprayed from outside [46]; the metal balls repulse and attract with each other and eventually form a stable aggregate, which corresponds to a low energy state with a fractal shape.

5. Conclusion

In this study, we proposed an amoeba-inspired nanooarchitectonic computing system to search for solutions to the SAT
problem, which can be applied to solving a wide variety of practical problems. We demonstrated that the performance of our system was significantly faster than that of a conventional fastest stochastic local search method for a randomly generated 3-SAT instance. We also demonstrated the possibility of implementing our architecture using a network of EBRs, which can be operated at the nanoscale with significantly low energy consumption. Thus, we expect that this new bio-inspired computing paradigm will obtain even greater efficiency, miniaturization and reductions in energy consumption.

Acknowledgments

This research was supported partly by the Core-to-Core Program of the Japan Society for the Promotion of Science.

References


Acknowledgments

This research was supported partly by the Core-to-Core Program of the Japan Society for the Promotion of Science.

References


Acknowledgments

This research was supported partly by the Core-to-Core Program of the Japan Society for the Promotion of Science.

References


Acknowledgments

This research was supported partly by the Core-to-Core Program of the Japan Society for the Promotion of Science.

References


Acknowledgments

This research was supported partly by the Core-to-Core Program of the Japan Society for the Promotion of Science.

References