Linear Time Solution to Prime Factorization by Tissue P Systems with Cell Division

Xingyi Zhang[1], Yunyun Niu[2], Linqiang Pan[2], Mario J. Pérez-Jiménez[3]

[1] School of Computer Science and Technology
Anhui University, 230039 Hefei, China
xyzhanghust@gmail.com
[2] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology, 430074 Wuhan, China
niuyunyun1003@163.com, lqpan@mail.hust.edu.cn
[3] Department of Computer Science and Artificial Intelligence
University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Summary. Prime factorization is useful and crucial for public-key cryptography, and its application in public-key cryptography is possible only because prime factorization has been presumed to be difficult. A polynomial-time algorithm for prime factorization on a quantum computer is given by P. W. Shor in 1997. In this work, a linear-time solution for prime factorization is given on a kind of biochemical computational devices – tissue P systems with cell division, instead of physical computational devices.

1 Introduction

In math, *prime factorization* is the breaking down of a composite number into smaller primes, which when multiplied together equal the original integer. Currently, though the prime factorization problem is not known to be **NP**-hand, no efficient algorithm is publicly known. It is generally considered intractable. The presumed computational hardness of this problem is at the heart of several algorithms in cryptography such as RSA [15].

Many areas of mathematics and computer science have been brought to bear on the prime factorization problem, including elliptic curves, algebraic number theory, and quantum computing. A polynomial-time algorithm for prime factorization on a quantum computer is given by P. W. Shor in 1997 [16]. This will have significant implications for cryptography if a large quantum computer is ever built. However, before a practical quantum computer appears, it is still of interest to find any reasonable computational devices for solving prime factorization problem. In this work, we shall give a linear-time solution to prime factorization on a class of

biochemical computational devices – *tissue P systems with cell division*, instead of physical computational devices.

Tissue P systems with cell division is a class of computational devices in *membrane computing*. Membrane computing is an emergent branch of natural computing, which is inspired by the structure and the functioning of living cells, as well as the organization of cells in tissues, organs, and other higher order structures. The devices in membrane computing, called *P systems*, provide distributed parallel and non-deterministic computing models. Since Gh. Păun introduced the first P system in [12], this area is heavily investigated. Please refer to [13] for an introduction of membrane computing, and refer to [17] for further bibliography.

Informally, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous, non-deterministic, maximally parallel manner. *Tissue P systems* are a class of P systems, where membranes are placed in the nodes of a graph. It is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules, which was introduced to P systems in [11]. Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. This model has two biological inspirations (see [9]): intercellular communication and cooperation between neurons. In [14], tissue P systems are endowed with the ability of getting new cells based on the mitosis or cellular division, thus obtaining the ability of generating an exponential amount of workspace in polynomial time. Such variant of tissue P systems is called *tissue P systems with cell division*.

Tissue P systems with cell division were widely investigated for solving **NP**-complete problems. Some of them deal with non-numerical **NP**-complete decision problems, such as SAT problem [14], 3-coloring problem [2], vertex cover [4]. Others deal with numerical **NP**-complete decision problems, that is, decision problems whose instances consist of sets or sequences of integer numbers, such as subset sum [3], partition problem [5]. Although prime factorization we shall consider is a numerical problem, it is neither a decision problem nor an optimization problem. In this work, we shall construct a family of tissue P systems with cell division, which can decompose integer numbers in a linear time with respect to the length of binary representation of the integer to be factored. As a result of computation, a prime number is sent to a prefixed output membrane, instead of `yes` or `no`.

Up to now, besides there are two polynomial-time solutions to prime factorization by P systems with active membranes [6, 10], one well known polynomial algorithm that solves factorization problem is based on quantum computer [16]. As the case of quantum computer, the solution given in this work indicates how powerful tissue P systems with cell division can be, although at this moment nobody knows how to build a biochemical computer.

The paper is organized as follows. In Section 2, some preliminaries are recalled. The formal definition of tissue P systems with cell division is given in Section 3. A family of tissue P systems that uniformly solve the factorization problem is

presented in Section 4, with a short overview of the computation and the necessary resources. Conclusions and comments are presented in Section 5.

2 Preliminaries

An *alphabet* Σ is a non-empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a subset from Σ^*.

A *multiset* m over a set A is a pair (A, f), where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset, then its *support* is defined as $supp(m) = \{x \in A \,|\, f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over A, and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}\}$. That is, superscripts indicate the multiplicity of each element. If $f(x) = 0$ for any $x \in A$, then this element is omitted.

3 Tissue P Systems with Cell Division

In [8, 9], the first definition of the model of tissue P systems was proposed, where the membrane structure did not change along the computation. We now shall introduce a model of *tissue P systems with cell division* based on the cell-like model of P systems with membranes division [14]. The biological inspiration of this model is clear: alive tissues are not *static* network of cells, since new cells are generated by membrane fission in a natural way.

The main features of this model, from the computational point of view, are that cells are not polarized (the contrary holds in the cell-like model of P systems with active membranes, see [13]); the cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells or with the environment is blocked during the division process. In some sense, this means that while a cell is dividing it closes its communication channels with other cells and with the environment.

Formally, a *(function) computing tissue P system with cell division* of degree $q \geq 1$ and order (m, n), $m \geq 1, n \geq 1$, is a tuple of the form

$$\Pi = (\Gamma, \Sigma, \Lambda, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_{out}),$$

where:

1. Γ is the *alphabet* of *objects*;

2. $\Sigma = \{a_1, \ldots, a_m\}$ is an ordered input alphabet strictly contained in Γ;
3. $\Lambda = \{b_1, \ldots, b_n\}$ is an ordered output alphabet contained in Γ;
4. $w_1, \ldots, w_q$ are strings over Γ, describing the initial multisets of objects placed in the cells of the system at the beginning of the computation;
5. $\mathcal{E} \subseteq \Gamma$ is the set of objects in the environment in arbitrarily copies each;
6. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
   Communication rules; $1, 2, \cdots, q$ identify the cells of the system, 0 is the environment; when applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and simultaneously the objects of the multiset v are sent from region j to region i ($|u| + |v|$ is called the length of the communication rule $(i, u/v, j)$);
   (b) $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$, $a, b, c \in \Gamma$, and $i \neq i_{out}$;
   Division rules; in reaction with an object a, the cell is divided into two cells with the same label; all the objects in the original cells are replicated and copies of them are placed in each of the new cells, with the exception of the object a, which is replaced by the object b in the first new cell and by c in the second one; the output cell cannot be divided;
7. $i_{in} \in \{1, 2, \ldots, q\}$ is the input cell;
8. $i_{out} \in \{0, 1, 2, \ldots, q\}$ is the output cell.

The rules of a system as above are used in the non-deterministic maximally parallel manner. In each step, all cells which can evolve must evolve in a maximally parallel way (in each step we apply a multiset of rules which is maximal, no further rule can be added). This way of applying rules has only one restriction when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not evolve by means of communication rules. Their labels precisely identify the rules which can be applied to them.

A configuration of tissue P system with cell division is described by all multisets of objects over Γ associated with all the cells present in the system and the multiset of objects over $\Gamma - \mathcal{E}$ associated with environment. The initial configuration of the system Π with input $w \in \Sigma^*$ is the tuple $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \emptyset)$; that is, the corresponding configuration after adding the multiset w to the content of the input cell i_{in}. The computation starts from the initial configuration and proceeds as defined above. When there is no rule can be applied, the computation stops. Only halting computations give a result. If $\mathcal{C} = \{C^i\}_{i<r}$ is a halting computation, where C^i are configurations, then the result of computation $Output(\mathcal{C}) = (C^{r-1}_{b_1}(i_{out}), C^{r-1}_{b_2}(i_{out}), \ldots, C^{r-1}_{b_n}(i_{out}))$, where $C^{r-1}_{b_j}(i_{out})$, $1 \leq j \leq n$, is the multiplicity of object b_j in the region i_{out} in the halting configuration C^{r-1}.

For a function f, we denote the domain of f by $D(f)$ and the range of f by $R(f)$. For a tissue P system with cell division Π having ordered input alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$ and ordered output alphabet $\Lambda = \{b_1, b_2, \ldots, b_n\}$, and partial function $f : \mathbb{N}^m \rightarrow \mathbb{N}^n$, function f is encoded in a unary notation in the following

way: $(\alpha_1, \ldots, \alpha_m) \in D(f)$ is expressed by $a_1^{\alpha_1} a_2^{\alpha_2} \ldots a_m^{\alpha_m}$; $(\beta_1, \ldots, \beta_n) \in R(f)$ is expressed by $b_1^{\beta_1} b_2^{\beta_2} \ldots b_n^{\beta_n}$.

Definition 1. *We say that a partial function $f : \mathbb{N}^m \to \mathbb{N}^n$ is computed in polynomial time by a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of tissue P systems with cell division in unary encoding if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(t)$ from $t \in \mathbb{N}$.*
- *There exist a polynomial-time computable function s over the domain $D(f)$ of function f such that:*
  - *for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, $s(u)$ is a natural number and $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (f, s), that is, there exists a polynomial function p, such that for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$ every computation of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ is halting and, moreover, it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (f, s), that is, for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, if there exists a computation $\mathcal{C}$ of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ such that $Output(\mathcal{C}) = (\beta_1, \ldots, \beta_n)$, then $f(u) = (\beta_1, \cdots, \beta_n)$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (f, s), that is, for each $u = (\alpha_1, \ldots, \alpha_m) \in D(f)$, if $f(u) = (\beta_1, \cdots, \beta_n)$, then every computation $\mathcal{C}$ of $\Pi(s(u))$ with input $a_1^{\alpha_1} \ldots a_m^{\alpha_m}$ has $Output(\mathcal{C}) = \{\beta_1, \ldots, \beta_n\}$.*

In the Definition 1, the input and output are encoded in unary notation. However, in classical complexity theory, based upon Turing machine, switching from binary to unary encoding generally corresponds to simplify the problem. In this work, binary encoding is used for integer factorization problem. In what follows, we will give the definition that a function is computed by a family of P systems with cell division in binary encoding. In the case of binary encoding, the input alphabet is not asked to be ordered, and no output alphabet is fixed.

A *(function) computing tissue P system with cell division with input* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, \Sigma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_{out}),$$

where:

1. Γ is the *alphabet* of *objects*;
2. Σ is an (un-ordered) input alphabet strictly contained in Γ;
3. $w_1, \ldots, w_q$ are strings over Γ, describing the initial multisets of objects placed in the cells of the system at the beginning of the computation;
4. $\mathcal{E} \subseteq \Gamma$ is the set of objects in the environment in arbitrarily copies each;
5. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;

(b) $[a]_i \to [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$, $a, b, c \in \Gamma$, and $i \neq i_{out}$;
6. $i_{in} \in \{1, 2, \ldots, q\}$ is the input cell;
7. $i_{out} \in \{0, 1, 2, \ldots, q\}$ is the output cell.

In semantics, P systems having un-ordered alphabets is the same with P systems with ordered input and output alphabets except for the way of encoding input and output. In the unary encoding, the sizes of ordered input and output alphabets are related with the dimensions of domain and range of function that is computed. Specifically, an ordered input alphabet $\{a_1, \ldots, a_m\}$ and an ordered output alphabet $\{b_1, \ldots, b_n\}$ can encode each function whose domain (resp. range) is a subset of $\mathbb{N}^{m'}$ $(m' \leq m)$ (resp. $\mathbb{N}^{n'}$ $(n' \leq n)$). In the binary encoding, the size of alphabet is related with both input value and output value. For example, for the function $f(x) = 2^{2^x}$ $(x \in \mathbb{N})$ and an input n, the length of input n in binary expression is $\lfloor \lg n \rfloor + 1$, and the length of output $f(n)$ in binary expressions is $2^n + 1$, which is an exponential function with respect to $\lfloor \lg n \rfloor + 1$. For functions such as $f(x) = 2^{2^x}$, maybe, we need exponential (with respect to the input size) large alphabet to encode the function in P systems, hence we cannot construct a family of P systems with cell division in polynomial time by Turing machine to compute functions such as $f(x) = 2^{2^x}$. It depends on the property of function whether a function can be computed by tissue P systems with cell division in binary encoding.

For prime factorization problem, the factors are less than the integer to be factored. In fact enables us to find a reasonable binary encoding for prime factorization problem. Specifically, we shall use the method from [7] to encode binary numbers by multisets of objects. Let $x_{k-1}, \cdots, x_1, x_0$ (with $k \geq 1$) be the binary representation of integer $x \geq 0$, that is, $x = \sum_{i=0}^{k-1} x_i 2^i$. We use the objects from the following alphabet $\mathcal{A}_k$, for $k \geq 1$:

$$A_k = \{\langle b, j \rangle \mid b \in \{0, 1\}, j \in \{1, 2, \cdots, k\}\}.$$

Objects $\langle b, j \rangle$ is used to represent bit b in position j in the binary encoding of an integer number. Hence, to represent the above number x we will use the following multiset (actually, a set) of objects:

$$\langle x_{k-1}, k-1 \rangle, \cdots, \langle x_1, 1 \rangle, \langle x_0, 0 \rangle.$$

Let us remark that the alphabet $\mathcal{A}_k$ depends on the length of the binary representation of the number x. Moreover, it is clear that with $\mathcal{A}_k$ we can represent all integer numbers in the range $0, 1, \cdots, 2^k - 1$. In order to distinguish between the objects that represent the bits of different integers A and B, a leading label A, B are used to mark each element in the multiset. To this aim, the alphabet $\mathcal{A}_k$ is modified as follows:

$$\mathcal{A}'_k = \{\langle l, b, j \rangle \mid l \in \{A, B\}, b \in \{0, 1\}, j \in \{1, 2, \cdots, k\}\}.$$

In this way, the i-th bit of A (that is, a_i) and the j-th bit of B (that is, b_j) are represented by the objects $\langle A, a_i, i \rangle$ and $\langle B, b_j, j \rangle$, respectively.

In general, we give the following definition that a function is computed by P systems with cell division in binary encoding.

Definition 2. *We say that a partial function $f : \mathbb{N} \to \mathbb{N}$ is computed in polynomial time by a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of tissue P systems with cell division in binary encoding if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(t)$ from $t \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over the domain $D(f)$ of function f such that:*
  - *for each $u \in D(f)$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (f, cod, s), that is, there exists a polynomial function p, such that for each $u \in D(f)$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (f, cod, s), that is, for each $u \in D(f)$, if there exists a computation $\mathcal{C}$ of $\Pi(s(u))$ with input $cod(u)$ and the objects in region i_{out} in the last configuration of $\mathcal{C}$ encode $(\beta_1, \cdots, \beta_q) \in \mathbb{N}^q$, then $f(u) = (\beta_1, \cdots, \beta_q)$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (f, cod, s), that is, for each $u \in D(f)$, if $f(u) = (\beta_1, \cdots, \beta_q) \in \mathbb{N}^q$, then in every computation of $\Pi(s(u))$ with input $cod(u)$, the objects in region i_{out} in the last configuration encode $(\beta_1, \cdots, \beta_q)$.*

4 A Linear Time Solution to the Factorization Problem

When we discuss the prime factorization problem, it is necessary to distinguish two different versions of the problem: decision problem version and function problem version.

The decision problem version of prime factorization can be formulated as "is n a composite number?" (or equivalently: "is n a prime number?"). This version is natural and useful because most well-studied complexity classes are defined as classes of decision problems, not function problems. But the decision problem version of prime factorization is much easier than the problem of finding the factors of n. Specifically, it can be solved in polynomial time (with respect to the number of digits of n) with the AKS primality test [1].

The function problem version of prime factorization: given an integer n, find an integer d with $1 < d < n$ that divides n (or conclude that n is prime). It is trivially in the class FNP, but we do not known whether it lies in class FP or not. This version is generally considered intractable, which means that no polynomial-time (with respect to the instance size) algorithm is known that solves it on every

instance; and it is the version solved by most practical implementations. In this work, we shall consider a restricted version of prime factorization problem, based on the following two facts. (1) Given an algorithm for integer factorization, one can factor any integer down to its constituent prime factors by repeated application of this algorithm. (2) Not all numbers of a given length are equally hard to factor. Semiprimes (the product of two prime numbers) are believed as the hardest instances of integer factorization for currently known techniques.

Problem 1. NAME: factorization.
– INSTANCE: a positive integer number which is the product of two prime numbers.
– OUTPUT: the prime factor that is not greater than another one.

Next, we shall construct a family $\{\Pi(k)\}_{k\in\mathbb{N}}$ of tissue P systems with cell division to factor integers, where each system $\Pi(k)$ can decompose all numbers of length k in binary form, provided that an appropriate input multiset is given. The resolution is a brute force algorithm, which consists of the following stages:

- *Generation Stage*: By division, all the possible pairs of integer numbers of length k in binary form are produced (one pair for each membrane with label 2).
- *Pre-checking Stage*: In this stage, the product of each pair of integer numbers of length k is calculated.
- *Checking Stage*: The system checks whether or not there exists a pair of integer numbers such that their product equals to the number n to be composed.
- *Output Stage*: The system sends to the output region a prime number.

For each $k \in \mathbb{N}$,

$$\Pi(k) = (\Gamma(k), \Sigma(k), w_1, w_2, \mathcal{R}(k), \mathcal{E}(k), i_{in}, i_{out}),$$

with the following components:

- $\Gamma(k) = \Sigma(k) \cup \{a_i, b_i, \langle X, 0, i\rangle, f_i, g_i \mid 0 \le i \le k-1\} \cup$
  $\{\langle A, j, i\rangle, \langle B, j, i\rangle, \langle A', j, i\rangle, \langle B', j, i\rangle \mid 0 \le i \le k-1, 0 \le j \le 1\} \cup$
  $\{\langle A_j, l, i\rangle, \langle B_j, l, i\rangle \mid 0 \le i \le k-1, 0 \le j \le \lceil \lg k\rceil + 1, 0 \le l \le 1\} \cup$
  $\{c_i \mid 0 \le i \le 4k + \lceil \lg 2k\rceil + \lceil \lg k\rceil + 5\} \cup \{c'_i \mid 1 \le i \le \lceil \lg k\rceil + 2k + 3\} \cup$
  $\{\langle C, 0, i\rangle, \langle C, 1, i\rangle \mid 0 \le i \le 2k-1\} \cup \{\langle i, j\rangle \mid 0 \le i, j \le k-1\} \cup$
  $\{d_i \mid -1 \le i \le k-2\} \cup \{e_i \mid -1 \le i \le k-1\} \cup \{z\}$.
- $\Sigma(k) = \{\langle n, 0, i\rangle, \langle n, 1, i\rangle \mid 0 \le i \le k-1\}$.
- $w_1 = \{\{c_0\}\}$.
- $w_2 = \{\{a_0 a_1 \cdots a_{k-1} b_0 b_1 \cdots b_{k-1} z\}\} \cup \{\{\langle i, j\rangle \mid 0 \le i, j \le k-1\}\}$.
- $\mathcal{R}(k)$ is the set of rules:

  1. **Division rule:**
     $r_{1,i} \equiv [a_i]_2 \to [\langle A, 0, i\rangle]_2[\langle A, 1, i\rangle]_2$, for $0 \le i \le k-1$;
     $r_{2,i} \equiv [b_i]_2 \to [\langle B, 0, i\rangle]_2[\langle B, 1, i\rangle]_2$, for $0 \le i \le k-1$.

2. **Communication rules:**

$r_{3,i} \equiv (1, c_i/c_{i+1}^2, 0)$, for $0 \leq i \leq 2k-1$;
$r_4 \equiv (1, c_{2k}/z, 2)$;
$r_{5,i} \equiv (2, c_{2k+i}/c_{2k+i+1}^2, 0)$, for $0 \leq i \leq \lceil \lg 2k \rceil - 1$;
$r_{6,i,j} \equiv (2, c_{2k+\lceil \lg 2k \rceil}\langle A, j, i\rangle / c_{2k+\lceil \lg 2k \rceil+1}\langle A_0, j, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{7,i,j} \equiv (2, c_{2k+\lceil \lg 2k \rceil}\langle B, j, i\rangle / c_{2k+\lceil \lg 2k \rceil+1}\langle B_0, j, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_8 \equiv (2, c_{2k+\lceil \lg 2k \rceil+1}/c'_1 c_{2k+\lceil \lg 2k \rceil+2}, 0)$;
$r_{9,i} \equiv (2, c_{2k+\lceil \lg 2k \rceil+i}/c_{2k+\lceil \lg 2k \rceil+i+1}, 0)$, for $2 \leq i \leq \lceil \lg k \rceil + 2k + 4$;
$r_{10,i} \equiv (2, c'_i/c'_{i+1}, 0)$, for $1 \leq i \leq \lceil \lg k \rceil + 2k + 2$;
$r_{11,i,j,l} \equiv (2, \langle A_j, l, i\rangle / \langle A_{j+1}, l, i\rangle^2, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq \lceil \lg k \rceil$, $0 \leq l \leq 1$;
$r_{12,i,j,l} \equiv (2, \langle B_j, l, i\rangle / \langle B_{j+1}, l, i\rangle^2, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq \lceil \lg k \rceil$, $0 \leq l \leq 1$;
$r_{13,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{14,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{15,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, j\rangle \langle i, j\rangle / \langle C, 0, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{16,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, j\rangle \langle i, j\rangle / \langle C, 1, i+j\rangle, 0)$,
for $0 \leq i, j \leq k-1$;
$r_{17,i} \equiv (2, \langle C, 0, i\rangle \langle C, 0, i\rangle / \langle C, 0, i\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{18,i} \equiv (2, \langle C, 0, i\rangle \langle C, 1, i\rangle / \langle C, 1, i\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{19,i} \equiv (2, \langle C, 1, i\rangle \langle C, 1, i\rangle / \langle C, 0, i\rangle \langle C, 1, i+1\rangle, 0)$, for $0 \leq i \leq 2k-2$;
$r_{20,i,j} \equiv (2, c'_{\lceil \lg k \rceil+2k+3}\langle C, 1, i\rangle \langle n, j, k-1\rangle / \lambda, 0)$,
for $k \leq i \leq 2k-2$, $0 \leq j \leq 1$;
$r_{21,i,j} \equiv (2, c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+5}\langle C, j, i\rangle \langle n, j, i\rangle / \langle X, 0, i\rangle, 0)$,
for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{22} \equiv (2, \langle X, 0, k-1\rangle / d_{k-2}, 0)$;
$r_{23,i} \equiv (2, d_i\langle X, 0, i\rangle / d_{i-1}, 0)$, for $0 \leq i \leq k-2$;
$r_{24} \equiv (2, d_{-1}/e_{k-1}, 0)$;
$r_{25,i,j} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, j, i\rangle \langle B_{\lceil \lg k \rceil+1}, j, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, j, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, j, i\rangle e_{i-1}, 0)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{26} \equiv (2, e_{-1}/f_0, 0)$;
$r_{27,i} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle \langle B_{\lceil \lg k \rceil+1}, 0, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, 1, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, 0, i\rangle f_0, 0)$, for $0 \leq i \leq k-1$;
$r_{28,i,j} \equiv (2, f_i\langle B_{\lceil \lg k \rceil+1}, j, i\rangle / f_{i+1}\langle B', j, i\rangle, 0)$, for $0 \leq i \leq k-2$, $0 \leq j \leq 1$;
$r_{29,j} \equiv (2, f_{k-1}\langle B_{\lceil \lg k \rceil+1}, j, k-1\rangle / \langle B', j, k-1\rangle, 0)$, for $0 \leq j \leq 1$;
$r_{30,i,j} \equiv (2, \langle B', j, i\rangle / \lambda, 3)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$;
$r_{31,i} \equiv (2, \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle \langle B_{\lceil \lg k \rceil+1}, 1, i\rangle e_i / \langle A_{\lceil \lg k \rceil+1}, 0, i\rangle$
$\langle B_{\lceil \lg k \rceil+1}, 1, i\rangle g_0, 0)$, for $0 \leq i \leq k-1$;
$r_{32,i,j} \equiv (2, g_i\langle A_{\lceil \lg k \rceil+1}, j, i\rangle / g_{i+1}\langle A', j, i\rangle, 0)$, for $0 \leq i \leq k-2$, $0 \leq j \leq 1$;

$r_{33,j} \equiv (2, g_{k-1}\langle A_{\lceil \lg k \rceil+1}, j, k-1\rangle / \langle A', j, k-1\rangle, 0)$, for $0 \leq j \leq 1$;
$r_{34,i,j} \equiv (2, \langle A', j, i\rangle / \lambda, 3)$, for $0 \leq i \leq k-1$, $0 \leq j \leq 1$.

- $\mathcal{E}(k) = \Gamma(k)$.
- $i_{in} = 2$ is the *input cell*.
- $i_{out} = 3$ is the *output cell*.

4.1 An Overview of the Computation

A family of tissue P systems with cell division is constructed as above. Let n be an instance of the prime factorization problem, where n is the integer number to be decomposed and k is the total number of binary bits to represent n. Then we consider a size mapping on the set of instances defined as $s(u) = k$. The coding of the instance is the multiset $cod(u) = \langle n, i_{k-1}, k-1\rangle\langle n, i_{k-2}, k-2\rangle \cdots \langle n, i_0, 0\rangle$, where $i_j = 0$ or 1 $(0 \leq j \leq k-1)$ is the bit at position j in the binary encoding of n. In what follows, we will informally describe how the tissue P system with cell division $\Pi(s(u))$ with input $cod(u)$ works.

Let us start with the generation stage. This stage has two parallel processes, which is described in two items.

- On one hand, in the cell with label 1 by using the rule $r_{3,i}$ the object c_i is multiplied until step $2k$; starting from c_0 object c_i grows its subscript by one in each step. Therefore, 4^k copies of c_{2k} are obtained in the cell with label 1 at step $2k$.
- On the other hand, in the cell with label 2 the division rules $r_{1,i}$ and $r_{2,i}$ are applied. For each object a_i (which is used to generate the two possible bits at position i in the binary encoding of integer number A), two cells labeled by 2 are produced, one of them containing a new object $\langle A, 0, i\rangle$ and the other one containing another new object $\langle A, 1, i\rangle$. Object $\langle A, 0, i\rangle$ (resp. $\langle A, 1, i\rangle$) represents the fact that the bit at position i in the binary encoding of A is 0 (resp. 1). Similarly, for each object b_i (which is used to generate the two possible bits at position i in the binary encoding of integer number B), two cells labeled by 2 are also produced, one of them containing a new object $\langle B, 0, i\rangle$ and the other one containing another new object $\langle B, 1, i\rangle$. The objects a_i, b_i are non-deterministically chosen, after $2k$ steps of division we obtain exactly 4^k cells with label 2, each of them encoding one possible pair of integer numbers A and B whose values range from 0 to $2^k - 1$. The object z is duplicated, hence a copy of z appears in each cell with label 2. Note that after step $2k$ the cells with label 2 cannot divide any more, because the objects a_i and b_i are exhausted.

The pre-checking stage starts from step $2k + 1$, in this stage, the product of each pair of integer numbers in cell with label 2 is calculated. At the step $2k + 1$, there are 4^k copies of c_{2k} in the cell with label 1, and there are 4^k cells with label 2, each of them containing a copy of z, so the rule r_4 is enabled and applied.

Due to the maximality of the parallelism of using the rule r_4, each cell with label 2 gets precisely one copy of c_{2k}. In the next $\lceil \lg 2k \rceil$ steps, by the rule $r_{5,i}$, the object c_{2k+i} is duplicated and its subscript increases by one in each step; so at step $2k + \lceil \lg 2k \rceil + 1$, there are at least $2k$ copies of $c_{2k+\lceil \lg 2k \rceil}$ in each cell with label 2. Once object $c_{2k+\lceil \lg 2k \rceil}$ is generated, by the rules $r_{6,i,j} - r_{7,i,j}$, each copy of objects $\langle A, j, i \rangle$ and $\langle B, j, i \rangle$, $0 \leq i \leq k-1$, $0 \leq j \leq 1$, together with a copy of object $c_{2k+\lceil \lg 2k \rceil}$, is traded for one copy of objects $\langle A_0, j, i \rangle$, $\langle B_0, j, i \rangle$, and one copy of object $c_{2k+\lceil \lg 2k \rceil+1}$ at step $2k + \lceil \lg 2k \rceil + 2$.

From step $2k + \lceil \lg 2k \rceil + 3$ to step $2k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 3$, by the rules $r_{11,i,j,l}$ and $r_{12,i,j,l}$, the objects $\langle A_j, l, i \rangle$ and $\langle B_j, l, i \rangle$ duplicate themselves until getting at least $k+1$ copies of objects $\langle A_{\lceil \lg k \rceil+1}, l, i \rangle$ and $\langle B_{\lceil \lg k \rceil+1}, l, i \rangle$. In the following computation, k copies of these objects are used to obtain the product of integer numbers A and B, the other one copy of these objects is used to output the computing result.

For any two k-bits integer numbers $A = \sum_{i=0}^{k-1} x_i 2^i$ ($x_i = 0$ or 1) and $B = \sum_{i=0}^{k-1} y_i 2^i$ ($y_i = 0$ or 1), the product of A and B can be written as $A \times B = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} x_i y_j 2^{i+j}$. In order to get the product, we will first compute the contribution of each pair of bits x_i and y_j, and then sum all the contributions. Specifically, the rules $r_{13,i,j} - r_{16,i,j}$ are used to get the contribution of each pair of bits; the rules $r_{17,i} - r_{19,i}$ are used to get the sum of all contributions. Since each cell with label 2 contains at least $k+1$ copies of objects $\langle A_{\lceil \lg k \rceil+1}, j, i \rangle$ and $\langle B_{\lceil \lg k \rceil+1}, j, i \rangle$, $j = 1$ or 0, the process of computing the product of each pair of bits a_i and b_j only needs one step, which produces some bits of the result C as well as the carry bits. It takes at most $2k$ steps to sum all the bits by the rules $r_{17,i} - r_{19,i}$. In this way, after step $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 4$ the product of two integer numbers in each cell with label 2 is computed and the pre-checking stage is finished. At this moment, the subscript of object c_j reaches $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 4$ by the rules r_8 and $r_{9,i}$, while the subscript of object c'_i reaches $\lceil \lg k \rceil + 2k + 3$ by the rule $r_{10,i}$.

The checking stage starts from step $4k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 5$ with the application of the rules $r_{20,i,j}$, $r_{21,i,j}$, r_{22} and $r_{23,i}$. The rules $r_{20,i,j}$ are used to check whether the "efficient" length of bits of the product is greater than $k-1$, thus whether there exists at least one position i, $k \leq i \leq 2k-2$, in the product having bit 1. If there exists such position in the binary encoding of the product, then the product must be greater than n. In this case, at least one of objects $\langle C, 1, i \rangle$, $k \leq i \leq 2k-2$, must appear in this cell. By the rule $r_{20,i,j}$, objects $\langle n, j, k-1 \rangle$, $j = 0$ or 1, are removed. The rules $r_{21,j,k-1}$ and r_{20} cannot be used without object $\langle n, j, k-1 \rangle$, thus this cell with label 2 will not send objects to the output cell. If the bit on each position i such that $k \leq i \leq 2k-2$ equals to 0, then at that step only the rule can be used by which object $c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+4}$ is traded for $c_{4k+\lceil \lg 2k \rceil+\lceil \lg k \rceil+5}$. The rues $r_{21,i,j}$, r_{22} and $r_{23,i}$ are used to check whether the product equals to n.

- If *the product equals to* n in a cell with label 2, then all objects $\langle X, 0, i \rangle$, $0 \leq i \leq k-1$, should be produced in this cell by the rule $r_{21,i,j}$. The rules r_{22} and $r_{23,i}$ are used to check whether all objects $\langle X, 0, i \rangle$, $0 \leq i \leq k-1$, are

produced. It is performed one bit by one bit, starting from the most significant bit. The object $\langle X, 0, k-1\rangle$ is traded for d_{k-2}, then d_{k-2} and $\langle X, 0, k-2\rangle$ are traded for d_{k-3}, the process continues until the position 0 is checked and the object d_{-1} is produced. Since the length of the integer number n is k, this process takes k steps. The computation passes to the output stage from step $5k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 7$.

- If *the product does not equal to* n in a cell with label 2, then at least one object $\langle X, 0, i\rangle$, $0 \leq i \leq k-1$, does not be produced in this cell. Without loss of generality, we assume that the first object without appearing in this cell is $\langle X, 0, s\rangle$, starting from the most significant bit, where $0 \leq s \leq k-1$. By the rules r_{22} and $r_{23,i}$, it is not difficult to find that object d_{s-1} will not be produced and the computation of the system halts at that moment. That means that this cell will not send any objects to the output cell.

The output stage starts from step $5k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 7$. In this stage, if the product of two integer numbers equals to n and the two numbers are also equal, then one of them will be outputted to the output cell; if the product of two integer numbers equals to n and the two numbers are not equal, then the smaller one will be outputted to the output cell. According to the checking stage, if the product of two integer numbers equals to n in a cell with label 2, then the object d_{-1} appears in this cell. The object d_{-1} is traded for e_{k-1}. The rules $r_{25,i,j}$, r_{26} and $r_{31,i}$ are used to check which integer number is smaller or whether they are equal. If object e_{-1} appears, then it means the fact that two integer numbers are equal, and the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell labeled by 3 by the rules r_{26}, $r_{28,i,j}$, $r_{29,j}$, and $r_{30,i,j}$. If two integer numbers are not equal, then object f_0 or g_0 should appear and object e_{-1} does not appear. If object f_0 appears in the cell with label 2, it means the fact that the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is greater than the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$, and the integer number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell with label 3 by the rules r_{26}, $r_{28,i,j}$, $r_{29,j}$, and $r_{30,i,j}$. If object g_0 appears in the cell with label 2, it means the fact that the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is less than the number corresponding to objects $\langle B_{\lceil \lg k \rceil+1}, j, i\rangle$, and the integer number corresponding to objects $\langle A_{\lceil \lg k \rceil+1}, j, i\rangle$ is outputted to the output cell with label 3 by the rules $r_{32,i,j}$ $r_{33,j}$, and $r_{34,i,j}$. This stage takes not more than $2k+3$ steps, and in the case that two integer numbers are equal, the output stage takes exactly $2k + 3$ steps. So the computation of the system stops after step $7k + \lceil \lg 2k \rceil + \lceil \lg k \rceil + 9$, and the computation result can read out from the objects in the output cell with label 3.

4.2 A Simple Example

In order to show how the tissue P systems with cell division constructed in Section 4.1 work, let us consider the factorization of integer number 2. Hence, we have $n = 2$ and $k = 2$. The initial configuration of tissue P system with cell division

$\Pi(2)$ for the factorization of integer number 2 is illustrated in Figure 1. From the figure, it can be found that 2 is represented by objects $\langle n, 1, 1\rangle$ and $\langle n, 0, 0\rangle$.

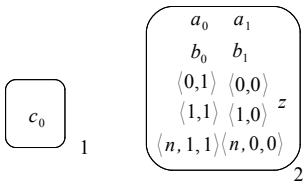

Fig. 1. The initial configuration of system $\Pi(2)$ for factoring integer number 2

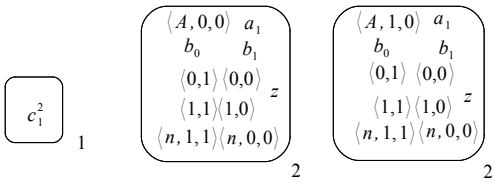

Fig. 2. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 1

At step 1, both the cell with label 1 and the cell with label 2 have rules which can be used. In the cell with label 1, by the rule $r_{3,0}$ object c_0 evolves to c_1 and its number is doubled; in the cell with label 2, objects a_0, a_1, b_0 and b_1 are non-deterministically chosen to divide this cell. Without loss of generality, we assume that a_0 is used to divide the cell with label 2. Object a_0 is consumed and two objects $\langle A, 0, 0\rangle$ and $\langle A, 1, 0\rangle$ are generated, with one object appearing in a cell with label 2 and another one appearing in the other cell with label 2. The other objects in the cell with label 2 are duplicated in the two new cell with label 2. The configuration of the system at this step is shown in Figure 2.

Similar to the work of object a_0, objects a_1, b_0 and b_1 can continue to divide the cells with label 2 in the following three steps, with one object dividing its corresponding cell one time. At the same time, in cell with label 1 the number of object c_4 becomes 16. The configuration of the system at step 4 is shown in Figure 3. Note that at this moment all pairs of integer numbers of length 2 are generated, with one cell with label 2 containing a pair.

After step 4, the system enters to the pre-checking stage. In this stage, the product of each pair of integer numbers is calculated. This process is done in parallel in the cells with label 2. The product of each pair of integer numbers is represented by objects $\langle C, i, j\rangle$, $i = 0$ or 1, $0 \leq j \leq 3$. Figure 4 gives the configuration of the system when the pre-checking stage is finished. The pre-checking stage finishes at step 15.

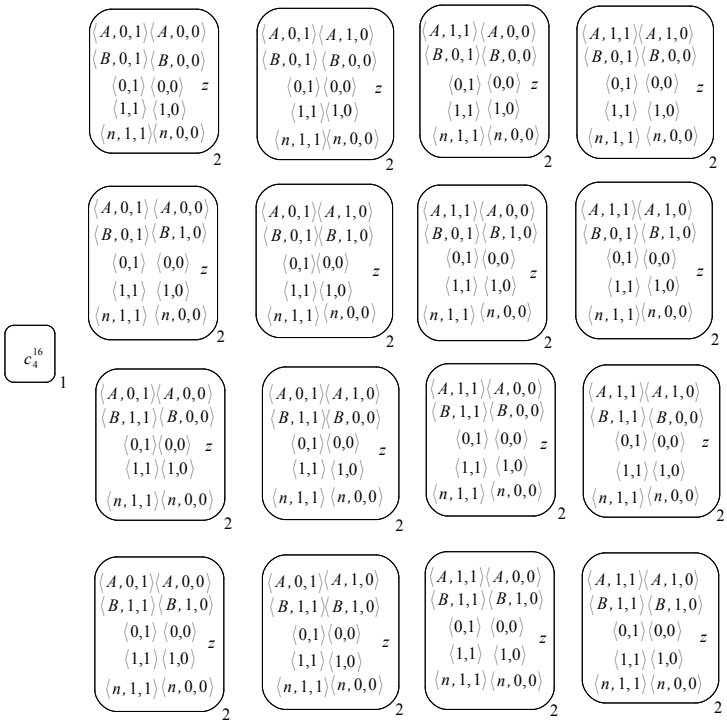

Fig. 3. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 4

The checking stage starts at step 16. In this stage, the system compares each product represented by objects $\langle C, i, j\rangle$, $i = 0$ or 1, $0 \leq j \leq 3$, with the integer number 2 represented by objects $\langle n, 1, 1\rangle$ and $\langle n, 0, 0\rangle$. If there exists at least one position i, $2 \leq i \leq 3$, at which the bit of a product equals to 1 (that is, there exists object $\langle C, 1, 2\rangle$ or $\langle C, 1, 2\rangle$), then this object together with objects $\langle n, 1, 1\rangle$ and c'_8 is removed from the corresponding cell with label 2. If the product equals to the integer number 2, then both object $\langle X, 0, 1\rangle$ and object $\langle X, 0, 0\rangle$ will appear in the corresponding cell with label 2. Figure 5 gives the configuration of the system when the checking stage is finished.

The output stage starts at step 20. In this stage, each cell with label 2 which contains the product that equals to the integer number 2, outputs the integer number that is not greater than another one. Such integer number is represented by objects of the form $\langle A', i, j\rangle$ or $\langle B', i, j\rangle$, $i, j = 0, 1$. The configuration of the system at step 25 is shown in Figure 6. From Figure 6, it is not difficult to find that, among 16 cells with label 2 there are two cells having objects of those forms. In a cell with label 2 the objects are $\langle A', 0, 1\rangle$, $\langle A', 1, 0\rangle$, in another cell with label 2 the objects are $\langle B', 0, 1\rangle$, $\langle B', 1, 0\rangle$. These objects will be sent to the output cell from cells with label 2. In fact, they represent the same integer number 1.

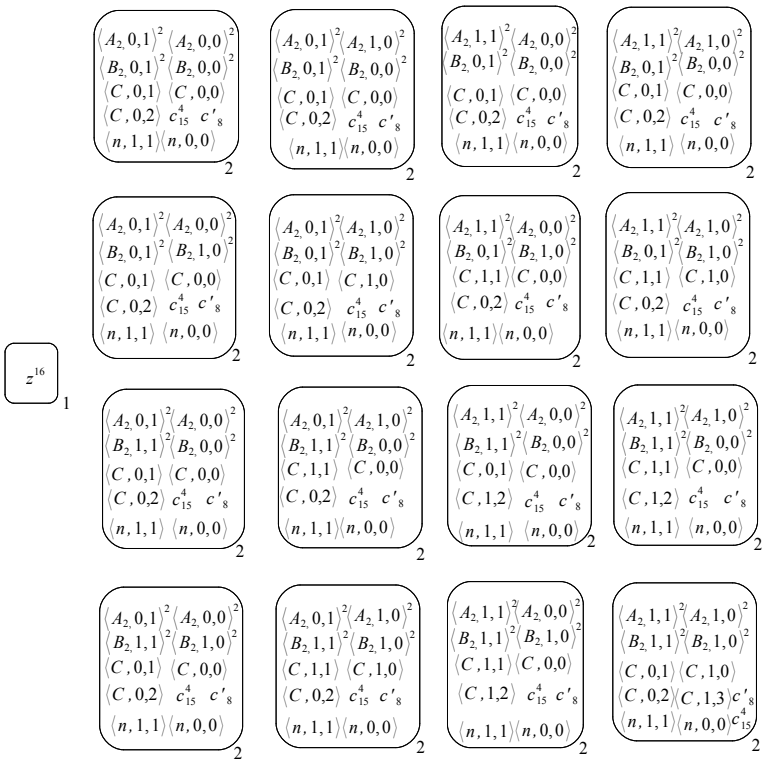

Fig. 4. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 15

4.3 Necessary Resources

From the overview of the computation, it can be found that the family $\{\Pi(k)\}_{k\in\mathbb{N}}$ constructed above can solve the factorization problem in a linear time with respect to the size of the integer to be factored. In what follows, we point out this family of tissue P systems with cell division can be constructed in polynomial time by deterministic Turing machine.

It is easy to check that the rules of a system $\Pi(k)$ of the family are defined recursively from the value k. The necessary resources to build an element of the family are of a polynomial order, as shown below:

- Size of the alphabet: $k^2 + 35k + (4k + 2)\lceil \lg k \rceil + \lceil \lg 2k \rceil + 11 \in O(k^2)$.
- Initial number of cells: $3 \in O(1)$.
- Initial number of objects: $k^2 + 2k + 2 \in O(k^2)$.
- Number of rules: $4k^2 + 39k + \lceil \lg 2k \rceil + (4k + 2)\lceil \lg k \rceil + 4 \in O(k^2)$.
- Maximal length of a rule: $6 \in O(1)$.

Therefore, a deterministic Turing machine can build the tissue P system $\Pi(k)$ in a polynomial time with respect to k.

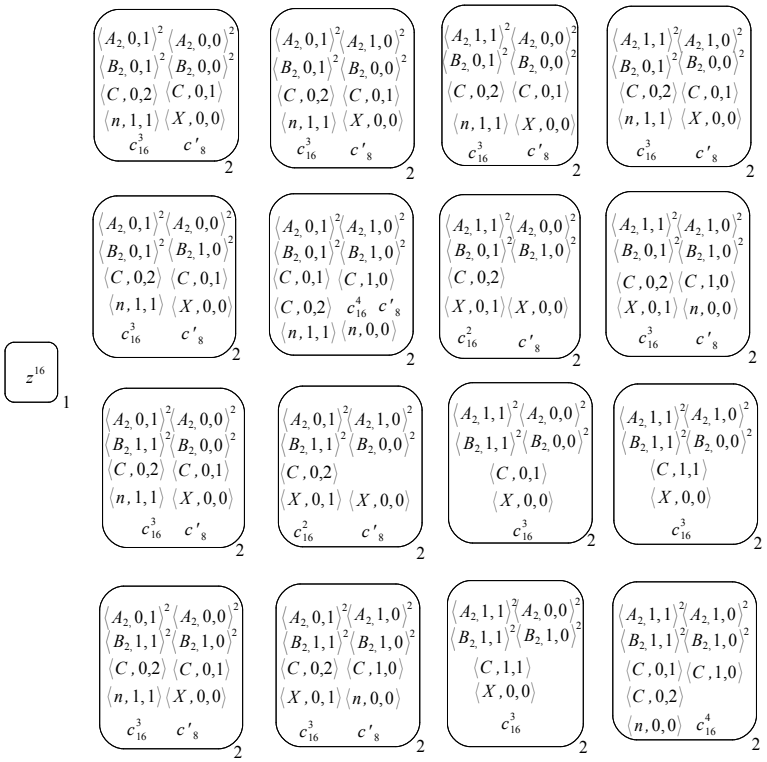

Fig. 5. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 19

5 Conclusions and Comments

Prime factorization problem is not in itself widely useful problem. It has become useful only because it has been found to be crucial for public-key cryptography, and this application is in turn possible only because they have been presumed to be difficult. Currently, no deterministic polynomial-time algorithm is known, which can be executed on Turing machines, that solves the problem for every possible instance. It is of interest to explore any possible and reasonable way to solve prime factorization problem because of its importance in public-key cryptography.

Prime factorization problem is neither decision problem nor optimization problem. In this work, it is considered as a function problem, and in the framework of tissue P systems with cell division, a linear-time solution to prime factorization problem is given. The initial structure of the systems is very simple, which consists of three cells. The system is initialized with inputting into the fixed input cell the multiset that expresses the integer number n to be factored. After a linear time with respect to the size of n (i. e., $\lfloor \lg k \rfloor + 1$), we can read out one factor of n in the output cell.

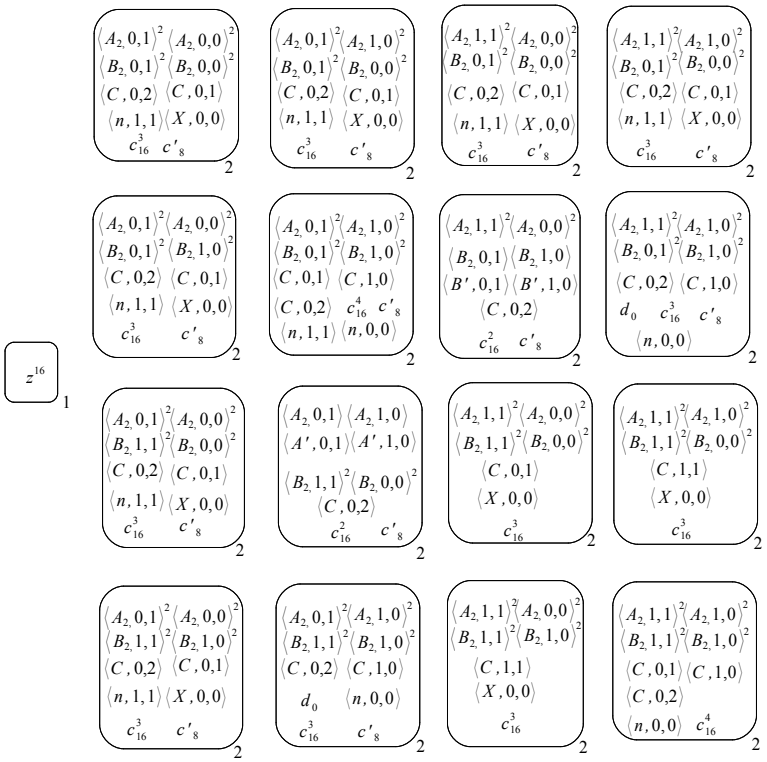

Fig. 6. The configuration of system $\Pi(2)$ for factoring integer number 2 at step 25

P system is a highly distributed parallel model of computation. Currently, nobody knows how to build a biochemical computer/an artificial tissue-like computer. P systems may be implemented using molecules, cells or a large computer network such as the Internet. Although it goes beyond the scope of this work to discuss the implementation of P systems, clearly, it is of particular interest and it is a big challenging topic.

Acknowledgements

The work was supported by National Natural Science Foundation of China (61033003, 61003038 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), Fundamental Research Funds for the Central Universities (2010ZD001), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). Mario J. Pérez-Jiménez also acknowledges the support of the project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the "Proyecto de Excelencia con Investigador de Reconocida Valía" of the Junta de Andalucía under grant P08-TIC04200.

References

1. M. Agrawal, N. Kayal, N. Saxena, PRIMES is in P, *Annals of Mathematics* 160(2) (2004) 781–793.
2. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, A uniform family of tissue P system with cell division solving 3-COL in a linear time, *Theoretical Computer Science* 404 (2008) 76–87.
3. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Solving subset sum in linear time by using tissue P system with cell division, in: *Lecture Notes in Computer Science*, vol. 4527, 2007, pp. 170–179.
4. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Computational efficiency of cellular division in tissue-like membrane systems, *Romanian Journal of Information Science and Technology* 11 (3) (2008) 229–241.
5. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.A. Pérez-Jiménez, A. Riscos-Núñez, Solving the partition problem by using tissue-like P systems with cell division, in: D. Kearney, V. Nguyen, G. Gioiosa, T. Hendtlass (Eds.), Third International Conference on Bio-Inspired Computing: Theories and Applications, Adelaide, 2008, pp. 43-48.
6. A. Leporati, C. Zandron, G. Mauri, Solving the factorization problem with P systems, *Progress in Natural Science*, 17 (4) (2007) 471–478.
7. A. Leporati, C. Zandron, M.A. Gutiérrez-Naranjo, P systems with input in binary form, *International Journal of Foundation of Computer Science*, 17(1) (2006) 127–146.
8. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón, A new class of symbolic abstract neural nets: tissue P systems, in: *Lecture Notes in Computer Science*, vol. 2387, 2002, pp. 290–299.
9. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón, Tissue P systems, *Theoretical Computer Science* 296 (2003) 295–326.
10. A. Obtulowicz, On P systems with active membranes solving the integer factorization problem in a polynomial time, in: *Lecture Notes in Computer Science*, vol. 2235, 2001, pp. 267–285.
11. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, *New Generation Computing* 20 (3) (2002) 295–305.
12. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences* 61(1) (2000) 108–143.
13. Gh. Păun, *Membrane Computing. An Introduction*, Springer–Verlag, Berlin, 2002.
14. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Tissue P system with cell division, *International Journal of Computers, Communications & Control* III (3) (2008) 295–302
15. R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comunications of the ACM* 21 (2) (2006) 120–126.
16. P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Journal on Computing* 26 (5) (1997) 1484–1509
17. P systems web page `http://ppage.psystems.eu/`