Solving SAT and Hamiltonian Cycle Problem Using Asynchronous P Systems

Hirofumi TAGAWA[†], Nonmember and Akihiro FUJIWARA^{††a)}, Member

SUMMARY In the present paper, we consider fully asynchronous parallelism in membrane computing, and propose two asynchronous P systems for the satisfiability (SAT) and Hamiltonian cycle problem. We first propose an asynchronous P system that solves SAT with n variables and m clauses, and show that the proposed P system computes SAT in $O(mn2^n)$ sequential steps or $O(mn)$ parallel steps using $O(mn)$ kinds of objects. We next propose an asynchronous P system that solves the Hamiltonian cycle problem with n nodes, and show that the proposed P system computes the problem in $O(n!)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

key words: membrane computing, asynchronous parallelism, SAT, Hamiltonian cycle problem

1. Introduction

Membrane computing, which is a representative example of natural computing, is a computational model inspired by the structures and behaviors of living cells. In the initial study on membrane computing, a basic feature of the membrane computing was introduced by Păun et al. [1] as a P system. Each P system consists of hierarchically embedded cell membranes, and each membrane may contain objects. Each object evolves according to evolution rules associated with a membrane in which the object is contained.

The P system and most variants are proved to be universal [2], and several P systems have been proposed for solving NP-complete decision problems [3]–[8] and numerical NP problems [9]–[12] in a polynomial number of steps. For example, Pan and Alhazov [5] have proposed two P systems that solve satisfiability problem (SAT) and Hamiltonian path problem (HPP). The first P system solves SAT with n variables and m clauses in $O(n + m)$ steps using $O(mn^2)$ kinds of objects and $O(2^n)$ membranes, and the second P system solves HPP with n vertices in $O(n^2)$ steps using $O(n^3)$ kinds of objects and $O(n!)$ membranes.

However, synchronous application of evolution rules is assumed on the above P systems with the maximal parallelism, which is a main feature of the P systems. The maximal parallelism means that all applicable rules in all membranes are applied synchronously.

On the other hand, there is obvious asynchronous par-

allelism in the cell biochemistry. The asynchronous parallelism means that all objects may react on rules with different speed, and evolution rules are applied to objects independently. Since all objects in a living cell basically works in asynchronous manner, the asynchronous parallelism must be considered to make P system more realistic model.

For considering the asynchronous parallelism, a number of P systems have been proposed. For example, some sequential P systems [13], [14], which assume sequential application of applicable evolution rules, have been proposed. In the papers, the computational powers of the sequential P systems are considered, and the P systems has been proved to be universal. For another example, P systems with minimal parallelism are considered in [15]. The P system is also universal, and a P system has been proposed for solving SAT in $O(n)$ steps. However, on the P system with minimal parallelism, at least one rule is applied in each membrane if there exists a rule that can be applied for the membrane. In other words, applicable rules in another membranes are applied synchronously, and some kind of synchronicity is assumed in the model.

In the present paper, we consider fully asynchronous parallelism such that any number of applicable rules may be applied in one step on the asynchronous P system. Since there is no restrictive assumption for application of rules, both of sequential and fully parallel executions are allowed on the asynchronous P system. Since the sequential P system with non-deterministically chosen rules is proved to be universal [13], the asynchronous P system is also universal.

As complexity of the asynchronous P system, we consider two kinds of numbers, which are a number of sequential steps and a number of parallel steps. The numbers of sequential steps is a number of executed steps in case that rules are applied sequentially, and the number of parallel steps is a number of executed steps with maximal parallelism.

We propose two asynchronous P systems for two NP problems in the paper. We first show an asynchronous P system for SAT, which is given in the conjunctive normal form with n variables and m clauses. We show that the proposed P system solves SAT in $O(mn2^n)$ sequential steps or $O(mn)$ parallel steps using $O(mn)$ kinds of objects. We next propose an asynchronous P system that computes the Hamiltonian cycle problem. We show that the proposed P system solves the Hamiltonian cycle problem with n nodes in $O(n!)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

Although the number of parallel steps of the proposed

Manuscript received April 11, 2011.

Manuscript revised June 28, 2011.

[†]The author is with NTT Data Corporation, Tokyo, 135-8671 Japan.

^{††}The author is with Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.

a) E-mail: fujiwara@cse.kyutech.ac.jp

DOI: 10.1587/transinf.E95.D.746

P system for SAT is not smaller than the other P systems for SAT [5], [15], the proposed P system solves SAT under any kinds of assumptions for synchronicity. In other words, the proposed P systems work on all of synchronous, sequential and fully asynchronous P systems.

The remainder of the present paper is organized as follows. In Sect. 2, we give a brief description of the model for asynchronous membrane computing. In Sect. 3, we propose the P system for SAT, and propose the P system for the Hamiltonian cycle problem in Sect. 4. Section 5 concludes the paper.

2. Preliminaries

2.1 Computational Model for Membrane Computing

Several models have been proposed for membrane computing [16]. We briefly introduce a basic model of the P system in this subsection.

The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. In other words, the membranes form nested structures. In the present paper, each membrane is denoted using a pair of square brackets, and the number on the right-hand side of each right-hand bracket denotes the label of the corresponding membrane.

An object in the P system is a memory cell, in which each data is stored, and can divide, dissolve, and pass through membranes. In the present paper, each object is denoted by finite strings over a given alphabet, and is contained in one of the membranes.

For example, $[[a]_2 [b]_3]_1$ and Fig. 1 denote the same membrane structure that consists of three membranes. The membrane labeled 1 contains two membranes labeled 2 and 3, and the two membranes contain objects a and b , respectively.

Computation of P systems is executed by evolution rules, which are defined as rewriting rules for membranes and objects. All objects and membranes are transformed in parallel according to applicable evolution rules. If no evolution rule is applicable for objects, the system ceases computation.

Now, we formally define a P system Π and the sets used in the system as follows.

$$\Pi = (O, H, \mu, \omega_1, \omega_2, \dots, \omega_m, R_1, R_2, \dots, R_m)$$

Fig. 1 An example of membrane structure.

- O : O is the set of all objects used in the system.
- H : H is a set of labels for membranes. In the present paper, we always use consecutive integer labels such that $H = \{1, 2, \dots, m\}$. (We assume that a membrane labeled 1, which is called the *skin* membrane, is the outermost membrane, i.e., the skin membrane contains all of the other membranes.)
- μ : μ is membrane structure that consists of m membranes. Each membrane in the structure is labeled with an element in H .
- $\omega_1, \omega_2, \dots, \omega_m$: Each ω_i is a set of objects initially contained only in the membrane labeled i . (For example, $\omega_1 = \phi$, $\omega_2 = \{a\}$, and $\omega_3 = \{b\}$ in Fig. 1.)
- $R_1, R_2, \dots, R_m$: Each R_i is a set of evolution rules that are applicable to objects in the membrane labeled i .

In the present paper, we assume that input objects are given from the outside region into the skin membrane, and computation is started by applying evolution rules. We also assume that output objects are sent out from the skin membrane to the outside region.

In membrane computing, several types of rules are proposed. In the present paper, we consider five basic rules of the following forms.

- (1) **Object evolution rule:** $[\alpha]_h \rightarrow [\beta]_h$
where $h \in H$, and $\alpha, \beta \in O$. Using the rule, an object α evolves into another object, β . (We omit the brackets in each evolution rule for cases that a corresponding membrane is obvious.)
- (2) **Send-in communication rule:** $\alpha []_h \rightarrow [\beta]_h$
where $h \in H$, and $\alpha, \beta \in O$. Using the rule, an object α is sent into the membrane, and can evolve into another object β .
- (3) **Send-out communication rule:** $[\alpha]_h \rightarrow []_h \beta$
where $h \in H$, and $\alpha, \beta \in O$. Using the rule, an object α is sent out of the membrane, and can evolve into another object β .
- (4) **Dissolution rule:** $[\alpha]_h \rightarrow \beta$
where $h \in H$, and $\alpha, \beta \in O$. Using the rule, the membrane, which contains object α , is dissolved, and the object can evolve into another object β . (The skin membrane cannot be dissolved.)
- (5) **Division rule:** $[\alpha]_h \rightarrow [\beta]_h [\gamma]_h$
where $h \in H$, and $\alpha, \beta, \gamma \in O$. Using the rule, the membrane, which contains object α , is divided into two membranes that contain objects β and γ , respectively.

We also assume that each of the above rules is applied in a constant number of biological steps and consider the number of steps executed in a P system as complexity of the P system in the following sections.

2.2 Maximal Parallelism and Asynchronicity

In the standard model in membrane computing, which is a P system with maximal parallelism, all of the above rules are applied in a non-deterministic maximally parallel manner.

In one step of the P system, each object is evolved according to one of applicable rules. (In case there are several possibilities, one of the applicable rules is non-deterministically chosen.) All object, for which no rules applicable, remain unchanged to the next step. In other words, all applicable rules are applied in parallel in each step of computation.

On the other hand, we assume that evolution rules are applied in fully asynchronous manner in the present paper. In the asynchronous P system, any number of applicable evolution rules is applied in each step of computation. In other words, the asynchronous P system can be executed sequentially, and also can be executed in maximal parallel manner.

We now show an example for difference between the P system with maximal parallelism and the asynchronous P system. The following Π_1 is a simple P system that computes AND function for any number of Boolean values.

$$\Pi_1 = (O, H, \mu, \omega_1, R_1),$$

- $O = \{\langle 0 \rangle, \langle 1 \rangle, \langle C_0 \rangle, \langle C_1 \rangle\}$
- $H = \{1\}$
- $\mu = []_1$
- $\omega_1 = \{\langle C_0 \rangle\}$
- $R_1 = R_{1,1} \cup R_{1,2}$
 - $R_{1,1} = \{[\langle 0 \rangle \langle C_0 \rangle] \rightarrow [] \langle 0 \rangle, \langle 1 \rangle \rightarrow \langle C_1 \rangle\}$
 - $R_{1,2} = \{\langle C_0 \rangle \langle C_1 \rangle \rightarrow [] \langle 1 \rangle\}$

In the above P system, two objects, $\langle 0 \rangle$ and $\langle 1 \rangle$, denote Boolean values 0 and 1, respectively.

Let us consider an execution of Π_1 on the P system with maximal parallelism. A set of the above objects, which denote input Boolean values, is given from the outside region into the membrane. Figure 2 (a) illustrates an execution in case that there exists object $\langle 0 \rangle$ in the input set. In this case, evolution rule $[\langle 0 \rangle \langle C_0 \rangle] \rightarrow [] \langle 0 \rangle$ is applied to one of objects $\langle 0 \rangle$, and output object $\langle 0 \rangle$ is sent out to the outside region. On the other hand, Fig. 2 (b) illustrates an execution in case

Fig. 2 An example of executions of the P system Π_1 with maximal parallelism: (a) an execution in case that there exists $\langle 0 \rangle$ in the input, (b) an execution in case that all input objects are $\langle 1 \rangle$.

that there exists no $\langle 0 \rangle$ in the input set, i.e. all input objects are $\langle 1 \rangle$. In this case, an evolution rule $\langle 1 \rangle \rightarrow \langle C_1 \rangle$ is applied to objects $\langle 1 \rangle$ in the first step of the execution. Then, an evolution rule $\langle C_0 \rangle \langle C_1 \rangle \rightarrow [] \langle 1 \rangle$ is applied in the second step, and output object $\langle 1 \rangle$ is sent out to the outside region.

In both cases, P system Π_1 computes AND function for any number of Boolean values in a constant number of steps with maximal parallelism.

However, P system Π_1 apparently does not work on any kinds of asynchronous P systems. Let us consider an input set that includes $\langle 0 \rangle$ and $\langle 1 \rangle$. In this case, evolution rules $\langle 1 \rangle \rightarrow \langle C_1 \rangle$ and $\langle C_0 \rangle \langle C_1 \rangle \rightarrow [] \langle 1 \rangle$ can be applied before application of $[\langle 0 \rangle \langle C_0 \rangle] \rightarrow [] \langle 0 \rangle$ on an asynchronous P system, and the P system may output a wrong object $\langle 1 \rangle$.

In fact, computation of AND function for any number of Boolean values is a hard problem on asynchronous P systems. The following Π_2 is a P system that asynchronously computes AND function for n Boolean values for a given n .

$$\Pi_2 = (O, H, \mu, \omega_1, R_1),$$

- $O = \{\langle 0 \rangle, \langle 1 \rangle, \langle C_0 \rangle, \langle C_1 \rangle, \dots, \langle C_n \rangle\}$
- $H = \{1\}$
- $\mu = []_1$
- $\omega_1 = \{\langle C_0 \rangle\}$
- $R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$
 - $R_{1,1} = \{[\langle 0 \rangle \langle C_0 \rangle] \rightarrow [] \langle 0 \rangle, \langle 1 \rangle \rightarrow \langle C_1 \rangle\}$
 - $R_{1,2} = \{\langle C_i \rangle \langle C_j \rangle \rightarrow \langle C_{i+j} \rangle \mid 1 \leq i \leq j \leq n-1\}$
 - $R_{1,3} = \{\langle C_n \rangle \rightarrow [] \langle 1 \rangle\}$

We now describe complexity of asynchronous P systems using the above example. We first introduce complexity of an asynchronous P system in maximal parallel manner. Since P system Π_2 can be obviously executed in one step if $\langle 0 \rangle$ is in the input set, we consider the case that P system Π_2 is executed for an input set that includes no $\langle 0 \rangle$.

Figure 3 (a) illustrates an execution in case that P system Π_2 is executed in maximal parallel manner for the input set. In the execution, object C_n is created in $O(\log n)$ steps

Fig. 3 An example of executions of the P system Π_2 : (a) an execution with maximal parallelism, (b) a sequential execution.

since the number of objects C_i is reduced by half in each step of the execution. We call the number of steps executed on the asynchronous P system in maximal parallel manner *the number of parallel steps*. For example, P system Π_2 computes the AND function in $O(\log n)$ parallel steps.

We next introduce complexity of an asynchronous P system in case that the P system is executed sequentially. Figure 3 (b) illustrates an example of execution in case that P system Π_2 is executed sequentially for the same input set. The computation is executed in $O(n)$ steps if one evolution rule is applied in each step of the execution. Since various sequential executions can be considered on the asynchronous P system, we call the worst number of sequentially executed steps *the number of sequential steps*. For example, P system Π_2 computes the AND function in $O(n)$ sequential steps.

For the asynchronous P system, the numbers of parallel and sequential steps means the best and worst case complexities, respectively. In addition to this, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

3. SAT

3.1 Input and Output

The SAT is a well-known problem that determines if there exists a truth assignment for a given Boolean formula. We assume that an input formula of SAT is given in the conjunctive normal form with n Boolean variables and m clauses. We also assume that an output of SAT is one of two values, “TRUE” or “FALSE”. The output value is “TRUE” if there exists a truth assignment for satisfying the formula, otherwise, the value is “FALSE”.

The following is an example of an input formula with 2 variables and 3 clauses. Since a truth assignment $x_1 = 1, x_2 = 0$ satisfies the formula, an output of SAT is “TRUE”.

$$(x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2) \wedge (\bar{x}_2)$$

The above input is given by the following set of objects O_L in the P system.

$$O_L = \{\langle X_{i,j}, V \rangle \mid 1 \leq i \leq n, 1 \leq j \leq m, V \in \{0, 1, N\}\}$$

Each object $\langle X_{i,j}, V \rangle$ denotes a Boolean value of variable x_i in the j -th clause. $V = 0$ and $V = 1$ denote that $\bar{x}_i$ and x_i are in the j -th clause, respectively. In addition, V is set to N if neither $\bar{x}_i$ nor x_i is in the j -th clause.

For example, the following set of objects denotes the above input formula.

$$O_L = \{\langle X_{1,1}, 1 \rangle, \langle X_{2,1}, 0 \rangle, \langle X_{1,2}, 1 \rangle, \langle X_{2,2}, 1 \rangle, \langle X_{1,3}, N \rangle, \langle X_{2,3}, 0 \rangle\}$$

We assume that input set O_L is given from the outside region into the skin membrane.

The output of the P system is one of two objects, $\langle TRUE \rangle$ and $\langle FALSE \rangle$. The object $\langle TRUE \rangle$ is sent out

from the skin membrane to the outside region if the input formula is satisfiable, otherwise, the object $\langle FALSE \rangle$ is sent out to the outside region.

3.2 Asynchronous P System for SAT

We first explain an overview of the asynchronous P system for computing SAT. The membrane structure used in the computation is $[[]_2]_1$. We call the membranes labeled 1 and 2 *outer* and *inner* membranes, respectively, for the sake of simplicity.

The computation of the P system mainly consists of the following 4 steps.

- Step 1:** Move all input objects in outer membrane into the inner membrane.
- Step 2:** Create all truth assignments by dividing the inner membrane repeatedly.
- Step 3:** Check satisfiability for the truth assignment in each divided membrane, and output result to the outer membrane.
- Step 4:** Send out a final result, “TRUE” or “FALSE” from the outer membrane.

We now explain outline of each step of the computation. In Step 1, all input objects in the outer membrane are moved into the inner membrane. Since the P system used in the paper is asynchronous, we cannot move the input objects in parallel, and input objects are moved one by one applying the following two sets of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,1} = & \{\langle X_{1,1}, V \rangle []_2 \rightarrow [\langle M_{2,1} \rangle \langle X_{1,1}, V \rangle]_2 \mid V \in \{0, 1, N\}\} \\ & \cup \{ \langle M_{i,j} \rangle \langle X_{i,j}, V \rangle []_2 \rightarrow [\langle M_{i+1,j} \rangle \langle X_{i,j}, V \rangle]_2 \\ & \quad \mid 1 \leq i \leq n, 1 \leq j \leq m, V \in \{0, 1, N\} \} \\ & \cup \{ \langle M_{n+1,j} \rangle \rightarrow \langle M_{1,j+1} \rangle \mid 1 \leq j \leq m \} \\ & \cup \{ \langle M_{1,m+1} \rangle []_2 \rightarrow [\langle S_1 \rangle]_2 \} \end{aligned}$$

(Evolution rules for the inner membrane)

$$R_{2,1} = \{ [\langle M_{i,j} \rangle]_2 \rightarrow []_2 \langle M_{i,j} \rangle \mid 2 \leq i \leq n+1, 1 \leq j \leq m \}$$

In the above evolution rules, object $\langle X_{1,1}, V \rangle$ starts the computation, and object $\langle M_{i,j} \rangle$ enables input object $\langle X_{i,j}, V \rangle$ to move into the inner membrane. At the end of Step 1, object $\langle S_1 \rangle$ is created in the inner membrane, and the object triggers computation of Step 2.

In Step 2, all truth assignments are created by dividing the inner membrane repeatedly. The assignment is denoted two objects, $\langle A_i, 0 \rangle$ and $\langle A_i, 1 \rangle$, which means assignments $x_i = 0$ and $x_i = 1$, respectively.

The division of the membrane in Step 2 is executed applying the following set of evolution rules repeatedly.

(Evolution rules for the inner membrane)

$$R_{2,2} = \{ [\langle S_i \rangle]_2 \rightarrow [\langle S_{i+1} \rangle \langle A_i, 0 \rangle]_2 [\langle S_{i+1} \rangle \langle A_i, 1 \rangle]_2 \}$$

$$| 1 \leq i \leq n \}$$

At the end of Step 2, 2^n membranes are created, and object $\langle S_{n+1} \rangle$ triggers computation of Step 3 in each created membrane.

In Step 3, object $\langle C_{i,j} \rangle$ is used to check an assignment to x_i in the j -th clause. The check is executed applying the following set of evolution rules repeatedly.

(Evolution rules for the inner membrane)

$$R_{2,3} = R_{2,3,1} \cup R_{2,3,2} \cup R_{2,3,3} \cup R_{2,3,4} \cup R_{2,3,5}$$

$$R_{2,3,1} = \{ \langle S_{n+1} \rangle \rightarrow \langle C_{1,1} \rangle \}$$

$$R_{2,3,2} = \{ \langle C_{i,j} \rangle \langle X_{i,j}, V \rangle \langle A_i, V \rangle \rightarrow \langle C_{i+1,j} \rangle \langle A_i, V \rangle \mid 1 \leq i \leq n, 1 \leq j \leq m, V \in \{0, 1\} \}$$

$$R_{2,3,3} = \{ \langle C_{i,j} \rangle \langle X_{i,j}, V \rangle \langle A_i, V' \rangle \rightarrow \langle C_{i+1,j} \rangle \langle A_i, V' \rangle \mid 1 \leq i \leq n, 1 \leq j \leq m, V \in \{0, 1, N\}, V' \in \{0, 1\}, V \neq V' \}$$

$$R_{2,3,4} = \{ [\langle C_{1,m+1} \rangle]_2 \rightarrow []_2 \langle TRUE \rangle \}$$

$$R_{2,3,5} = \{ [\langle C_{n+1,j} \rangle]_2 \rightarrow []_2 \langle FALSE, 1 \rangle \mid 1 \leq j \leq m \}$$

The above sub-rules are applied as follows. $R_{2,3,1}$ is an evolution rule that starts Step 3. $R_{2,3,2}$ is a set of evolution rules that are applied in case that the j -th clause is satisfied by an assignment for x_i . For example, in case that the j -th clause is satisfied by $x_i = 1$, two objects, $\langle X_{i,j}, 1 \rangle$ and $\langle A_i, 1 \rangle$, are in the membrane, and then, an evolution rule $\langle C_{i,j} \rangle \langle X_{i,j}, 1 \rangle \langle A_i, 1 \rangle \rightarrow \langle C_{i+1,j} \rangle \langle A_i, 1 \rangle$ is applied. In case that the rule is applied, the next check is executed for variable x_1 in the $j+1$ -st clause. On the other hand, $R_{2,3,3}$ is a set of evolution rules that are applied in case that the j -th clause cannot be satisfied by an assignment for x_i . After application of the rule for x_i , the next check is executed for variable x_{i+1} in the same clause.

If all clauses are checked and satisfied by the assignment, an evolution rule in $R_{2,3,4}$ is applied, and object $\langle TRUE \rangle$ is sent out to the outer membrane. In the other case, i.e. if there is a clause that cannot be satisfied by the assignment, an evolution rule in $R_{2,3,5}$ is applied, and object $\langle FALSE, 1 \rangle$ is sent out to the outer membrane.

In Step 4, a final result is sent out from the outer membrane. If object $\langle TRUE \rangle$ is in the outer membrane, there is a assignment that satisfies the input formula, and the object is sent out to the outside region immediately applying $[\langle TRUE \rangle]_1 \rightarrow []_1 \langle TRUE \rangle$. On the other hand, 2^n objects $\langle FALSE, 1 \rangle$ are sent out from the divided membranes to the outer membrane in case that the formula cannot be satisfied. Since we assume asynchronous P system, the sum of the objects must be counted asynchronously. These computation in Step 4 is executed applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$R_{1,4} = \{ [\langle TRUE \rangle]_1 \rightarrow []_1 \langle TRUE \rangle \}$$

$$\cup \{ \langle FALSE, 2^k \rangle \langle FALSE, 2^k \rangle \rightarrow \langle FALSE, 2^{k+1} \rangle \mid 0 \leq k \leq n-1 \}$$

$$\cup \{ [\langle FALSE, 2^n \rangle]_1 \rightarrow []_1 \langle FALSE \rangle \}$$

We now formally define P system Π_{SAT} that asynchronously computes SAT with n variables and m clauses.

$$\Pi_{SAT} = (O, \mu, \omega_1, \omega_2, R_1, R_2),$$

- $O = O_L \cup \{ \langle S_i \rangle, \langle M_{i,j} \rangle, \langle C_{i,j} \rangle \mid 1 \leq i \leq n+1, 1 \leq j \leq m+1 \} \cup \{ \langle TRUE \rangle, \langle FALSE \rangle \} \cup \{ \langle FALSE, 2^k \rangle \mid 0 \leq k \leq n \}$
- $\mu = []_2 []_1$
- $\omega_1 = \omega_2 = \phi$
- $R_1 = R_{1,1} \cup R_{1,4}, R_2 = R_{2,1} \cup R_{2,2} \cup R_{2,3}$

Figure 4 illustrates an execution of the P system Π_{SAT} . An input formula for the example is $(x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2) \wedge (\bar{x}_2)$, and $n = 2, m = 3$. A set of objects that denote the input formula is given from the outside region into the skin membrane. Then, sets of evolution rules, $R_{1,1} \cup R_{2,1}$, are applied, and all objects are moved into the inner membrane.

Fig. 4 An example of execution of Π_{SAT} .

In the next step, each inner membrane is repeatedly divided so as to create objects denoting all truth assignments. For example, a pair of objects $\langle A_1, 1 \rangle$ and $\langle A_2, 0 \rangle$ denotes a truth assignment $x_1 = 1$ and $x_2 = 0$, respectively.

After creating each truth assignment, satisfiability of the formula for each assignment is checked in each inner membrane applying a set of evolution rule $R_{2,3}$. In the example, the formula is satisfiable with an assignment $x_1 = 1, x_2 = 0$, and an object $\langle TRUE \rangle$ is sent out from the membrane that denotes the assignment.

It is worth while noticing that there are various executions for the above example since the P system is fully asynchronous. In any asynchronous execution, the proposed P system always outputs a correct solution for the input formula from the following reasons.

- A set of evolution rules for each membrane is designed to be applied sequentially.
- Computation in divided membranes can be executed independently.
- Some barrier mechanisms, which synchronize executions in membranes, are implemented in the P system.

3.3 Complexity

We now consider complexity of the proposed P system Π_{SAT} . Both of the numbers of sequential and parallel steps in Step 1 are $O(mn)$ because Step 1 is executed sequentially. The number of sequential steps in Step 2 is $O(2^n)$ because 2^n membranes are created using division. On the other hand, Step 2 can be executed in $O(n)$ parallel steps if the P system is executed in maximal parallel manner.

In Step 3, $O(mn)$ checks are executed in each of $O(2^n)$ membranes. Therefore, the number of sequential and parallel steps are $O(mn2^n)$ and $O(mn)$, respectively. The numbers of sequential and parallel steps in Step 4 are $O(2^n)$ and $O(n)$ in case that object $\langle FALSE \rangle$ is sent out as output.

Since the number of types of objects in the P system is $O(mn)$ and the number of kinds of evolution rules is also $O(mn)$, we obtain the following theorem for Π_{SAT} .

Theorem 1: The asynchronous P system Π_{SAT} , which computes the satisfiability problem with n Boolean variables and m clauses, works in $O(mn2^n)$ sequential steps or $O(mn)$ parallel steps using $O(mn)$ types of objects and evolution rules of size $O(mn)$. $\square$

4. Hamiltonian Cycle Problem

4.1 Input and Output

The Hamiltonian cycle is a typical NP-complete graph problem. Given a directed graph $G = (V, E)$, a cycle in G is called a Hamiltonian cycle if the cycle visits each vertex exactly once. For example, the directed graph in Fig. 5 has a Hamiltonian cycle $(v_1, v_3, v_2, v_4, v_1)$.

In the present paper, an output of the problem is one

Fig. 5 A directed graph containing a Hamiltonian cycle $(v_1, v_3, v_2, v_4, v_1)$.

of two values “TRUE” and “FALSE”. The output value “TRUE” if there exists a Hamiltonian cycle for an input graph, otherwise, the value is “FALSE”.

We assume that an input graph is given by the following two sets of objects, O_V and O_E , in the P system.

$$O_V = \{\langle v_i \rangle \mid 1 \leq i \leq n\}$$

$$O_E = \{\langle e_{i,j}, W \rangle \mid 1 \leq i \leq n, 1 \leq j \leq n, W \in \{T, F\}\}$$

Objects $\langle v_i \rangle$ and $\langle e_{i,j}, W \rangle$ denote a vertex v_i and an edge (v_i, v_j) in the graph G , respectively. If there exists an edge (v_i, v_j) in the graph, W is set to T , i.e. $\langle e_{i,j}, T \rangle$ is in O_E , otherwise, W is set to F . A set of objects $\{\langle e_{i,i}, F \rangle \mid 1 \leq i \leq n\}$ is also contained in O_E for simplicity of the P system.

For example, the following two sets of objects denote a directed graph in Fig. 5.

$$O_V = \{\langle v_1 \rangle, \langle v_2 \rangle, \langle v_3 \rangle, \langle v_4 \rangle\}$$

$$O_E = \{\langle e_{1,1}, F \rangle, \langle e_{1,2}, F \rangle, \langle e_{1,3}, T \rangle, \langle e_{1,4}, F \rangle, \\ \langle e_{2,1}, F \rangle, \langle e_{2,2}, F \rangle, \langle e_{2,3}, F \rangle, \langle e_{2,4}, T \rangle, \\ \langle e_{3,1}, F \rangle, \langle e_{3,2}, T \rangle, \langle e_{3,3}, F \rangle, \langle e_{3,4}, T \rangle, \\ \langle e_{4,1}, T \rangle, \langle e_{4,2}, F \rangle, \langle e_{4,3}, F \rangle, \langle e_{4,4}, F \rangle\}$$

We assume that O_V and O_E are given from the outside region into the skin membrane.

The output of the P system is the same as the P system for SAT. The object $\langle TRUE \rangle$ is sent out from the skin membrane to the outside region if there exists a Hamiltonian cycle in the graph, otherwise, the object $\langle FALSE \rangle$ is sent out to the outside region.

4.2 Asynchronous P System for Hamiltonian Cycle Problem

The P system for the Hamiltonian cycle problem also consists of inner and outer membranes, i.e. membrane structure of the P system is $[[]_2]_1$.

The computation in the asynchronous P system mainly consists of the following 4 steps.

Step 1: Move all input objects in outer membrane into the inner membrane.

Step 2: Create permutations of vertices by dividing the inner membrane repeatedly.

Step 3: Check each permutation of vertices whether the permutation is a Hamiltonian cycle, in each divided membrane, and output result to the outer membrane.

Step 4: Send out a final result, “TRUE” or “FALSE” from the outer membrane.

We now explain an outline of each step of the computation. Step 1 is executed using similar rules to the P system for SAT.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,1} = & \{ \langle e_{1,1}, F \rangle []_2 \rightarrow [\langle M_{2,1} \rangle \langle e_{1,1}, F \rangle]_2 \} \\ & \cup \{ \langle M_{i,j} \rangle \langle e_{i,j}, V \rangle []_2 \rightarrow [\langle M_{i+1,j} \rangle \langle e_{i,j}, V \rangle]_2 \\ & \quad | 1 \leq i \leq n, 1 \leq j \leq n, V \in \{T, F\} \} \\ & \cup \{ \langle M_{i,n+1} \rangle \langle v_i \rangle []_2 \rightarrow [\langle M_{i+1,n+1} \rangle \langle v_i \rangle]_2 \\ & \quad | 1 \leq i \leq n \} \\ & \cup \{ \langle M_{n+1,j} \rangle \rightarrow \langle M_{1,j+1} \rangle | 1 \leq j \leq n+1 \} \\ & \cup \{ \langle M_{1,n+2} \rangle []_2 \rightarrow [\langle C_{1,1} \rangle]_2 \} \end{aligned}$$

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,1} = & \{ [\langle M_{i,j} \rangle]_2 \rightarrow []_2 \langle M_{i,j} \rangle \\ & \quad | 2 \leq i \leq n+1, 1 \leq j \leq n+1 \} \end{aligned}$$

In the above evolution rules, object $\langle e_{1,1}, F \rangle$ starts the computation, and two kinds of input objects are moved into the inner membrane using object $\langle M_{i,j} \rangle$. At the end of Step 1, object $\langle C_{1,1} \rangle$ is created in the inner membrane, and the object triggers computation of Step 2.

In Step 2, permutations of vertices are created by dividing the inner membrane repeatedly. Each permutation starts from vertex v_1 , and v_1 is checked so as not to be in the same permutation again. Then, the membrane is divided for the next vertex of the permutation. The division is executed sequentially for all of the other vertices. (The number of divided membranes is counted in Step 4.) If the membrane is divided for vertex v_k , the vertex v_k is checked so that the division is not executed for the checked vertex v_k again.

The Step 2 is executed applying the following set of evolution rules repeatedly. In the evolution rules, object $\langle \lambda_i \rangle$ denotes that vertex v_i is already checked, and $\langle z_{k,i} \rangle$ denotes that v_i is the k -th vertex in the permutation. (These two objects and $\langle v_i \rangle$ are never in the same membrane.) In addition, $\langle H_1 \rangle$ is an object that triggers computation of Step 3.

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,2} = & \{ \langle C_{1,1} \rangle \langle v_1 \rangle \rightarrow \langle C_{2,1} \rangle \langle \lambda_1 \rangle \langle z_{1,1} \rangle \} \\ & \cup \{ [\langle C_{k,j} \rangle \langle v_j \rangle]_2 \rightarrow [\langle C_{k,j+1} \rangle \langle v_j \rangle]_2 [\langle C_{k+1,1} \rangle \langle \lambda_j \rangle \langle z_{k,j} \rangle]_2 \\ & \quad | 2 \leq k \leq n, 1 \leq j \leq n \} \\ & \cup \{ \langle C_{k,j} \rangle \langle \lambda_j \rangle \rightarrow \langle C_{k,j+1} \rangle \langle \lambda_j \rangle | 2 \leq k \leq n, 1 \leq j \leq n \} \\ & \cup \{ \langle C_{n+1,1} \rangle \rightarrow \langle H_1 \rangle \} \end{aligned}$$

After application of the above evolution rule $R_{2,2}$, $(n-1)!$ membranes, which contain permutations of vertices, are created. The following is an example of the membrane, which denotes a permutation (v_1, v_3, v_2, v_4) . (A set of objects O_E , which denotes edges in the graph, is also contained in the same membrane.)

$$[\langle \lambda_1 \rangle, \langle \lambda_2 \rangle, \langle \lambda_3 \rangle, \langle \lambda_4 \rangle, \langle z_{1,1} \rangle, \langle z_{2,3} \rangle, \langle z_{3,2} \rangle, \langle z_{4,4} \rangle, \langle H_1 \rangle]_2$$

In Step 3, each permutation of vertices is checked whether the permutation is a Hamiltonian cycle, in each divided membrane. The permutation is a Hamiltonian cycle if and only if there exist edges between all pairs of consecutive vertices in the permutation. The edges are checked using the following evolution rules in each membrane.

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,3} = & R_{2,3,1} \cup R_{2,3,2} \cup R_{2,3,3} \\ R_{2,3,1} = & \{ \langle H_k \rangle \langle z_{k,i} \rangle \langle e_{i,j}, T \rangle \langle z_{k+1,j} \rangle \rightarrow \langle H_{k+1} \rangle \langle z_{k+1,j} \rangle \\ & \quad | 1 \leq k \leq n-1, 1 \leq i \leq n, 1 \leq j \leq n \} \\ & \cup \{ \langle H_n \rangle \langle z_{n,i} \rangle \langle e_{i,1}, T \rangle \rightarrow \langle TRUE \rangle \} \\ R_{2,3,2} = & \{ \langle H_k \rangle \langle z_{k,i} \rangle \langle e_{i,j}, F \rangle \langle z_{k+1,j} \rangle \rightarrow \langle FALSE, 1 \rangle \\ & \quad | 1 \leq k \leq n-1, 1 \leq i \leq n, 1 \leq j \leq n \} \\ & \cup \{ \langle H_n \rangle \langle z_{n,i} \rangle \langle e_{i,1}, F \rangle \rightarrow \langle FALSE, 1 \rangle \} \\ R_{2,3,3} = & \{ [\langle TRUE \rangle]_2 \rightarrow []_2 \langle TRUE \rangle \} \\ & \cup \{ [\langle FALSE, 1 \rangle]_2 \rightarrow []_2 \langle FALSE, 1 \rangle \} \end{aligned}$$

In case that the permutation is a Hamiltonian cycle, $R_{2,3,1}$ is repeatedly applied, and object $\langle TRUE \rangle$ is created. On the other hand, $R_{2,3,2}$ is applied if there is no edge between the k -th and the $k+1$ -st vertices in the permutation, and object $\langle FALSE, 1 \rangle$ is created. Then, one of two objects, $\langle TRUE \rangle$ and $\langle FALSE, 1 \rangle$, is sent out from the membrane using $R_{2,3,3}$.

In Step 4, a final result is sent out from the outer membrane. The step is similar to Step 4 of P system for SAT. If object $\langle TRUE \rangle$ is in the outer membrane, the object is sent out to the outside region immediately. On the other hand, $(n-1)!$ objects, $\langle FALSE, 1 \rangle$, are in the outer membrane in case that there is no Hamiltonian cycle. Since $k! = k \times (k-1)!$, the computation in Step 4 is executed applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,4} = & \{ [\langle TRUE \rangle]_1 \rightarrow []_1 \langle TRUE \rangle \} \\ & \cup \{ \langle FALSE, k-1 \rangle^k \rightarrow \langle FALSE, k \rangle \\ & \quad | 2 \leq k \leq n-1 \} \\ & \cup \{ [\langle FALSE, n-1 \rangle]_1 \rightarrow []_1 \langle FALSE \rangle \} \end{aligned}$$

We now summarize the P system Π_{HCP} , which asynchronously computes the Hamiltonian cycle problem with n vertices.

$$\Pi_{HCP} = (O, \mu, \omega_1, \omega_2, R_1, R_2),$$

- $O = O_V \cup O_E \cup \{ \langle M_{i,j} \rangle | 1 \leq i \leq n+1, 1 \leq j \leq n+2 \} \cup \{ \langle C_{i,j} \rangle | 1 \leq i \leq n+1, 1 \leq j \leq n+1 \} \cup \{ \langle z_{k,j} \rangle | 1 \leq k \leq n, 1 \leq j \leq n \} \cup \{ \langle H_k \rangle | 1 \leq k \leq n \} \cup \{ \langle TRUE \rangle, \langle FALSE \rangle \} \cup \{ \langle FALSE, k \rangle | 1 \leq k \leq n-1 \}$
- $\mu = [[]_2]_1$
- $\omega_1 = \omega_2 = \phi$
- $R_1 = R_{1,1} \cup R_{1,4}, R_2 = R_{2,1} \cup R_{2,2} \cup R_{2,3}$

Fig. 6 An example of execution of Π_{HCP} .

Figure 6 illustrates an execution of the P system Π_{HCP} . An input for the example is the directed graph in Fig. 5. Two sets of objects, O_V and O_E , are given from the outside region into the skin membrane. Then, sets of evolution rules, $R_{1,1} \cup R_{2,1}$, are applied, and all objects are moved into the inner membrane.

In the next step, each inner membrane is repeatedly divided so as to create objects denoting all permutations of vertices applying $R_{2,2}$. Note that some redundant inner membranes are created applying $R_{2,2}$. Since no evolution rule can be applied for the redundant membrane, computation in the redundant membrane automatically stops.

In the final step, an object $\langle TRUE \rangle$ is created applying $R_{2,3}$ in the membrane such that the permutation is a Hamiltonian cycle, and then, the object is sent out from the outer membrane applying $R_{1,4}$.

4.3 Complexity

Complexity of the asynchronous P system Π_{HCP} is as follows. Since $O(n^2)$ objects are moved sequentially in Step 1, both of the numbers of sequential and parallel steps in Step 1

are $O(n^2)$. In Step 2, $O((n-1)!)$ membranes are created, and evolution rules are applied sequentially at most $O(n)$ times in each membrane. Therefore, the number of sequential and parallel steps of Step 2 are $O(n!)$ and $O(n^2)$, respectively.

In Step 3, each permutation, whose length is n , is checked sequentially. Thus, both of the numbers of sequential and parallel steps in Step 3 are $O(n)$. The numbers of sequential and parallel steps in Step 4 are $O(n!)$ and $O(n)$ in case that object $\langle FALS E \rangle$ is sent out as the output.

Since the number of types of objects in the P system Π_{HCP} is $O(n^2)$, and $O(n^2)$ kinds of evolution rules are used in Step 2, we obtain the following theorem for Π_{HCP} .

Theorem 2: The asynchronous P system Π_{HCP} , which computes the Hamiltonian cycle problem with n vertices, works in $O(n!)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ types of objects and evolution rules of size $O(n^3)$. $\square$

5. Conclusions

In the present paper, we proposed two P systems that solve SAT and Hamiltonian cycle problem. The proposed P systems are fully asynchronous, i.e. any number of applicable rules may be applied in one step of the P system. We showed that the proposed P systems works in a polynomial number of steps in maximal parallel manner, and also showed that the P systems works sequentially.

Acknowledgments

This research was partially supported by the Ministry of Education, Culture, Sports Science and Technology, Grand-in-Aid for Young Scientists (B), 20700013.

References

- [1] G. Păun, “Computing with membranes,” *J. Comput. Syst. Sci.*, vol.61, no.1, pp.108–143, 2000.
- [2] G. Păun, *Introduction to Membrane Computing*, Springer, 2006.
- [3] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, and F.J. Romero-Campero, “A uniform solution to SAT using membrane creation,” *Theor. Comput. Sci.*, vol.371, no.1–2, pp.54–61, 2007.
- [4] V. Manca, “DNA and membrane algorithms for SAT,” *Fundamenta Informaticae*, vol.49, no.1–3, pp.205–221, 2002.
- [5] L.Q. Pan and A. Alhazov, “Solving HPP and SAT by P systems with active membranes and separation rules,” *Acta Informatica*, vol.43, no.2, pp.131–145, 2006.
- [6] G. Păun, “P systems with active membranes: Attacking NP-complete problems,” *J. Autom. Lang. Combin.*, vol.6, no.1, pp.75–90, 2001.
- [7] M.J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini, “A polynomial complexity class in P systems using membrane division,” *J. Autom. Lang. Combin.*, vol.11, no.4, pp.423–434, 2003.
- [8] C. Zandron, C. Ferretti, and G. Mauri, “Solving NP-complete problems using P systems with active membranes,” *Unconventional Models of Computation*, pp.289–301, 2000.
- [9] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, and A. Riscos-Núñez, “A fast P system for finding a balanced 2-partition,” *Soft Computing*, vol.9, no.9, pp.673–678, 2005.
- [10] M.J. Pérez-Jiménez and A. Riscos-Núñez, “A linear-time solution to the knapsack problem using P systems with active membranes,” *Membrane Computing*, vol.2933, pp.250–268, 2004.

- [11] M.J. Pérez-Jiménez and A. Riscos-Núñez, "Solving the subset-sum problem by P systems with active membranes," *New Generation Computing*, vol.23, no.4, pp.339–356, 2005.
- [12] M.J. Pérez-Jiménez and F. Romero-Campero, "Solving the BIN PACKING problem by recognizer P systems with active membranes," *The Second Brainstorming Week on Membrane Computing*, pp.414–430, 2004.
- [13] Z. Dang and O.H. Ibarra, "On one-membrane P systems operating in sequential mode," *Int. J. Found. Comput. Sci.*, vol.16, no.5, pp.867–881, 2005.
- [14] R. Freund, A. Leporati, M. Oswald, and C. Zandron, "Sequential P systems with unit rules and energy assigned to membranes," *Machines, Computations, and Universality*, vol.3354, pp.200–210, 2005.
- [15] G. Ciobanu, L.Q. Pan, G. Păun, and M.J. Pérez-Jiménez, "P systems with minimal parallelism," *Theor. Comput. Sci.*, vol.378, no.1, pp.117–130, 2007.
- [16] G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, *Applications of Membrane Computing*, Springer, 2006.

Hirofumi Tagawa received the B.E. and M.E. degrees in Computer Science and Electronics from Kyushu Institute of Technology in 2008 and 2010, respectively. His research interests include algorithms for membrane computing.

Akihiro Fujiwara received the B.E. degree in Osaka University in 1993, and received the M.E. and Ph.D. degrees in Nara Institute of Science and Technology (NAIST) in 1995 and 1997, respectively. He is now an associate professor of Kyushu Institute of Technology. His main research interests are network algorithms, and algorithms for natural computing. He is a member of IEEE.