Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 8, Number 2, pages 74–80, July 2019

An asynchronous P system with branch and bound for solving Hamiltonian cycle problem
(preliminary version)

Akihiro Fujiwara Koutaro Umetsu Fumiya Nozato
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan
fujiwara@cse.kyutech.ac.jp

Abstract—Membrane computing, which is a computational model based on cell activity, has considerable attention as one of new paradigms of computations. In the general membrane computing, computationally hard problems have been solved in a polynomial number of steps using an exponential number of membranes. However, reduction of the number of membranes must be considered to make P system more realistic model.

In the paper, we propose an asynchronous P system with branch and bound, which is a well known optimization technique, to reduce the number of membranes. The proposed P system solves Hamiltonian cycle for a graph with n vertices, and works in $O(n!)$ sequential steps or $O(n^2)$ parallel steps.

In addition, the number of membranes used in the proposed P system is evaluated using a computational simulation. The experimental results show validity and efficiency of the proposed P system.

Index Terms—membrane computing, Hamiltonian cycle, branch and bound

I. Introduction

A number of next-generation computing paradigms have been considered due to limitation of silicon-based computational hardware. As an example of the computing paradigms, natural computing, which works using natural materials for computation, has considerable attention. Membrane computing, which is a representative of the natural computing, is a computational model inspired by the structures and behaviors of living cells.

A basic feature of the membrane computing was introduced in [10] as a P system. 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. Each object evolves according to evolution rules associated with a membrane in which the object is contained.

The P system and most variants have been proved to be universal [12], and several P systems have been proposed for solving computationally hard problems [2], [3], [6], [7], [9], [11], [13]–[16], [19].

In addition, asynchronous parallelism has been considered on the P system. 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 makes P system a more realistic computational model.

A number of asynchronous P systems have been proposed for the computationally hard problems in [1], [4], [8], [17], [18]. For example, an asynchronous P system has been proposed for solving Hamiltonian cycle in [17]. The P system solves Hamiltonian cycle for a graph with n vertices in $O(n!)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

In all of the above P systems, the computationally hard problems have been solved in polynomial numbers of steps using exponential numbers of membranes. The number of membranes means the number of living cells, and reduction of the number of membranes must be considered in case that the P system is implemented using living cells because living cells cannot be created exponentially.

Recently, an asynchoronous P system using branch and bound has been proposed in [5] for reducing the number of membranes. The branch and bound is a well known optimization technique, and the technique is used in the P system for omitting partial value assignments that cannot satisfy a given Boolean formula.

In the paper, we propose an asynchronous P system for solving Hamiltonian cycle with branch and bound. In the proposed P system, a partial permutation of vertices is created, and then, existence of edges between vertices is checked for the partial permutation. If there is no edge between the vertices, the partial permutation is discarded as a bounding operation. Since the number of membranes increases according to the number of created permutations of vertices, the number can be reduced by omitting permutations that must not be a Hamiltonian cycle.

We show that the proposed P system solves Hamiltonian cycle for a graph with n vertices, and works in $O(n!)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

In addition, validity of the proposed system is evaluated using a computational simulation. In the simulation, various instances are executed on the previous P system

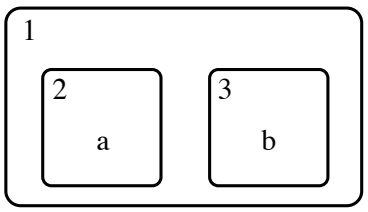

Fig. 1. An example of membrane structure

[17] and the proposed P system, and the number of membranes are compared for the same instances. The experimental results show validity and efficiency of the proposed P system with branch and bound.

The remainder of the present paper is organized as follows. In Section 2, we give a brief description of the model for asynchronous membrane computing. In Section 3, we propose the P system with branch and bound for Hamiltonian cycle, and experimental results for the proposed P system are shown in Section 4. Section 5 concludes the paper.

II. Preliminaries

A. Computational model for membrane computing

Several models have been proposed for membrane computing. 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 by 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 Figure 1 denotes 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 according to 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.

We now formally define a P system and the sets used in the system as follows.

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

O: O is the set of all objects used in the system.

μ: μ is membrane structure that consists of m membranes. Each membrane in the structure is labeled with an integer.

ω_i: Each ω_i is a set of objects initially contained only in the membrane labeled i.

R_i: 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 outermost membrane, and computation is started by applying evolution rules. We also assume that output objects are sent from the outermost 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:

$$[\ a\ ]_h \to [\ b\ ]_h$$

In the above rule, h is a label of the membrane and $a, b \in O$. Using the rule, an object a evolves into another object b. (We omit the brackets in each evolution rule such as $a \to b$ for cases that a corresponding membrane is obvious.)

(2) Send-in communication rule:

$$a[\ ]_h \to [\ b\ ]_h$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using the rule, an object a is sent into the membrane, and can evolve into another object b.

(3) Send-out communication rule:

$$[\ a\ ]_h \to [\ ]_h b$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using the rule, an object a is sent out of the membrane, and can evolve into another object b.

(4) Dissolution rule:

$$[\ a\ ]_h \to b$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using the rule, the membrane, which contains object a, is dissolved, and the object can evolve into another object b. (The outermost membrane cannot be dissolved.)

(5) Division rule:

$$[\ a\ ]_h \to [\ b\ ]_h[\ c\ ]_h$$

In the above rule, h is a label of the membrane, and $a, b, c \in O$. Using the rule, the membrane, which contains object a, is divided into two membranes that contain objects b and c, respectively.

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

B. Maximal parallelism and asynchronous parallelism

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 computation 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 objects, 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, evolution rules are applied in a fully asynchronous manner on the asynchronous P system [17], and 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 the maximal parallel manner.

The reason why we consider the asynchronous parallelism in this paper is based on the fact that every living cell acts independently and asynchronously. Since the standard P system ignores the asynchronous feature of living cells, the asynchronous P system is a more realistic computation model for cell activities.

In the asynchronous P system, all evolution rules can be applied completely in parallel, which is the same as the conventional P system, or all evolution rules can be applied sequentially. We define the number of steps executed in the asynchronous P system in the maximal parallel manner as the number of parallel steps. We also define the number of steps in the case that the applicable evolution rules are applied sequentially as the number of sequential steps. The numbers of parallel and sequential steps indicate the best and worst case complexities for the asynchronous P system. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

III. An asynchronous P system with branch and bound for Hamiltonian cycle

In this section, we present an asynchronous P system with branch and bound for Hamiltonian cycle. We first explain an input and an output of the problem for the system, and then, show an outline and details of the P system. Finally, we discuss complexity of the proposed P system.

A. Input and output for Hamiltonian cycle

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. 2 has a Hamiltonian cycle $(v_1, v_3, v_2, v_4, v_1)$.

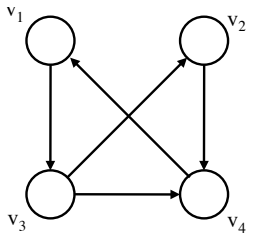

Fig. 2. A directed graph containing a Hamiltonian cycle $(v_1, v_3, v_2, v_4, v_1)$.

In the present paper, an output of the problem is one of two values "TRUE" and "FALSE". An output value is "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.

$$\begin{aligned} O_V &= \{\langle v_i \rangle \mid 1 \leq i \leq n\} \\ O_E &= \{\langle e_{i,j}, W \rangle \mid 1 \leq i \leq n, 1 \leq j \leq n, W \in \{T, F\}\} \end{aligned}$$

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. 2.

$$\begin{aligned} O_V &= \{\langle v_1 \rangle, \langle v_2 \rangle, \langle v_3 \rangle, \langle v_4 \rangle\} \\ O_E &= \{\langle e_{1,1}, F \rangle, \langle e_{1,2}, F \rangle, \langle e_{1,3}, T \rangle, \langle e_{1,4}, F \rangle, \\ &\quad \langle e_{2,1}, F \rangle, \langle e_{2,2}, F \rangle, \langle e_{2,3}, F \rangle, \langle e_{2,4}, T \rangle, \\ &\quad \langle e_{3,1}, F \rangle, \langle e_{3,2}, T \rangle, \langle e_{3,3}, F \rangle, \langle e_{3,4}, T \rangle, \\ &\quad \langle e_{4,1}, T \rangle, \langle e_{4,2}, F \rangle, \langle e_{4,3}, F \rangle, \langle e_{4,4}, F \rangle\} \end{aligned}$$

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 one of the following two objects. 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.

B. Branch and bound technique for Hamiltonian cycle

Branch and bound is a well known computing paradigm for optimization problem. On P system for solving Hamiltonian cycle [17], all permutation of vertices are created for an input graph with n vertices, and $(n-1)!$ permutations are checked whether the cycle is Hamiltonian. However, a partial permutation of vertices can be discarded if the output is determined to be "FALSE".

Fig. 3 shows a search tree for illustration of the above idea. Let a directed graph in Fig. 2 be an input graph. In the search tree, a permutation of vertices is denoted by a path from root node to a leaf node. In this case, a

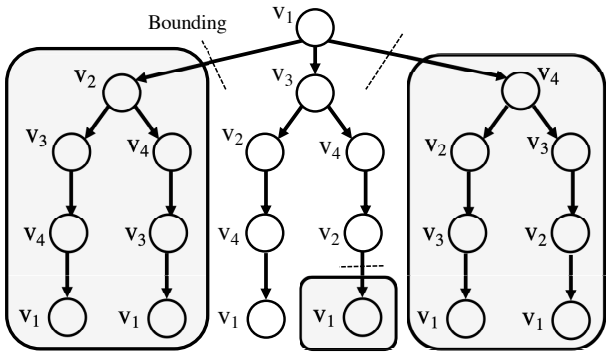

Fig. 3. An example of branch and bound for Hamiltonian cycle

path starting with an edge from v_1 to v_2 can be bounded because there is no edge between v_1 and v_2, and the permutation must not be cycle. In another case, a path starting with an edge from v_1 to v_4 is bounded similarly.

We now explain an overview of the asynchronous P system with branch and bound for solving Hamiltonian cycle. An initial membrane structure for the computation is $[\ [\ ]_2\ ]_1$. We call the membranes labeled 1 and 2 *outer* and *inner* membranes, respectively.

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

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

Step 2: In each inner membrane, repeat the following (2-1) and (2-2) until "TRUE" or "FALSE" is outputted.

(2-1) Select a vertex, which is not visited, as a next vertex. In case that there is an edge from the last vertex in the permutation to the next vertex, the next vertex is added to the partial permutation of vertices. (The partial permutation denotes a path from a start vertex.) The partial permutation with the next vertex is created by dividing the inner membrane.
In case that there is no edge from the last vertex to the next vertex, the partial permutation is discarded, and an object that denotes "FALSE" is outputted. (The divided membrane stops the computation.)

(2-2) In case that length of the permutation is equal to the number of vertices, check the created permutation of vertices whether the permutation denotes a Hamiltonian cycle. If there exist an edge from the last vertex in the permutation to the start vertex, output an object that denotes "TRUE", otherwise, output an object that denotes "FALSE".

Step 3: Send out a final result, "TRUE" or "FALSE", from the outer membrane.

C. Details of the P system

We now explain details 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 in the paper is asynchronous, we cannot move the input objects in parallel, and input objects are moved one by one applying following two sets of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{OUT,1} \;=\; & \{\langle e_{1,1}, F\rangle[\ ]_2 \to [\langle M_{2,1}\rangle\langle e_{1,1}, F\rangle]_2\} \\ & \cup \{\langle M_{i,j}\rangle\langle e_{i,j}, V\rangle[\ ]_2 \to [\langle M_{i+1,j}\rangle\langle e_{i,j}, V\rangle]_2 \\ & \quad \mid 1 \le i \le n, 1 \le j \le n, V \in \{T, F\}\} \\ & \cup \{\langle M_{i,n+1}\rangle\langle v_i\rangle[\ ]_2 \to [\langle M_{i+1,n+1}\rangle\langle v_i\rangle]_2 \\ & \quad \mid 1 \le i \le n\} \\ & \cup \{\langle M_{n+1,j}\rangle \to \langle M_{1,j+1}\rangle \mid 1 \le j \le n+1\} \\ & \cup \{\langle M_{1,n+2}\rangle \to \langle OUT\rangle\langle C\rangle, \\ & \quad \langle C\rangle[\ ]_2 \to [\langle C_{1,1}\rangle]_2\} \end{aligned}$$

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{IN,1} \;=\; & \{[\langle M_{i,j}\rangle]_2 \to [\ ]_2\langle M_{i,j}\rangle \\ & \quad \mid \ 2 \le i \le n+1, 1 \le j \le n+1\} \end{aligned}$$

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, two objects, $\langle OUT\rangle$ and $\langle C\rangle$, are created in the outer membrane. The created object $\langle OUT\rangle$ is used for outputting an object that denotes "TRUE" or "FALSE" at the end of the final step. The other created object $\langle C\rangle$ is sent into the inner membrane as $\langle C_{1,1}\rangle$, and the object triggers computation of Step 2.

We next explain details of Step 2. Step 2 consists of sub-steps (2-1) and (2-2). At the beginning of Step 2, the following evolution rule is executed for setting status of the start vertex v_1 to visited.

$$R_{IN,(2\text{-}0)} = \{\langle C_{1,1}\rangle\langle v_1\rangle \to \langle C_{2,1}\rangle\langle \lambda_1\rangle\langle z_{1,1}\rangle\}$$

In the evolution rules, object $\langle \lambda_j\rangle$ denotes that vertex v_j is visited, and object $\langle z_{i,j}\rangle$ denotes that i-th vertex in the permutation is v_j.

In (2-1), unvisited vertex is selected as a next vertex, and the next vertex is added to the partial permutation of vertices in case that there is an edge from the last vertex in the permutation to the next vertex. (The partial permutation denotes a path from a start vertex.) The (2-1) is executed using the following set of evolution rules. The partial permutation with the next vertex is created by dividing the inner membrane with the first set of evolution

rules, and the unvisited next vertex is selected using the second set of evolution rules.

$$\begin{aligned} R_{IN,(2\text{-}1\text{-}1)} &= \{[\langle z_{k-1,i}\rangle\langle C_{k,j}\rangle\langle v_j\rangle\langle e_{i,j},T\rangle]_2 \\ &\quad \to [\langle z_{k-1,i}\rangle\langle C_{k,j+1}\rangle\langle v_j\rangle\langle e_{i,j},T\rangle]_2 \\ &\quad\quad [\langle z_{k-1,i}\rangle\langle C_{k+1,1}\rangle\langle z_{k,j}\rangle\langle \lambda_j\rangle]_2 \\ &\quad\quad \mid 2\le k\le n, 1\le i\le n, 1\le j\le n\} \\ &\quad \cup\{\langle C_{k,j}\rangle\langle\lambda_j\rangle \to \langle C_{k,j+1}\rangle\langle\lambda_j\rangle \\ &\quad\quad \mid 2\le k\le n, 1\le j\le n\} \end{aligned}$$

On the other hand, in (2-1), the partial permutation is discarded in case that there is no edge from the last vertex to the next vertex. In this case, membrane division is executed to select another next vertex, and the an object that denotes "FALSE" is outputted by applying the following set of evolution rules.

$$\begin{aligned} R_{IN,(2\text{-}1\text{-}2)} &= \{[\langle z_{k-1,i}\rangle\langle C_{k,j}\rangle\langle v_j\rangle\langle e_{i,j},F\rangle]_2 \\ &\quad \to [\langle z_{k-1,i}\rangle\langle C_{k,j+1}\rangle\langle v_j\rangle\langle e_{i,j},F\rangle]_2 \\ &\quad\quad [\langle FALSE,n-k\rangle]_2 \\ &\quad\quad \mid 2\le k\le n, 1\le i\le n, 1\le j\le n\} \\ &\quad \cup\{[\langle FALSE,n-k\rangle]_2 \\ &\quad\quad \to [\ ]_2\langle FALSE,n-k\rangle \\ &\quad\quad \mid 2\le k\le n\} \end{aligned}$$

In the above evolution rules, object $\langle FALSE, n-k\rangle$ is created to count the number of "FALSE" in Step 3.

In (2-2), the created permutation of vertices is checked whether the permutation denotes a Hamiltonian cycle. In the following two sets of evolution rules, the first set is applied for outputting an object that denotes "TRUE" in case that there exist an edge from the last vertex in the permutation to the start vertex. Otherwise, the second set is applied for outputting an object that denotes "FALSE".

$$\begin{aligned} R_{IN,(2\text{-}2)} &= \{[\langle C_{n+1,1}\rangle\langle z_{n,i}\rangle\langle e_{i,1},T\rangle]_2 \\ &\quad \to [\ ]_2\ \langle TRUE\rangle \mid 1\le i\le n\} \\ &\quad \cup\{[\langle C_{n+1,1}\rangle\langle z_{n,i}\rangle\langle e_{i,1},F\rangle]_2 \\ &\quad \to [\ ]_2\ \langle FALSE,0\rangle \mid 1\le i\le n\} \end{aligned}$$

We now summarize the set of evolution rules for Step 2 as follows.

(Evolution rules for the inner membrane)

$$R_{IN,2} = R_{IN,(2\text{-}0)} \cup R_{IN,(2\text{-}1\text{-}1)} \cup R_{IN,(2\text{-}1\text{-}2)} \cup R_{IN,(2\text{-}2)}$$

In Step 3, a final result is sent out from the outer membrane. The Step 3 is executed applying the following sets of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{OUT,3} &= \{\{[\langle TRUE\rangle\langle OUT\rangle]_1 \to [\ ]_1\langle TRUE\rangle\} \\ &\quad \cup\{\langle FALSE,k\rangle^{k+1} \to \langle FALSE,k+1\rangle \\ &\quad\quad \mid 0\le k\le n-2\} \\ &\quad \cup\{\{[\langle FALSE,n-1\rangle\langle OUT\rangle]_1 \\ &\quad\quad \to [\ ]_1\langle FALSE\rangle\} \end{aligned}$$

If object $\langle TRUE\rangle$ is in the outer membrane, there is a Hamiltonian cycle in the input graph, and the object is sent out from the outer membrane immediately applying the first evolution rules. On the other hand, objects that denotes "FALSE" is outputted from all inner membranes in case that the final result is "FALSE". Therefore, the sum of "FALSE" is counted asynchronously using the second set of evolution rules, and final object $\langle FALSE\rangle$ is outputted if and only if all outputs of inner membranes are "FALSE".

We now summarize the asynchronous P system $\Pi_{\mathrm{BB-HC}}$ for solving Hamiltonian cycle as follows.

$$\Pi_{\mathrm{BB-SAT}} = (O, \mu, \omega_1, \omega_2, R_{OUT}, R_{IN})$$

- $O = O_V \cup O_E \cup O_{TRUE} \cup O_{FALSE} \cup O_M \cup O_C \cup O_\lambda \cup O_z$
- $O_{TRUE} = \{\langle TRUE\rangle\}$
- $O_{FALSE} = \{\langle FALSE\rangle\} \cup \{\langle FALSE,k\rangle \mid 1\le k\le n\}$
- $O_M = \{\langle M_{i,j}\rangle \mid 1\le i\le n+1, 1\le j\le n+1\}$
- $O_C = \{\langle C_{i,j}\rangle \mid 1\le i\le n+1, 1\le j\le n+1\}$
- $O_\lambda = \{\langle\lambda_j\rangle \mid 1\le j\le n\}$
- $O_z = \{\langle z_{i,j}\rangle \mid 1\le i\le n, 1\le j\le n\}$
- $\mu = [\ [\ ]_2\ ]_1$
- $\omega_1 = \omega_2 = \phi$
- $R_{OUT} = R_{OUT,1} \cup R_{OUT,3}$, $R_{IN} = R_{IN,1} \cup R_{IN,2}$

D. Complexity of the P system

We now consider the complexity of asynchronous P system Π_{BB-HC}. Since $O(n^2)$ objects are moved from the outer membrane into the inner membrane sequentially, Step 1 can be executed in $O(n^2)$ parallel or sequential steps using $O(n^2)$ kinds of objects and $O(n^2)$ kinds of evolution rules.

In Step 2, $O((n-1)!)$ membranes are created in the worst case, and evolution rules are applied sequentially at most $O(n)$ times in each membrane. Therefore, Step 2 can be executed in $O(n^2)$ parallel steps and $O(n!)$ sequential steps using $O(n^3)$ kinds of objects and $O(n^3)$ kinds of evolution rules.

In the final step, $O(n)$ evolution rules are applied sequentially in cast that the output is "FALSE", and Step 3 can be executed in $O(n)$ parallel step or $O(n)$ sequential step using $O(n)$ kinds of objects and $O(n)$ kinds of evolution rules.

Therefore, we obtain the following theorem for the asynchronous P system Π_{BB-HC}.

Theorem 1: The asynchronous P system Π_{BB-HC}, which computes Hamiltonian cycle for a directed graph

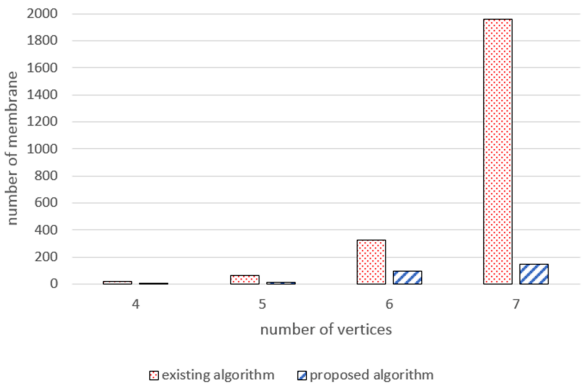

Fig. 4. Experimental results

with n vertices, works in $O(n!)$ sequential steps or $O(n^2)$ parallel steps by using $O(n^2)$ types of objects and evolution rules of size $O(n^2)$. □

IV. Experimental simulations

We develop an original simulator for asynchronous P systems using Python, and compare the numbers of membranes used in executions of an existing P system [17] and our proposed P system for Hamiltonian cycle ptoblem.

Since the simulator executes P systems with asynchronous parallelism, all of the evolution rules are applied in fully asynchronous manner, in other words, any number of applicable evolution rules are applied in each step of executions on the simulator. Therefore, applied evolution rules are different among executions on the simulator, and output of the simulation may be different for the same input. We first implement the existing P system and the proposed P system for Hamiltonian cycle problem on the simulator, and execute simulations for various inputs. In the simulation, valid results are obtained for all inputs. In other word, all outputs are the same for the same input on all executions.

Next, we compare the number of membranes used on the existing P system and the proposed P system for Hamiltonian cycle problem. For each the number of vertices n $(4 \leq n \leq 7)$, 10 inputs are randomly created, and the existing P system and the proposed P system are simulated for the given inputs.

Figure 4 shows that average values of the number of membranes on the simulation. The number of membranes of the existing P system increases exponentially to the number of vertices n. Although the number of membranes on the proposed P system also increases according to n, the number of membranes on the proposed P system is more than 70 percent less than the number of membranes on the existing P system for each n $(3 \leq n \leq 7)$.

V. Conclusions

In this paper, we proposed the asynchronous P system with branch and bound for solving Hamiltonian cycle. The P system with branch and bound reduces the number of membranes by discarding partial permutations that cannot be Hamiltonian cycle.

The proposed P systems are fully asynchronous, and works in a polynomial number of steps in the maximal parallel manner and also works sequentially. Although the number of sequential steps is a factorial of the number of vertices, the result means that the proposed P system works for any combinations of sequential and asynchronous applications of evolution rules, and guarantees that the P systems can output a correct solution in the case that any number of evolution rules are synchronized.

We experimented with the proposed P system and the existing P system, and the experimental results show that the proposed P system outputs valid results, and also show that the number of the membrane on the proposed P system is at most 70 percent less than the number of membranes on the existing P system for Hamiltonian cycle.

In our future research, we are considering reduction of the number of objects and evolution rules used in the proposed P system. We are also considering asynchronous P systems with branch and bound for other computationally hard problems.

VI. Acknowledgments

This research was partially supported by JSPS KAKENHI, Grand-in-Aid for Scientific Research (C), 16K00021.

References

[1] R. Freund. Asynchronous P systems and P systems working in the sequential mode. In International workshop on Membrane Computing, pages 36–62, 2005.

[2] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A fast P system for finding a balanced 2-partition. Soft Computing, 9(9):673–678, 2005.

[3] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero. A uniform solution to SAT using membrane creation. Theoretical Computer Science, 371(1-2):54–61, 2007.

[4] J. Imatomi and A. Fujiwara. An asynchronous P system for MAX-SAT. In 8th International Workshop on Parallel and Distributed Algorithms and Applications, pages 572–578, 2016.

[5] Y. Jimen and A. Fujiwara. Asynchronous P systems for hard graph problems. International Journal of Networking and Computing, 8(2):141–152, 2018.

[6] A. Leporati, C. Zandron, and G. Mauri. Solving the factorization problem with P systems. Progress in Natural Science, 17(4):471–478, 2007.

[7] V. Manca. DNA and membrane algorithms for SAT. Fundamenta Informaticae, 49(1-3):205–221, 2002.

[8] T. Murakawa and A. Fujiwara. Arithmetic operations and factorization using asynchronous P systems. International Journal of Networking and Computing, 2(2):217–233, 2012.

[9] L. Q. Pan and A. Alhazov. Solving HPP and SAT by P systems with active membranes and separation rules. Acta Informatica, 43(2):131–145, 2006.

[10] G. Păun. Computing with membranes. Journal of Computer and System Sciences, 61(1):108–143, 2000.

[11] G. Păun. P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics, 6(1):75–90, 2001.
[12] G. Păun. Introduction to Membrane Computing. Springer, 2006.
[13] M. J. Pérez-Jiménez and A. Riscos-Núñez. A linear-time solution to the knapsack problem using P systems with active membranes. Membrane Computing, 2933:250–268, 2004.
[14] M. J. Pérez-Jiménez and A. Riscos-Núñez. Solving the subset-sum problem by P systems with active membranes. New Generation Computing, 23(4):339–356, 2005.
[15] M. J. Pérez-Jiménez and F.J. Romero-Campero. Solving the BIN PACKING problem by recognizer P systems with active membranes. In The Second Brainstorming Week on Membrane Computing, pages 414–430, 2004.
[16] M. J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics, 11(4):423–434, 2003.
[17] H. Tagawa and A. Fujiwara. Solving SAT and Hamiltonian cycle problem using asynchronous p systems. IEICE Transactions on Information and Systems (Special section on Foundations of Computer Science), E95-D(3), 2012.
[18] K. Tanaka and A. Fujiwara. Asynchronous P systems for hard graph problems. International Journal of Networking and Computing, 4(1):2–22, 2014.
[19] C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In Unconventional Models of Computation, pages 289–301, 2000.