Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 975798, 7 pages
http://dx.doi.org/10.1155/2013/975798

Research Article

Time-Free Solution to Hamilton Path Problems Using P Systems with *d*-Division

Tao Song,[1] Xun Wang,[2] and Hongjiang Zheng[3]

[1] *School of Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China*
[2] *Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 3050006, Japan*
[3] *School of Information Engineering, Tarim University, Akesu, Xinjiang 843300, China*

Correspondence should be addressed to Hongjiang Zheng; zhjwhut@gmail.com

Received 25 June 2013; Accepted 14 August 2013

Academic Editor: Michael Meylan

P systems with *d*-division are a particular class of distributed and parallel computing models investigated in membrane computing, which are inspired from the budding behavior of Baker's yeast (a cell can generate several cells in one reproducing cycle). In previous works, such systems can theoretically generate exponential working space in linear time and thus provide a way to solve computational hard problems in polynomial time by a space-time tradeoff, where the precise execution time of each evolution rule, one time unit, plays a crucial role. However, the restriction that each rule has a precise same execution time does not coincide with the biological fact, since the execution time of biochemical reactions can vary because of external uncontrollable conditions. In this work, we consider timed P systems with *d*-division by adding a time mapping to the rules to specify the execution time for each rule, as well as the efficiency of the systems. As a result, a time-free solution to Hamiltonian path problem (**HPP**) is obtained by a family of such systems (constructed in a uniform way), that is, the execution time of the rules (specified by different time mappings) has no influence on the correctness of the solution.

1. Introduction

Membrane computing, introduced in [1], is one of the recent branches of natural computing. The computing systems investigated in membrane computing are distributed and parallel computing devices, usually called P systems. The aim is to abstract computing concepts and ideas (i.e., models, data structures, data manipulation operations, operation control modes, etc.) from the structure and the functioning of living cells, considered both individually and as part of complexes, such as tissues and organs like the brain. There are two main classes of P systems investigated: cell-like P systems [1] and tissue-like P systems [2]. In the cell-like P systems, the membrane structure graphically represented by a tree and in tissue-like P systems, the membrane structure is a general graph. In tissue-like P systems, we can find tissue P systems, neural P systems, and numerical P systems. Such three models have different alphabets, rules, and semantics, but all three keep the same membrane structure [3]. During the past years, many variants of P systems have been developed and proved to be universal (do what Turing machine can do) [4–6], and a particular class of P systems with cell reproduction properties (such as P systems with active membranes [7], tissue P systems with cell division [8], and spiking neural P systems with budding [5]) that can theoretically generate exponential working space in linear time have been used to solve computational hard problems in polynomial (even in linear) time [8–10]. An introduction to the area of membrane computing can be found in [7], while an overview of this field can be found in [3, 11], with up-to-date information available at the membrane computing website [12]. In the present work, we deal with a variant of cell-like P systems with active membranes, called P systems with *d*-division, which was inspired from the budding behavior of Baker's yeast (a cell can generate several new cells in a reproducing cycle) [7].

Briefly, a P system with *d*-division consists of a hierarchical membrane structure, a number of objects and evolution rules. In every membrane, objects (represented by multisets over a given alphabet) and evolution rules are present, where objects correspond to the chemical compounds that exist

inside the cells, and the evolution rules correspond to the chemical reactions taking place in the cells. By using the evolution rules, objects can evolve to other objects, and the membrane structure can change. The membranes can also have an electrical charge, positive (+), negative (−), or neutral (0). A global clock is assumed to mark the time for the system. In each instant, if a rule is applicable over objects existing inside a membrane, the rule must be applied on the tick of the clock. If more than one rule is applicable at certain moment, one of these rules is nondeterministically chosen to use. Each cell in the systems works in a sequential way; in each time unit only one evolution can be used, but; for different cells, they work in a synchronous way. The configuration of the systems at a given instant of time is described by both the membrane structure and the multisets of objects present in each membrane. The systems can proceed from one configuration to another by applying evolution rules (it is said that a transition takes place from one configuration to the next one), which takes exactly one time unit. A sequence of transitions between configurations define a computation. A computation halts if it reaches a configuration where no rule can be applied in any membrane. In this case, the result of the computation can be interpreted as the number of objects inside a specified output membrane. P systems with d-division ($d \geq 2$) can theoretically generate exponential working space in linear time and thus provide a way to solve computational hard problems in polynomial time by a time-space tradeoff, where the precise execution time of each evolution rule (one time unit) plays a crucial role [7].

In [8], inspired by the mitosis function of living cells, tissue P systems with cell division were proposed. The system can generate working space (cells) by using cell division rules and thus can generate exponential working space in linear steps. The newly generated working space provides a rich source for computation, particular for solving computational hard problems. In [8], tissue P systems with cell division can solve **SAT** problem in a polynomial time.

In [7, 12], many variants of P systems with different strategies of generating working space have been reported, such as P systems with cell separation and SN P systems with budding. These systems can expand working space (by generating new cells or neurons) during the computation and thus can be used to solve computation hard problems in feasible (polynomial or even linear) time.

In the results, the time cost of a computation is obtained by counting the steps used in the computation to solve the problem. In the systems, a global clock is assumed to mark the time of the systems, and there is a restriction that each biological operation should be completed in exactly one time unit, even for different operations. In P systems, each operation usually corresponds to a particular biochemical reaction, so it is not natural to impose that different operation should cost the same time.

However, programming living things cannot assume neither general restrictions on execution time nor the presence of global clocks synchronizing the execution of different parallel processes. Moreover, the time of execution of certain biological processes could vary because of external uncontrollable conditions. Therefore, it seems crucial to investigate P systems when such timing assumption is not used.

In this work, we consider P systems with d-division without the time assumption, which is achieved by adding a time mapping to specify the execution time of all the rules in the systems. The obtained systems are close to the biological fact, and their computational properties, particularly in computational efficiency, need to be investigated. The systems with d-division are considered by adding a time mapping to the rules to specify the execution time of each rule. The obtained systems are called timed tissue P systems with d-division. Particularly interesting are time-free P systems with d-division where, given an arbitrary time mapping for a system, results computed by the system are always the same, independent of the assigned time mapping. Following this line of work, finding solutions for hard computational problems by means of time-free systems was explored in [13] from a theoretical point of view. In this paper, we prove (in a constructive way) that we can solve **NP**-complete problems by means of time-free P systems with d-division. Specifically, a family of time-free P systems with d-division, constructed in a uniform way, capable of solving **HPP** problem are presented.

2. Preliminaries

In what follows some required concepts of formal language theory are presented as the necessary background for the topics covered in subsequent sections of this paper. Readers needing additional background can refer to works such as [14].

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, while the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+.

By $\mathbb{N}$ we denote the set of nonpositive integers. Let U be arbitrary set. A multiset (over U) is a mapping $M : U \to \mathbb{N}$. The multiplicity of a in the multiset M can be denoted by $M(a)$ with any $a \in U$. It can be expressed by the pair $(a, M(a))$. If the set $U = \{a_1, a_2, \ldots, a_n\}$ is finite, a multiset M over U, represented by the set of mappings $\{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ can also be represented by a string $w = a_1^{M(a_1)} a_2^{M(a_2)} \cdots a_n^{M(a_n)}$ or by any of its permutations.

In what concern to the rest of this paper, we will not distinguish between the representation of multiset in mapping form or string form.

3. Time-Free Solutions to Decision Problems by Means of P Systems with d-Division

It is started by briefly recalling the formal definition of P systems with d-division introduced in [7], and then a timed extension of such systems is reviewed, followed by a description of time-free P systems with d-division. Finally, some notions of time-free solutions to decision problems by means of such systems are discussed.

3.1. P Systems with d-Division. A *P system with d-division* ($d \geq 2$) of degree m is a construct

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R), \quad \text{where} \tag{1}$$

(i) $m \geq 1$ is the initial number of membranes of the system;

(ii) O is the alphabet of *objects*;

(iii) H is a finite set of *labels* for membranes;

(iv) μ is the initial *membrane structure*, consisting of m membranes; membranes are labelled (not necessarily in an injective way) with elements of H and are electrically polarized, being possible charge positive (+), negative (−), or neutral (0);

(v) $w_1, \ldots, w_m$ are strings over O, describing the *initial multisets of objects* placed in the m regions of μ;

(vi) R is a finite set of *evolution rules*, of the following types:

(a) $[a \to v]_h^\alpha$, $h \in H$, $\alpha \in \{+, -, 0\}$, $a \in O$, $v \in O^*$;

(b) $a[]_h^{\alpha_1} \to [b]_h^{\alpha_2}$, $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in O$;

(c) $[a]_h^{\alpha_1} \to []_h^{\alpha_2} b$, $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in O$;

(d) $[a]_h^\alpha \to b$, $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in O$;

(e) $[a]_h^\alpha \to [b_1]_h^{\alpha_1} [b_2]_h^{\alpha_2} \cdots [b_d]_h^{\alpha_d}$, $h \in H$, $\alpha_1, \alpha_2, \ldots, \alpha_d \in \{+, -, 0\}$, $a, b_1, b_2, \ldots, b_d \in O$;

(f) $[[]_{h_1}^{\alpha_1} \cdots []_{h_k}^{\alpha_1} []_{h_{k+1}}^{\alpha_2} \cdots []_{h_n}^{\alpha_2}]_{h_0}^{\alpha_0} \to [[]_{h_1}^{\alpha_3} \cdots []_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5} [[]_{h_{k+1}}^{\alpha_4} \cdots []_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$, $k \geq 1$, $n > k$, $h_i \in H$, $0 \leq i \leq n$, and $\alpha_0, \ldots, \alpha_6 \in \{+, -, 0\}$ with $\{\alpha_1, \alpha_2\} = \{+, -\}$.

Rules of type (a) are objects evolution rules, whose application is controlled by both the label and the charge of the membranes. Rules of type (b) and (c) are communication rules, by which an object can be sent into or out of the membrane. The objects can be possibly modified during this process, as well as the polarization of the membrane can be changed, but not its label. The dissolving rules are of type (d), by which a membrane can be dissolved, while the object specified in the rule can be modified. The division rules for elementary membranes are of type (e). In reaction with an object, the elementary membrane can be divided into d membranes with the same labels, possibly of different polarizations; the object specified in the rule is replaced by some newly generated objects in the newly membranes. As shown in rules of type (f), if a membrane contains other membranes that have charges + or − but have globally neutral charge, these membranes with same charges can be be separated into two new membranes. The previous rules can be considered as "standard" rules; two possible extensions for rules of type (e) can be also considered:

(e′) $[a]_h^\alpha \to [b_1]_{h_1}^{\alpha_1} [b_2]_{h_2}^{\alpha_2} \cdots [b_d]_{h_d}^{\alpha_d}$, $h, h_1, h_2, \ldots h_d \in H$, $\alpha, \alpha_1, \alpha_2, \ldots, \alpha_d \in \{+, -, 0\}$, $a, b_1, b_2, \ldots, b_d \in O$, $d \geq 2$;

(e″) $[a]_h^{\alpha_1} \to [b_1]_h^{\alpha_2} [b_2]_h^{\alpha_2} \cdots [b_d]_h^{\alpha_d}$, $h \in H$, $\alpha_1, \alpha_2, \ldots, \alpha_d \in \{+, -, 0\}$, $a, b_1, b_2, \ldots, b_d \in O$, $d \geq 2$.

By rules of type (e′), the membranes produced by division possibly have different labels with the divided membrane, and the polarizations of the new membranes can be different from the polarization of the initial one. Division rules (e″) allow the division of nonelementary membranes, where all membranes from the initial membranes will be replicated and will appear in the new copies of this membrane.

In each time unit (a global clock is assumed, marking the time for the whole system), the selected rules to be executed in each cell are applied on the tick of the clock, taking exactly one time unit to complete their execution. In a given time, if an object can evolve by more than one rules, only one of the rules is nondeterministically selected. All objects and membranes not specified in any executed rule remain unchanged.

If a membrane is dissolved, all the objects contained in the region delimited by such membrane are left free in the surrounding region. If in a given computation step a membrane is divided and there are objects in this membrane which can evolve, it is assumed that evolution of objects takes place first; thus, the evolved objects will be present in the divided membranes. Objects that do not evolve are replicated. Thus, rules are applied "bottom-up"; that is, rules in the innermost membrane are applied first and, subsequently, level by level up to the region of the skin membrane. The skin membrane can never divide, but it can be "electrically charged."

Note that every cell in the systems works in a sequential way, in each time unit only one evolution can be used, but, for different cells, they work in a synchronous way.

The configuration of the system at a given time unit is described by both the membrane structure and the multisets of objects present in each cell. By applying the rules specified above, one can define a transition between configurations. A sequence of transitions starting from the initial configuration are called a computation. A computation halts if it reaches a configuration such that no rules can be applied in any membrane. During a computation, objects can leave the skin membrane and pass to the environment. The result of a halting computation is defined as the number of objects that are sent out of the system during the computation. A nonhalting computation provides no result.

3.2. Timed and Time-Free P Systems with d-Division. Given a P system with d-division

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R) \tag{2}$$

and a mapping $e : R \to \mathbb{N}$, it is said that

$$\Pi(e) = (O, H, \mu, w_1, \ldots, w_m, R, e) \tag{3}$$

is a timed P system with d-division such that $e : R \to \mathbb{N}$ specifies the execution time of each rule of the system.

A timed P system with d-division $\Pi(e)$ works in the following way. An external clock is assumed, which marks time units of equal length, starting from instant 0. According

to this clock, the step t of computation is defined by the period of time that goes from instant $t-1$ to instant t. If a membrane i contains some rule r from types (a)–(e$''$) selected to be executed, execution of such rule takes $e(r)$ time units to complete. Therefore, if the execution is started at instant j, the rule is completed at instant $j+e(r)$ and the resulting objects and membranes become available only at the beginning of step $j+e(r)+1$.

For a given P systems with d-division, by adding different time mappings, we can obtain a family of timed P systems with d-division. The systems obtained, although in the same family, may produce different computation results for having different time mapping. A timed P system with d-division $\Pi(e)$ is said to be time-free if and only if, for any time mapping e, the system $\Pi(e)$ produces the same computation result (if any); that is, the execution time of the rules has no influence on the computation result of such systems.

Time-free P systems with d-division are particularly interesting because they allow modelling biological phenomena where the execution time of the processes involved can vary unexpectedly. Similarly, a system like that, if implemented, would be robust against environmental changes that could affect, in an unpredicted manner, the execution time of the evolution rules of the system.

3.3. Uniform Time-Free Solutions to Decision Problems by Timed P Systems with d-Division. In order to formally define the concept of uniform time-free solution to decision problems by timed P systems (in our case with d-division), following from [9, 13], some notions related to solving decision problems with P systems with d-division are reviewed. It is started by introducing recognizer timed P systems with d-division.

A recognizer P system with d-division is the one such that (i) the working alphabet contains two distinguished elements `yes` and `no`; (ii) all computations halt; and (iii) if C is a computation of the system, either object `yes` or object `no` (but not both) must have been released into the environment when the system halts. The recognizer P systems with d-division can be used to solve decision problems as follows. For an instance γ of the problem X, the computations of the systems start from an initial configuration by adding the code of the problem $\text{cod}(\gamma)$ to the systems. If the computations of the systems finally halt with object `yes` (`no`) present in the environment, we say that the problem has a positive (negative) answer. In recognizer P systems with d-division, a computation is said to be accepting (rejecting) if the object `yes` (`no`) appears in the environment associated with the corresponding halting configuration.

In the following description, we will formally describe the notions of solutions to decision problems by P systems with d-division. Let $X=(I_X,\Theta_X)$ be a decision problem, where I_X is a set of instances and Θ_X is a predicate over I_X, and let $\Pi=\{\Pi(n)\mid n\in\mathbb{N}\}$ be a family of recognizer P systems with d-division. We say that the decision problem X is solvable in polynomial time by a (countable) family of P systems with d-division Π if the following holds.

(i) The family Π is polynomial uniform by Turing machine; that is, there exists a deterministic Turing machine constructing the system $\Pi(n)$ for $n\in\mathbb{N}$ in polynomial time.

(ii) There is a pair of polynomial computable functions (cod, s) over the problem X such that

(a) for each instance $u\in I_X$, $\text{cod}(u)$ is the input of the system Π and $s(u)\in\mathbb{N}$;

(b) the family Π is *sound* with respect to X, cod, s; that is, for each instance of the problem $u\in I_X$ if there exists an accepting computation of $\Pi(s(u))$ with input $\text{cod}(u)$, we have $\Theta_X(u)=1$;

(c) the family Π is *complete* with respect to X, cod, s; that is, for each instance of the problem $u\in I_X$ if $\Theta_X(u)=1$, every computation of $\Pi(s(u))$ is an accepting computation;

(d) the family Π is *polynomial bounded* if there exists a polynomial function p such that, for each $u\in I_X$, all computations in $\Pi(s(u))$ halt in, at most, $p(|u|)$ steps.

The notions of soundness and completeness described above can be extended to timed P systems with d-division, with the exception that the polynomially bounding needs to be reconsidered. In timed P systems with d-division, execution of time of rules is determined by the time mapping e; thus, the existence of rules whose execution time is inherently exponential is possible. Therefore, the polynomial bounded restriction on the computation time cannot be assured any more.

To circumvent this, another way to define computation steps in timed P systems is considered, based on *rule starting steps* (RS steps for short). Hence, although there exists an external clock ticking, marking the time for the system, only those ticks of the clock in which at least one rule starts its execution are considered as the beginning of a computation step. The ticks in which no object "begins" to evolve or no membrane "begins" to change, are omitted from computation steps.

With the previous definitions we can then define the concepts of timed soundness, timed completeness and timed polynomially bounding for recognizer timed P systems with d-division. Let $\Pi=\{\Pi(n,e)\mid n\in\mathbb{N}\}$ be a family of timed recognizer P systems with d-division;

(e) the family Π is *timed sound* with respect to X, cod, s; that is, for each instance of the problem $u\in I_X$ and a given time mapping e, if there exists an accepting computation of $\Pi(s(u),e)$ with input $\text{cod}(u)$, we have $\Theta_X(u)=1$;

(f) the family Π is *timed complete* with respect to X, cod, s; that is, for each instance of the problem $u\in I_X$ and a given time mapping e, if $\Theta_X(u)=1$, every computation of $\Pi(s(u),e)$ is an accepting computation;

(g) the family Π is *timed polynomial* bounded if there exists a polynomial function p such that, for each $u \in I_X$ and given time mapping e, all computations in $\Pi(s(u), e)$ halt in, at most, $p(|u|)$ RS-steps.

Similarly, we can define the concepts defined above for recognizer time-free P systems with d-division. Let $\Pi = \{\Pi(n, e) \mid n \in \mathbb{N}\}$ be a family of timed recognizer P systems with d-division;

(h) the family Π is *time-free* sound with respect to X, cod, s; that is, for each instance of the problem $u \in I_X$ and any time mapping e, if there exists an accepting computation of $\Pi(s(u), e)$ with input $\mathrm{cod}(u)$, then we have $\Theta_X(u) = 1$;

(i) the family Π is *time-free complete* with respect to X, cod, s; that is, for each instance of the problem $u \in I_X$ and any time mapping e, if $\Theta_X(u) = 1$, every computation of $\Pi(s(u), e)$ is an accepting computation;

(j) the family Π is *time-free polynomial bounded* if there exists a polynomial function p such that, for each $u \in I_X$ and any time mapping e, computations in $\Pi(s(u), e)$ halt in, at most, $p(|u|)$ RS-steps.

Finally, we define the concept of uniform time-free solution to decision problems.

Let $X = (I_X, \Theta_X)$ be a decision problem. A family of timed recognizer P systems $\Pi = \{\Pi(s(u), e) \mid u \in I_X\}$ is a uniform *time-free* solution to decision problem X if the following statements are true:

(k) the family Π is polynomially uniform with respect to Turing machines; that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(e)$ with knowledge involving only the size of the problem X for every instance of X;

(l) The family Π is time-free polynomially bounded (halts in a polynomial number of RS-steps);

(m) the family Π is time-free sound and time-free complete.

Given a decision problem X, if there exists a family of timed P system $\Pi = \{\Pi(s(u), e) \mid u \in I_X\}$ which computes a solution of the instance u of X, and the correctness of the solution does not depend on the execution time of the rules involved in the computation, then the solution is called a time-free solution to problem X computed by the systems Π.

4. A Uniform Time-Free Solution to HPP

In this section, we prove that a family of timed P systems with d-division can solve **HPP** in polynomial RS-steps, and the systems are constructed in a uniform way. It is noticed that in our systems we extend the number of objects involved in the rules of type (a). Specifically, the extended rules of type (a) are of the form

$$(a') \quad [u \to v]_h^{\alpha}, \ h \in H, \ \alpha \in \{+, -, 0\}, \ u, v \in O^*,$$

by which a multiset of objects can evolve. After that, we will prove that the correctness of our solution does not depend on the execution time of the rules; that is, for any time mapping e, we will obtain the same answer to the instances of the problems.

Hamiltonian Path Problem (**HPP**) is one of the best known **NP**-complete problems, which asks whether or not for a given graph $\gamma = (N, E)$ (N is the set of nodes and E is the set of edges in γ) contains a Hamiltonian path, that is a path of length n that visits all nodes from γ exactly once; we do not specify the first and last nodes of a path, and each node in node set N can be the first node or the last one.

Theorem 1. *The* **HPP** can be solved in polynomial RS steps with respect to the number of nodes of a directed graph by a family of timed P systems with d-division using rules of types (a'), (b), (c), and (e').

Proof. Let us consider a directed graph $\gamma = (N, E)$, where $N = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E is the set of edges. We codify γ by multiset $\mathrm{cod}(\gamma) = a_1 a_2 \cdots a_n$, where variable a_i represents vertex v_i. In this way, we pass from γ to $\mathrm{cod}(\gamma)$ in linear steps with respect to n. The instance γ will be processed by timed P systems with d-division $\Pi(s(\gamma))$ with input $\mathrm{cod}(\gamma)$, where $s(\gamma) = n$.

We construct the family of timed P system with d-division of degree 2:

$$\Pi(n, e) = \left(O, H, \mu, w_0, w_p, R, e\right), \quad \text{where} \tag{4}$$

(i) $O = \{a_i \mid 1 \le i \le n\} \cup \{b_i \mid 1 \le i \le n\} \cup \{\mathtt{yes}, \mathtt{no}, c, c', d, t, t'\}$ is the alphabet, and each object a_i corresponds to node v_i, $1 \le i \le n$;

(ii) $H = \{0, 1, 2, \ldots, 2n + 1, p\}$ is the set of labels of the membranes;

(iii) $\mu = [[]_0]_p$ is the initial membrane structure;

(iv) $w_0 = dcb^{n-1}$ and $w_p = \mathtt{no}$;

(v) R is the set evolution rules;

(vi) $e : R \to \mathbb{N}$ is the time mapping for the rules from R.

In what follows, a detailed explanation of how the system $\Pi(n, e)$ computes a solution to **HPP** is present. Three stages in the computation process are considered: generating, checking, and outputting.

Generating stage:

G_1: $[\mathtt{no}]_p \to [\ ]_p \mathtt{no}$,

G_2: $[d]_0 \to [a_1]_1 [a_2]_1 \cdots [a_n]_1$.

At the beginning of the computation, object no can be ejected to the environment after $e(G_1)$ steps by the rule G_1. By using the rule G_2, we start to create the paths in γ starting from all the nodes v_i, $1 \le i \le n$. This process will be completed in $e(G_2)$ time units, costing only one RS step.

G_{3i}: $[b]_i \to [a_1]_{i+1}[a_2]_{i+1} \cdots [a_n]_{i+1}, 1 \le i \le n-1$.

Prolonging the paths with one node, where all the nodes are assumed to have an edge between each node. For the membrane with same label, the rule G_{3i} will be used in parallel; thus, for each i, the application of rule G_{3i} costs one RS step to prolong one node to all the nodes:

G_{4i}: $[a_k a_h \to \lambda]_i$ with $1 \le i \le n$ and $k, h \in \{1, 2, \ldots, n\}$ if there is no edge between node v_k and v_h.

By using rules G_{4i}, the objects representing the edges that are not in γ can evolve into empty string (thus are removed out of the systems), and only the objects representing the edges in γ are left in the system. In the worst case, each pair of (v_k, v_h) is not in E, so the membrane with label i will use at most n^2 rules in parallel, so this process will cost at most n^2 RS-steps for each $i \in \{1, 2, \ldots, n\}$. Note that it is assumed that when division rules and evolutions can both be used (in our case it is possible that rules G_{3i} and G_{4i} can be applied at the same time), evolution of objects takes place first; thus, the objects evolved by rule G_{3i} will be present in the divided membranes by rule G_{4i} for any $i \in \{1, 2, \ldots, n\}$:

G_5: $[c]_n \to [c']_{n+1}[c']_{n+1}$.

Since the length of Hamiltonian path is n, we need to check all the paths in γ with length n. After the application of rules G_{4i} is completed, the membranes with label n containing object c can divide. The generating stage will finish when membranes with label $n+1$ are generated.

Checking stage:

C_{1i}: $[a_i]_{n+i} \to [a]_{n+i+1}[a]_{n+i+1}$ with $i = 1, 2, \ldots, n-1$,

C_2: $[a_n]_{2n} \to [t']_{2n+1}[t]_{2n+1}$.

After generating all the possible paths with length not more than n (each in a separated membrane with label n), we need to check whether there exist some paths that contain all the nodes $v_1, v_2, \ldots, v_n$. This process is started by checking object a_1 in the membranes $n+1$. For each membrane $n+1$, if it contains object a_1, rule C_{11} can be applied, and after $e(C_{11})$ steps, membranes with label $n+2$ are produced. Membranes $n+1$ without object a_1 can not evolve any more, and no further rule can be used over them. After membranes labelled with $n+2$ are generated, object a_2 in the membranes can evolve by using the rule C_{12}, and so forth. The checking stage will cost n RS steps at most (in case there is at least one membrane $n+1$ that contains all the objects $a_1, a_2, \ldots, a_n$).

If there is no membrane $n+1$ containing all the objects $a_1, a_2, \ldots, a_n$, the checking stage halts and cannot go to outputting stage. There are the following two cases. If the rule G_1 has completed its application, that is object `no` presents in the environment. In this case, we get object `no` in the environment when system halts; hence, the answer to the decision problem is negative. When the computation of the system halts the application of rule G_1 does not complete. In this case, the system halts as soon as the application of rule G_1 is finished. We also get object `no` in the environment, hence the answer of the problem is negative.

If there exists at least one membrane $n+1$ that contains all the objects $a_1, a_2, \ldots, a_n$, the computation proceeds to the outputting stage.

Outputting stage:

O_1: $[t]_{2n+1} \to [\ ]^0_{2n+1}\texttt{yes}$

O_2: $\texttt{no}[\texttt{yes}]_p \to [\texttt{no}]_p\texttt{yes}.$

Membranes with label $2n+1$ that contains object t will be produced. Object t in membrane $2n+1$ can be sent out of the membrane and evolves into object `yes`. This process costs one RS step. In this way, object `yes` passes to membrane p. Subsequently, by using rule O_2, object `yes` in membrane p is sent out to the environment and object `no` goes into membrane p. After that, the system halts with object `yes` placed in the environment, so the answer to the decision problem is positive.

We can easily check that the presented system $\Pi(n, e)$ can be built in polynomial time with respect to n (the number of nodes in N of γ). Specifically, the necessary resources to build the system $\Pi(n, e)$ are shown as follows: the size of the alphabet, $2n + 7 \in O(n)$; the initial number of cells, $2 \in O(1)$; the initial number of objects, $2n + 2 \in O(n)$; and the number of rules (in the most case), $n^3 + 2n + 4 \in O(n^3)$.

Therefore, there exists a deterministic Turing machine working in polynomial time which constructs the system with knowledge only of the number of nodes in the graph; that is, the family of P systems with d-division $\Pi(n, e)$ is constructed in a uniform way. It is easy to check that the family of systems $\Pi(n, e)$ are timed sound and timed complete.

In the generating stage, the applications of rules G_1 and G_2 cost two RS steps; every membrane i applies rules G_{3i}, $1 \le i \le n-1$ in parallel; thus, $n-1$ RS-steps are required to start their execution; for any membrane i, the application of rule G_{4i} costs n^2 RS steps at most to start their execution, (since in each membrane i, in the worst case it contains all the rules $[a_k a_h \to \lambda]_i$ with $h, k = 1, 2, \ldots, n$); for all the n membranes, totally we need n^3 RS-steps for the execution of rules G_{4i}; one RS step is needed to start the execution of rule G_5. In the checking stage, it costs at most n RS steps in the case that there exists at least one membrane $n+1$, contains all the variables $a_1, a_2, \ldots, a_n$. Two RS steps are needed to output the object `yes` (if any) to the environment. Therefore, $n^3 + 2n + 4$ RS-steps are required at most to complete the computation of $\Pi(n, e)$, which is polynomial.

As explained above, we can conclude that the family of timed P systems $\Pi(n, e)$ with d-division is a uniform solution to **HPP**. □

Corollary 2. *For any time mapping e, by means of the family of P systems with d-division $\Pi(n, e)$ constructed in Theorem 1, the same solution can be obtained for a given instance of the* **HPP**; that is, the correctness of the solution does not depend on the execution time of the rules.

Proof. It is easy to check that (for any time mapping e) the family of P systems with d-division $\Pi(n, e)$ constructed in

Theorem 1 are time-free sound, time-free complete, and time-free polynomially bounded. Therefore, the solution to **HPP** by the family of timed P systems with d-division $\Pi(n, e)$ is time-free. □

5. Conclusions and Future Work

In this paper, inspired from the biological fact that execution time of biochemical reactions can vary because of external uncontrollable conditions, we consider timed P systems with d-division by adding a time mapping to each evolution rule. In the systems, the general restriction on the constant execution time of the rules as defined in classical P systems is removed. We consider rule starting steps (RS-steps) as "valid" computation steps; that is, only those ticks of the clock in which at least one rule starts its execution are considered as the beginning of a computation step. The ticks, in which no object "begins" to evolve or no membrane "begins" to change, are omitted from computation steps. In this way, we can avoid the inherently possible exponential execution time of the rules in such systems. We investigate the efficiency of timed P systems with d-division. As a result, a time-free uniform solution to **HPP** is obtained.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (51268051, 61074169, and 61033003), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), and the Natural Science Foundation of Hubei Province (2011CDA027 and 2011CDB233).

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[2] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón, "Tissue P systems," *Theoretical Computer Science*, vol. 296, no. 2, pp. 295–326, 2003.

[3] G. Păun, G. Rozenberg, and A. Salomaa, Eds., *Handbook of Membrane Computing*, Oxford University Press, Oxford, UK, 2010.

[4] T. Song, L. Pan, and G. Păun, "Asynchronous spiking neural P systems with local synchronization," *Information Sciences*, vol. 219, pp. 197–207, 2013.

[5] T. Song, L. Pan, J. Wang, I. Venkat, K. G. Subramanian, and R. Abdullah, "Normal forms of spiking neural P systems with anti-spikes," *IEEE Transactions on Nanobioscience*, vol. 11, no. 4, pp. 352–359, 2012.

[6] T. Song, X. Wang, Z. Zhang, and Z. Chen, "Homogenous spiking neural P systems with anti-spikes," *Neural Computing and Applications*, 2013.

[7] G. Păun, *Membrane Computing. An Introduction*, Springer, Berlin, Germany, 2002.

[8] G. Păun, M. J. Perez-Jimenez, and A. Riscos-Nunez, "Tissue P systems with cell division," *International Journal of Computers Communications & Control*, vol. 3, no. 3, 2008.

[9] L. Pan and C. Martín-Vide, "Further remark on P systems with active membranes and two polarizations," *Journal of Parallel and Distributed Computing*, vol. 66, no. 6, pp. 867–872, 2006.

[10] L. Pan, D. Díaz-Pernil, and M. J. Pérez-Jiménez, "Computation of Ramsey numbers by P systems with active membranes," *International Journal of Foundations of Computer Science*, vol. 22, no. 1, pp. 29–38, 2011.

[11] G.-X. Zhang and L.-Q. Pan, "A survey of membrane computing as a new branch of natural computing," *Chinese Journal of Computers*, vol. 33, no. 2, pp. 208–214, 2010.

[12] The P System, http://ppage.psystems.eu.

[13] C. Matteo, "Time-free solution to hard computational problems," in *Proceeding of 10th Brainstorming Week on Membrane Computing*, vol. 1, Sevilla, Spain, 2012.

[14] G. Rozenberg and A. Salomaa, Eds., *Handbook of Formal Languages*, Springer, Berlin, Germany, 1991.