P Systems with Gemmation of Mobile Membranes

Daniela BESOZZI*, Claudio ZANDRON**
Giancarlo MAURI**, Nicoletta SABADINI*

*Università degli Studi dell'Insubria
Via Valleggio 11, 22100 Como, Italy
E-mail:dbesozzi@tin.it, nicoletta.sabadini@uninsubria.it

*Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20136 Milano, Italy
E-mail: zandron/mauri@disco.unimib.it

Fax Number:+39 02 64487839

Abstract. P systems are computational models inspired by some biological features of the structure and the functioning of real cells. In this paper we introduce a new kind of communication between membranes, based upon the natural budding of vesicles in a cell. We define the operations of gemmation and fusion of mobile membranes, and we use membrane structures and rules over strings of biological inspiration only. We prove that P systems of this type can generate all recursively enumerable languages and, moreover, the Hamiltonian Path Problem can be solved in a quadratic time. Some open problems are also formulated.

Keywords: Molecular Computing, NP-Complete Problems, Recursively Enumerable Languages

Track: A - Algorithms, Automata, Complexity and Games

1 Introduction

The P systems were recently introduced in [6] as a class of distributed parallel computing devices of a biochemical type. The basic model consists of a membrane structure composed by several cell-membranes, hierarchically embedded in a main membrane called the skin membrane. The membranes delimit regions and can contain objects, which evolve according to given evolution rules associated with the regions. Such rules are applied in a maximally parallel manner: at each step, all the objects which can evolve should evolve.
A computation device is obtained: we start from an initial configuration and we let the system evolve. A computation halts when no further rule can be applied. The objects in a specified output membrane (or expelled through the skin membrane) are the result of the computation. Many variants are considered in [6], [7], and [8]. A survey and an up-to-date bibliography can be found at the web address `http://bioinformatics.bio.disco.unimib.it/psystems`.

Up to now, in all the variants of P systems only the direct communication of objects through membranes has been considered: an object can exit the membrane where it is placed and enter the upper level region, or it can enter a lower level region. Such communications are defined by means of target indications *in/out* attached to the evolution rules of the system. The aim of the present work is to introduce a new kind of communication between membranes and to keep the definition of P systems closer to the real structure of cells.
The notion of mobility was first introduced into P systems area in [9], where the creation of traveling cells was proposed in order to get secure communications between non-adjacent membranes. Here we introduce a different definition of *mobile membranes*, based upon a biological process of alive cells. Cellular membranes are selectively permeable to many small substances as water and ions, but not to bigger ones as proteins (see, e.g., [12]). Such substances are communicated inside or outside the cell by means of vesicles, which are little parts of a membrane, encased on their cytosolic face by a specific protein that causes their budding from the membrane. When the vesicle fuses with its target membrane, the carried proteins are introduced inside it, where they can be modified by other chemical reactions. Many cellular compartments use this kind of communication, in particular this is the case of the Golgi apparatus ([10]), a stack of distinct elementary membranes (i.e. membranes without other membranes inside) where, in sequence, many proteins are stepwise modified and then sent to another Golgi-region. Specifically, only the substances that have completed their "maturation path" inside the current region can be communicated by a vesicle to the next destination.
In order to simulate all these natural features, we consider P systems with simple membrane structures (the skin membrane can only contain elementary membranes) and with operations on strings of a biochemical inspiration, such as mutation, replication and splitting rules. Moreover, we define a *meta-priority* between the set of classical evolution rules and the set of *pre-dynamical rules*, which are the rules that give rise to the *gemmation* of mobile membranes (that is, the budding of vesicles in the cell). The meta-priority is needed to the aim

of simulating the completion of the maturation path of an object. After a predynamical rule has been used, the phases of gemmation and fusion of mobile membranes take place. In particular, the output of the system is due to the fusion of a mobile membrane with the skin membrane: this process causes the release of the objects outside the system. The set of strings that exit the skin membrane is the language generated by the system, as usual in P systems with external output ([8]).
We show that the obtained system is able to generate every recursively enumerable language and that it can be used to solve NP-complete problems in polynomial time. In particular, we prove this by showing how to solve the Hamiltonian Path Problem in a quadratic time with respect to the input length.

2 Definition

We refer to [11] for formal language theory prerequisites, we only mention here that we denote by V^* the free monoid generated by the alphabet V under the operation of concatenation. The empty string is denoted by λ and $V^+ = V^* - \{\lambda\}$ is the set of non-empty strings over V.

A membrane structure is a construct consisting of several membranes hierarchically embedded in a unique membrane, called a *skin membrane*. We identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. We can also associate a tree with the structure, in a natural way; the height of the tree is the *depth* of the structure itself.

A membrane identifies a region, delimited by it and the membrane immediately inside it. If we place multisets of objects in the region from a specified finite set V, we get a super-cell. A super-cell system (or P-system) is a super-cell provided with evolution rules for its objects.

We will work with string-objects, so with every region $i = 0, 1, \ldots, n$ of μ we associate a multiset of finite support over V^*, that is a map $M_i : V^* \to N$ where $M_i = \{(x_1, M_i(x_1)), \ldots, (x_p, M_i(x_p))\}$, for some $x_k \in V^+$ such that $M_i(x_k) > 0 \; \forall k = 1, \ldots, p$.
We will use three types of operations on strings over V, which were first considered in [1]:

1. *Mutation*: a *mutation rule* is a context-free rewriting rule $r_m : a \to u$, where $a \in V$ and $u \in V^*$. For strings $w_1, w_2 \in V^+$ we write $w_1 \Rightarrow_{r_m} w_2$ if $w_1 = x_1 a x_2$ and $w_2 = x_1 u x_2$, for some $x_1, x_2 \in V^*$.
2. *Replication*: if $a \in V$ and $u_1, u_2 \in V^+$, then $r_r : a \to u_1 \parallel u_2$ is called a *replication rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Rightarrow_{r_r} (w_2, w_3)$ (and we say that w_1 is replicated with respect to rule r_r) if $w_1 = x_1 a x_2$, $w_2 = x_1 u_1 x_2$ and $w_3 = x_1 u_2 x_2$ for some $x_1, x_2 \in V^*$.
3. *Splitting*: if $a \in V$ and $u_1, u_2 \in V^+$, then $r_s : a \to u_1 : u_2$ is called a *splitting rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Rightarrow_{r_s} (w_2, w_3)$ (and we say that w_1 is split with respect to rule r_s) if $w_1 = x_1 a x_2$, $w_2 = x_1 u_1$ and $w_3 = u_2 x_2$ for some $x_1, x_2 \in V^*$.

Note that only replication and splitting rules increase the number of strings, while mutation rules can delete symbols. When using such operations in P systems, we will add target indications to rules, indicating the regions where the resulting strings will be communicated at the next step.
With each region $i = 0, 1, \ldots, n$ of the membrane structure we associate a set of *classical* evolution rules C_i, that is a set of mutation, replication and splitting rules as defined above, where u, u_1, u_2 are strings over $(V \times \{here, out\})$ if $i = 1, \ldots, n$, or over $(V \times \{here, out\}) \cup (V \times \{in_1, \ldots, in_n\})$ if $i = 0$.
Moreover, for each inner region of μ we associate a set of *pre-dynamical* evolution rules D_i, that is a set of mutation, replication and splitting rules, where now $x_1 = \lambda$ (or $x_2 = \lambda$), u and at least one between u_1, u_2 are strings over $((\{@_j\} \cdot V^*) \times \{here\})$ (respectively $((V^* \cdot \{@_j\}) \times \{here\})$), where $@_j$ is a special symbol not in V and $j \in \{0, 1, \ldots, n\}, j \neq i$.
We point out that a pre-dynamical rule can introduce the special symbol $@_j$ *only* at the ends of the string, that is the reason why we ask for x_1 or x_2 to be empty. Note that if $i = 0$, then the set D_0 is empty, that is no pre-dynamical rule is defined inside the skin membrane.

Once the symbol $@_j$ as been introduced by a pre-dynamical rule in membrane i, for $j \neq i$, inside the P system we have two sequential and dynamical communication processes carried out by a *mobile membrane*, which we write as a couple of well-matching round brackets $(_{i,j} \ldots)_{i,j}$, where i is the label of the originating membrane and j is the label of the target membrane. The communication steps are defined by means of the following rules:

1. *Gemmation* of a mobile membrane:

$$[_0 \ldots [_i w @_j]_i \ldots]_0 \rightarrow_G [_0 \ldots [_i]_i (_{i,j} w)_{i,j} \ldots]_0$$

for some $i \in \{1, \ldots, n\}, j \in \{0, 1, \ldots, n\}, j \neq i, w \in V^+$.
During this first phase the symbol $@_j$ is removed, its subscript becomes the second label of the mobile membrane, the string w leaves membrane i and enters the freshly created mobile membrane.
If there are more than one string as $w_1@_j, \ldots, w_k@_j$ inside membrane i, all of which directed to the same target membrane j, then a single common mobile membrane will be budded off from membrane i:

$$[_0 \ldots [_i w_1 @_j, \ldots, w_k @_j]_i \ldots]_0 \rightarrow_G [_0 \ldots [_i]_i (_{i,j} w_1, \ldots, w_k)_{i,j} \ldots]_0.$$

Otherwise, if inside membrane i there are strings $w_1@_{j_1}, \ldots, w_h@_{j_k}$ $(h \geq k)$ such that $j_1, \ldots, j_h$ are pairly distinct, then k distinct mobile membranes will be gemmated, each one containing the strings directed to the specified membrane:

$$[_0 \ldots [_i w_1 @_{j_1}, \ldots, w_{h_1} @_{j_1}, \ldots, w_{h_k} @_{j_k}, \ldots, w_h @_{j_k}]_i \ldots]_0 \rightarrow_G$$

$$[_0 \ldots [_i]_i (_{i,j_1} w_1, \ldots, w_{h_1})_{i,j_1} \ldots (_{i,j_k} w_{h_k}, \ldots, w_h)_{i,j_k} \ldots]_0.$$

2. *Fusion* of the mobile membrane:

$$[_0 \ldots (_{i,j} w)_{i,j} [_j]_j \ldots]_0 \rightarrow_F [_0 \ldots [_j w]_j \ldots]_0$$

for some $i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}, j \neq i, w \in V^+$.
During this second phase the mobile membrane becomes a part of the target membrane, leaving its contents inside it.
In particular, if $j = 0$ the mobile membrane fuses with the skin membrane and the objects exit the system. In this way we simulate the biological process of exocytosis and hence we have the (external) output of the string:

$$[_0 \ldots (_{i,0} w)_{i,0} \ldots]_0 \rightarrow_F [_0 \ldots]_0 w.$$

In order to stay close to the structure of a real cell, we will introduce two further limitations:
1. We consider membrane structures μ of depth 2, with the skin membrane always labelled with 0 and the inner membranes injectively labelled with numbers in the set $\{1, \ldots, n\}$;
2. We do not define any priority relation between rules in the set C_i neither between rules in the set D_i. Instead, we define a *meta-priority* between the whole set C_i and the whole set D_i, $\forall i = 1, \ldots, n$, meaning that all applicable classical rules in C_i must be used before any other applicable pre-dynamical rule in D_i.

Finally, a *P system* (of degree $n + 1$) with *gemmation of mobile membranes* is defined by the construct

$$\Pi_{(G,F)} = (V, T, \mu, M_0, \ldots, M_n, (C_0, \emptyset), (C_1, D_1), \ldots, (C_n, D_n), \infty)$$

with the following components:

- (i) V is an alphabet such that $V \cap \{@_j\} = \emptyset, \forall j = 0, 1, \ldots, n$;
- (ii) $T \subseteq V$ is the output alphabet;
- (iii) $\mu = [_0 [_1]_1 [_2]_2 \ldots [_{n-1}]_{n-1} [_n]_n]_0$ is a membrane structure of depth 2 and degree $n + 1$;
- (iv) $M_0, \ldots, M_n$ are multisets of finite support over V^*;
- (v) (C_i, D_i) $\forall\, i = 0, 1, \ldots, n$ are the set of classical evolution rules and the set of pre-dynamical evolution rules, respectively. C_i has a meta-priority above D_i as far as the application of all of their rules is concerned, $\forall\, i = 1, \ldots, n$. The set D_0 is empty;
- (vi) ∞ means that the system has external output.

The application of the rules is done as usual in P system area, following some specific principles: the rules are applied in a maximally parallel manner (each string which can evolve should evolve). At each step of computation, a string can be processed by one rule only, and its multiplicity is decreased by one. The multiplicity of strings produced by an operation is accordingly increased. The strings resulting after the application of a rule are communicated by mobile

membranes to the regions specified by the target indications.
The membrane structure at a given time, together with all multisets of objects associated with the regions defined by the membrane structure, is the *configuration* of the system at that time. The $(n+1)$-tuple $(\mu, M_1, \ldots, M_n)$ constitute the *initial configuration* of the system. We have a *transition* from a configuration to another one by applying the rules present in the sets (C_i, D_i), $0 \leq i \leq n$. A sequence of transitions forms a *computation*. A computation *halts* when there is no rule which can be applied in the current configuration. The *output* is the set of strings over T sent out of the system during the computation. A non-halting computation provides no output. The language generated in this way by a P system Π is denoted by $L(\Pi)$.

3 Computational completeness

We show that P systems with gemmation of mobile membranes and in/out communications are able to generate any recursively enumerable language.
In the proof we need the notion of a *matrix grammar with appearance checking*; such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets of nonterminal and terminal symbols, $S \in N$ is the axiom, M is a finite set of matrices, which are sequences of context-free rules of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M.
For $w, z \in (N \cup T)^*$ we write $w \Rightarrow z$ if there are a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – one says that these rules are applied in the *appearance checking* mode). The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When $F = \emptyset$ (hence we do not use the appearance checking feature), the generated family is denoted by MAT.
A matrix grammar with appearance checking $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \dagger\}$ is the union of mutually disjoint sets, and the matrices in M are of one of the following forms:

1. $(S \to XA)$ with $X \in N_1, A \in N_2$;
2. $(X \to Y, A \to x)$ with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$;
3. $(X \to Y, A \to \dagger)$ with $X, Y \in N_1, A \in N_2$;
4. $(X \to \lambda, A \to x)$ with $X \in N_1, A \in N_2, x \in T^*$.

Moreover, there exists only one matrix of type 1, F exactly consists of all rules $A \to \dagger$ appearing in matrices of type 3 and $\dagger$ is a trap-symbol (once introduced, it can never be removed). Finally, each matrix of type 4 is used only once, at the last step of a derivation. According to Lemma 1.3.7 in [2], for each matrix grammar there exists an equivalent one in the binary normal form.

We denote by CF and RE the families of context free and recursively enumerable languages. The following proper inclusions hold: $CF \subset MAT \subset MAT_{ac} = RE$. Further details about matrix grammars can be found in [11] and [4]. Moreover, in [4] it is shown that all one-letter languages in MAT are regular.
We denote by $GP_n(MPri, (in/out))$ the family of languages generated by gemmating P systems of degree n, for $n \geq 1$, with relation of meta-priority and communications of type in/out. If the number of membranes is not limited, then the subscript n is replaced by $*$.

Theorem 1. *$GP_*(MPri, (in/out)) = RE$.*

Proof. The inclusion $GP_*(MPri, (in/out)) \subseteq RE$ directly follows from Church-Turing thesis. So, we only have to prove the opposite inclusion; to this aim, we make use of the equality $RE = MAT_{ac}$ and we consider a matrix grammar with appearance checking $G = (N, T, S, M, F)$, in the binary normal form previously described. Let p be the number of matrices of type 2 in G, q the number of matrices of type 3 and r the number of matrices of type 4, with $p \geq 1, q \geq 1, r \geq 1$.
We show how to construct a gemmating P system of degree $s = 2p + q + 2r + 3$ that generates the same language as G:

$$\begin{aligned}\Pi_{(G,F)} = (&V, \mu, M_0, M_1, M_{(2)_1}, M_{(2)_2}, \ldots, M_{(p+1)_1}, M_{(p+1)_2}, M_{(p+2)_1}, \ldots,\\ &M_{(p+q+1)_1}, M_{(p+q+2)_1}, M_{(p+q+2)_2}, \ldots, M_{(p+q+r+1)_1}, M_{(p+q+r+1)_2}, M_c,\\ &C_0, (C_1, D_1), (C_{(2)_1}, D_{(2)_1}), \ldots, (C_{(p+q+r+1)_2}, D_{(p+q+r+1)_2}), (C_c, D_c))\end{aligned}$$

with

$V = N_1 \cup N_2 \cup T \cup \{P, J, J', \sharp\}$
(we use all the symbols of G plus four support-symbols P, J, J' and $\sharp$);

$\mu = [_0\ [_1]_1\ [_{(2)_1}\ ]_{(2)_1}\ [_{(2)_2}\ ]_{(2)_2}\ \cdots\ [_{(p+1)_1}\ ]_{(p+1)_1}\ [_{(p+1)_2}\ ]_{(p+1)_2}\ [_{(p+2)_1}\ ]_{(p+2)_1}$
$\cdots\ [_{(p+q+1)_1}\ ]_{(p+q+1)_1}\ [_{(p+q+2)_1}\ ]_{(p+q+2)_1}\ [_{(p+q+2)_2}\ ]_{(p+q+2)_2}\ \cdots$
$[_{(p+q+r+1)_1}\ ]_{(p+q+r+1)_1}\ [_{(p+q+r+1)_2}\ ]_{(p+q+r+1)_2}\ [_c\ ]_c\ ]_0$
(membrane 1 simulates the matrix of type 1 in G, each couple of membranes labelled with $(i)_1, (i)_2, 2 \leq i \leq p+1$, simulate a matrix of type 2 in G, each membrane labelled with $(j)_1, p+2 \leq j \leq p+q+1$, simulate a matrix of type 3 in G, each couple of membranes labelled with $(k)_1, (k)_2, p+q+2 \leq k \leq p+q+r+1$, simulate a matrix of type 4 in G; we also use a control-membrane labelled with c);

$M_1 = \{XAP \mid S \to XA$ is the rule of the matrix of type $1\}$, all other multisets are empty;

$C_0 = \{J \to (\lambda, in_1), J' \to (\lambda, in_c)\}$
(the skin membrane contains two classical rules, one for each support-symbol J, J', which redirect the received strings to membrane 1 or to the control-membrane);

$C_1 = \emptyset$ and $D_1 = \{P \to P@_{(i)_1}, P \to P@_{(j)_1}, P \to P@_{(k)_1}\}$,
$\forall\, i = 2, \ldots, p+1, \forall\, j = p+2, \ldots, p+q+1, \forall\, k = p+q+2, \ldots, p+q+r+1$
(membrane 1 sends the current string to the first membrane $(i)_1, (k)_1$ of

any couple of membranes simulating matrices of type 2 and 4, or to any of the single membranes $(j)_1$ simulating the matrices of type 3);

$\forall\, i = 2, \ldots, p+1$ we define:

$C_{(i)_1} = \emptyset$ and $D_{(i)_1} = \{X \to @_{(i)_2} Y\}$
(for each matrix of type 2, in the first membrane we simulate the first rule of the matrix with one pre-dynamical rule),
$C_{(i)_2} = \{A \to (Jx, out)\}$ and $D_{(i)_2} = \emptyset$
(for each matrix of type 2, in the second membrane we simulate the second rule of the matrix with one classical rule, which introduces the support-symbol J in the string);

$\forall\, j = p+2, \ldots, p+q+1$ we define:

$C_{(j)_1} = \{A \to A\}$ and $D_{(j)_1} = \{X \to @_1 Y\}$
(for each matrix of type 3, in a unique membrane we simulate the second rule of the matrix by one classical evolution rule, and the first rule of the matrix by one pre-dynamical rule);

$\forall\, k = p+q+2, \ldots, p+q+r+1$ we define:

$C_{(k)_1} = \emptyset$ and $D_{(k)_1} = \{X \to @_{(k)_2} \lambda\}$
(for each matrix of type 4, in the first membrane we simulate the first rule of the matrix with one pre-dynamical rule),
$C_{(k)_2} = \{A \to (J'x, out)\}$ and $D_{(k)_2} = \emptyset$
(for each matrix of type 4, in the second membrane we simulate the second rule of the matrix with one classical rule. The support-symbol J' is introduced in the string);

and finally

$C_c = \{N_h \to (\sharp, here), \sharp \to (\sharp, here)\}\ \forall\, h = 1, \ldots, \mid N_1 \cup N_2 \mid$
(in the control-membrane we define a classical rule for each nonterminal symbol in N, and one classical mutation rule over $\sharp$ which causes the non termination of a computation),
$D_c = \{P \to \lambda @_0\}$
(this pre-dynamical rule erases the support-symbol P and sends the string outside the system).

The system works as follows: consider the string ZwP in membrane 1 with $Z \in N_1$ and $w \in (N_2 \cup T)^*$. Initially we have $Z = X$ and $w = A$.
We nondeterministically choose between any of the pre-dynamical rules defined in membrane 1, the string ZwP is so rewritten as $ZwP@_t$, with $t \in \{(i)_1, (j)_1, (k)_1\}$. This string is sent to membrane $(i)_1$, with $2 \leq i \leq p+1$, or $(k)_1$, with $p+q+2 \leq k \leq p+q+r+1$, or $(j)_1$, with $p+2 \leq j \leq p+q+1$, where we simulate the first rule of the matrices of type 2 and 4, or both rules of matrices of type 3, respectively.
If the string ZwP enters a membrane $(i)_1$, for any $i = 2, \ldots, p+1$, and $Z = X$,

then we can apply the pre-dynamical rule $X \to @_{(i)_2} Y$: the rule of the corresponding matrix is correctly simulated and the string enters the second membrane $(i)_2$. On the contrary, if $Z \neq X$, then the rule cannot be applied and the computation halts, no string will be generated. In the first case, when the string YwP enters membrane $(i)_2$, if the symbol $A \in w$ then we can apply the rule $A \to (Jx, out)$. The string Yw_1Jxw_2P, with $w_1, w_2 \in (N_2 \cup T)^*$ such that $w = w_1Aw_2$, enters membrane 0 and then it returns to membrane 1 by means of the rule $J \to (\lambda, in_1)$. Observe that the support-symbol J is immediately erased and it will never appear in any terminal string. From membrane 1 we can now start the simulation of another matrix. In membrane $(i)_2$, if the symbol $A \notin w$, then the string will never exit the current membrane, the computation halts and no string will be generated. Thus, with two membranes we are able to simulate the productions of any matrix of type 2, and we can correctly do it.

If the string ZwP enters a membrane $(j)_1$, for any $j = p+2, \ldots, p+q+1$, and $A \in w$, then the computation will never stop: the rule $A \to A$ will be applied forever because of the meta-priority relation. No string will be generated, thus we correctly simulate the introduction of the symbol † in a production of G. On the contrary, if $A \notin w$, then the classical rule $A \to A$ cannot be applied and we pass to the pre-dynamical rule $X \to @_1 Y$. If $Z = X$ then the string YwP will be sent to membrane 1, otherwise the rule cannot be applied and the computation stops. Thus we only need one membrane for every matrix in G whose rules are to be applied in the appearance checking mode. Observe that the order of the rules in the membrane is opposite to the order of the rules in the matrix, but this fact does not change the set of generated strings.

As it can be easily seen, in any couple of membranes $(k)_1, (k)_2$, for $k = p+q+2, \ldots, p+q+r+1$, the application of rules is similar to the one performed inside membranes $(i)_1, (i)_2$. The only difference is that in these membranes we erase the nonterminal symbols in N_1 and we use the support symbol J'. As for matrices of type 2, we are therefore able to simulate the productions of any matrix of type 4 with two membranes, and we can do it in the correct order.

When a string reaches a membrane labelled with $(k)_2$, for any $k = p+q+2, \ldots, p+q+r+1$, the simulation of the matrices of G has to be ended. To this aim, we make use of the support-symbol J': in membrane 0 we define the rule $J' \to (\lambda, in_c)$ which will send the received string w_1xw_2P to membrane c. Again, the support-symbol J' is immediately erased and it will never appear in any terminal string. Inside the control-membrane we check that the string does not contain any nonterminal symbol: if it is so, then the string w_1xw_2 will exit the system by a final gemmation due to the rule $P \to \lambda @_0$. Otherwise, if w_1xw_2P contains a symbol $N_h \in (N_1 \cup N_2)$, then the classical rules $N_h \to \sharp$ will introduce the trap symbol $\sharp$ that causes never halting computations. No string will be generated and, once more, we can correctly simulate any production in G.

It follows that we exactly generate the strings of terminal symbols generated by G, that is $L(\Pi_{(G,F)}) = L(G)$. □

We want to point out that, as seen in the proof, a unique membrane suffices for simulating each matrix of type 3, while we need two membranes and in/out communications for each matrix of type 2 and 4. The meta-priority relation and the gemmation of mobile membranes yield here an easy and immediate simulation of the appearance checking mode, unlike all other variants of P systems where this aspect of matrix grammars is harder to be proved.

If we do not use in/out communications, it is possible to show that the family of languages MAT is properly included in the family of languages generated by gemmating P systems. In fact the language $L = \{a^{2^n} \mid n \geq 1\}$, which is a non regular language over one-letter alphabet, can be easily generated by a gemmating P systems of degree 4. It follows that:

Theorem 2. $GP_4(MPri, n(in/out)) - MAT \neq \emptyset$.

4 Solving the HPP in quadratic time

Consider a directed graph $\gamma = (N, A)$ where N is a finite set of n vertices, identified with the numbers $1, 2, \ldots, n$, and A is a set of ordered pairs of vertices (v_i, v_j), for $i, j \in \{1, \ldots, n\}$. The Hamiltonian Path Problem (in short HPP) for γ asks whether or not there exists a path from a given initial vertex v_1 to a final vertex v_n which passes exactly once through each and every vertex of the graph ([3]). We write as r_i the outdegree of the vertex v_i, $\forall\, i \in \{1, \ldots, n\}$, and we ignore useless arcs of the form (v_i, v_i), so r_i will be at most equal to $n - 1$. We show how to construct a P system with gemmation of mobile membranes, with rules similar to those used in [1], which actually finds all Hamiltonian paths in a given graph and not only their existence (if any). The computation halts for all inputs and the problem is solved at most in a quadratic time with respect to the number of vertices.

Theorem 3. *The HPP can be solved by P systems with gemmation of mobile membranes in a quadratic time with respect to the number of vertices.*

Proof. We define a gemmating P system of degree $n + 1$ associated with γ

$$\Pi_{HPP} = (V, T, \mu, M_0, \ldots, M_n, C_0, (C_1, D_1), \ldots, (C_n, D_n), \infty)$$

with the following components:

$V = \{\langle i, k\rangle, [i, k], \langle i, k; j_1, \ldots, j_r\rangle, \sharp \mid 1 \leq i \leq n$ is the label of a vertex v_i, $0 \leq k \leq n - 1$ counts the steps from v_1 to $v_n, j_1, \ldots, j_r$ are labels in $\{1, \ldots, n\}$ such that $(v_i, v_{j_h}) \in A, \forall\, h = 1, \ldots, r\}$;

$T = \{[i, k] \mid 1 \leq i \leq n, 0 \leq k \leq n - 1\}$;

$\mu = [_0[_1]_1[_2]_2 \ldots [_{n-1}]_{n-1}[_n]_n]_0$, that is we define an inner membrane i for each vertex v_i in N;

$M_1 = \{\langle 1, 0\rangle\}$, all other multisets are empty;

$C_0 = \emptyset$,

and with the following sets of rules which, starting from the object $\langle 1,0\rangle$ in membrane 1 and by repeatedly using replication rules in the inner membranes, create all the strings that correspond to paths in γ:

C_1: $\langle \mathbf{1},\mathbf{k}\rangle \to (\sharp;\mathbf{here})$ $\forall\ k=1,\ldots,n-1$
(in membrane 1 if a string contains the symbol $\langle 1,k\rangle$ for $k\neq 0$, then it codifies a wrong path because it surely visited the current membrane twice. We stop such strings by the introduction of the symbol $\sharp$);

D_1: $\langle \mathbf{1},\mathbf{0}\rangle \to ([\mathbf{1},\mathbf{0}]\langle \mathbf{j_1},\mathbf{1}\rangle@_{\mathbf{j_1}};\mathbf{here})$ if $r_1=1$
(if there is a single arc from vertex v_1 to vertex v_{j_1}, then the nonterminal symbol $\langle 1,0\rangle$ is rewritten as the corresponding terminal symbol $[1,0]$, and the string is prolonged by adding $\langle j_1,1\rangle$, which denotes the label of the membrane to be visited and the next step in the path);
$\langle \mathbf{1},\mathbf{0}\rangle \to ([\mathbf{1},\mathbf{0}]\langle \mathbf{j_1},\mathbf{1}\rangle@_{\mathbf{j_1}} \parallel [\mathbf{1},\mathbf{0}]\langle \mathbf{j_2},\mathbf{1}\rangle@_{\mathbf{j_2}};\mathbf{here},\mathbf{here})$ if $r_1=2$
(if there are two arcs exiting from vertex v_1, then we replicate the initial object into two strings at the same step);
$\langle \mathbf{1},\mathbf{0}\rangle \to ([\mathbf{1},\mathbf{0}]\langle \mathbf{j_1},\mathbf{1}\rangle@_{\mathbf{j_1}} \parallel \langle \mathbf{1},\mathbf{0};\mathbf{j_2},\ldots,\mathbf{j_{r_1}}\rangle;\mathbf{here},\mathbf{here})$ if $r_1>2$
(if there are more than two vertices exiting vertex v_1, then we use a replication rule to prolong one string and to memorize all the others vertex-labels in the nonterminal symbol $\langle 1,0;j_2,\ldots,j_{r_1}\rangle$);
$\langle \mathbf{1},\mathbf{0};\mathbf{j_h},\ldots,\mathbf{j_{r_1}}\rangle \to ([\mathbf{1},\mathbf{0}]\langle \mathbf{j_h},\mathbf{1}\rangle@_{\mathbf{j_h}} \parallel \langle \mathbf{1},\mathbf{0};\mathbf{j_{h+1}},\ldots,\mathbf{j_{r_1}}\rangle;\mathbf{here},\mathbf{here})$
$\forall\ h=2,\ldots,r_1-2$
(if there are more than two memorized vertices, then in a step we prolong only one of them, while keeping memorized all the others);
$\langle \mathbf{1},\mathbf{0};\mathbf{j_{r_1-1}},\mathbf{j_{r_1}}\rangle \to ([\mathbf{1},\mathbf{0}]\langle \mathbf{j_{r_1-1}},\mathbf{1}\rangle@_{\mathbf{r_1-1}} \parallel [\mathbf{1},\mathbf{0}]\langle \mathbf{j_{r_1}},\mathbf{1}\rangle@\mathbf{j_{r_1}};\mathbf{here},\mathbf{here})$
(if there are exactly two memorized vertices, then we replicate the single object into a new couple of strings in a single step);

C_i, $\forall\ i=2,\ldots,n-1$, consists of:
$[\mathbf{i},\mathbf{k}] \to (\sharp\ :\ \sharp;\mathbf{here},\mathbf{out})$ $\forall\ k=1,\ldots,n-2$
(if a string containing the symbol $[i,k]$ enters membrane i, then such string must be stopped because it codifies a wrong path and we break it into two substrings by a splitting rule);

D_i, $\forall\ i=2,\ldots,n-1$, consists of rules analogous to those defined for the set D_1 (here the range of the step counter is $1\leq k\leq n-2$ for all rules):
$\langle \mathbf{i},\mathbf{k}\rangle \to ([\mathbf{i},\mathbf{k}]\langle \mathbf{j_1},\mathbf{k+1}\rangle@_{\mathbf{j_1}};\mathbf{here})$ if $r_i=1$;
$\langle \mathbf{i},\mathbf{k}\rangle \to ([\mathbf{i},\mathbf{k}]\langle \mathbf{j_1},\mathbf{k+1}\rangle@_{\mathbf{j_1}} \parallel [\mathbf{i},\mathbf{k}]\langle \mathbf{j_2},\mathbf{k+1}\rangle@_{\mathbf{j_2}};\mathbf{here},\mathbf{here})$ if $r_i=2$;
$\langle \mathbf{i},\mathbf{k}\rangle \to ([\mathbf{i},\mathbf{k}]\langle \mathbf{j_1},\mathbf{k+1}\rangle@_{\mathbf{j_1}} \parallel \langle \mathbf{i},\mathbf{k};\mathbf{j_2},\ldots,\mathbf{j_{r_i}}\rangle;\mathbf{here},\mathbf{here})$ if $r_i>2$;
$\langle \mathbf{i},\mathbf{k};\mathbf{j_h},\ldots,\mathbf{j_{r_i}}\rangle \to ([\mathbf{i},\mathbf{k}]\langle \mathbf{j_h},\mathbf{k+1}\rangle@_{\mathbf{j_h}} \parallel \langle \mathbf{i},\mathbf{k};\mathbf{j_{h+1}},\ldots,\mathbf{j_{r_i}}\rangle;\mathbf{here},\mathbf{here})$
$\forall\ h=2,\ldots,r_i-2$;
$\langle \mathbf{i},\mathbf{k};\mathbf{j_{r_i-1}},\mathbf{j_{r_i}}\rangle \to$
$([\mathbf{i},\mathbf{k}]\langle \mathbf{j_{r_i-1}},\mathbf{k+1}\rangle@_{\mathbf{j_{r_i-1}}} \parallel [\mathbf{i},\mathbf{k}]\langle \mathbf{j_{r_i}},\mathbf{k+1}\rangle@\mathbf{j_{r_i}};\mathbf{here},\mathbf{here})$;

C_n: $\langle \mathbf{n},\mathbf{k}\rangle \to (\sharp;\mathbf{here})$ $\forall\ k=1,\ldots,n-2$
(in membrane n, if $k\neq n-1$ we introduce the symbol $\sharp$ to stop every

string containing the symbol $\langle n, k\rangle$, which codifies a wrong path);

D_n: $\langle \mathbf{n}, \mathbf{n-1}\rangle \to ([\mathbf{n}, \mathbf{n-1}]@_{\mathbf{0}}; \mathbf{here})$
(if a string reaches membrane n and if it contains the symbol $\langle n, n-1\rangle$, then it surely codifies a correct path and it can leave the system).

The computation starts in membrane 1 from the unique object $\langle 1, 0\rangle$: we start from vertex v_1 at the step 0 and, by repeatedly using pre-dynamical replication rules, we prolong all the strings which correspond to paths in γ. The paths can be correctly continued if either no vertex label is repeated in the string or we reach membrane n (that is, the final vertex in γ) at the step $n-1$.
The special symbol $\sharp$ is thus needed in order to break and stop every wrong Hamiltonian path, $\sharp$ is introduced in membrane 1 and n by classical rules, which have meta-priority above all other rules defined inside the membrane, so we can assure that no wrong path will be prolonged in the system.
Observe that, to this aim, we could not use a similar mutation rule in membranes $2, \ldots, n-1$, in fact if a string containing the symbol $[i, k]$ enters membrane i, then it will certainly be of the form $[1, 0]x_1[i, k]x_2\langle i, k'\rangle$, where $k' > k$, x_1 is a (possible empty) string over $\{[j, k]\}$, and x_2 is a non-empty string over $\{[j, k]\}$, for $j \in \{2, \ldots, n-1\}, j \neq i$. If we would use the rule $[i, k] \to (\sharp, here)$ then the last symbol $\langle i, k'\rangle$ would cause the continuation of the path and we would finally have a wrong output. We choose not to use the similar rule $[i, k] \to (\sharp, out)$ because we try to simulate the direct transport through a membrane only for those objects which do not have "too long" length. So we break any wrong string by a splitting rule and then we send to the skin membrane the second halves of such strings, which would otherwise be processed again. As the first half of any wrong string is not dangerous at all, we can decide both to send it out or to keep it inside the current membrane.
We continue in this way only those paths that pass exactly one time through each and every membrane, the computation always stops and we send every Hamiltonian path (if existing) outside the system by a mobile membrane gemmated from membrane n.

Let's now compute the maximum number of steps until a computation halts. We suppose that the outdegree of each vertex is equal to $n-1$. In membrane 1 all the possible continuations of the starting string are generated and sent to the destination membrane step by step, hence the worst complexity case corresponds to the last generated strings, which take $n-2$ steps to be ready to leave the membrane. With two more steps (gemmation and fusion phases) such strings are communicated to any membrane i, for $i = 2, \ldots, n-1$, where again other $n-2$ steps (plus two communication steps) are needed for the local last generated strings to reach their destination membranes. So it takes $n^2 - n$ steps until the last generated strings reach membrane n. Here after three more steps (evolution, gemmation, fusion) the strings codifying Hamiltonian paths (if any) will be sent out of the system. Hence, in total we perform at most $n^2 - n + 3$ steps before the system stops its computation.

The exact number of steps is given by the formula $\left[\sum_{i=1}^{n-1}((r_i - 1) + 2)\right] + 3$, when $r_i \leq n - 1$. □

Note that, in particular, if the maximum outdegree of each vertex of the graph is bounded by 2, then the computation always halts after $3n$ steps. The quadratic time would collapse to linear time also if parallel replication rules were used, as introduced in [5]. In this case, in fact, we could prolong all the paths from each vertex in a single step, then in other two communication steps (gemmation and fusion of the mobile membranes) the strings would be sent to the target membranes. It follows that we would only need $3n$ steps to prolong and output all the Hamiltonian paths in the graph.

5 Final remarks

We have introduced a new kind of communication for P systems and worked with membrane structures and evolution rules of biological inspiration, keeping the model as closer as possible to the real structure of cells, to the aim of shortening the distance from an eventual implementation of the model. We have proved that P systems with such features characterize the recursively enumerable languages and they can solve the Hamiltonian Path Problem in a quadratic time. We close the paper with three topics for further research.

As no priority is defined between classical evolution rules, we could think about a parallel application of all applicable rules over the same string, as in Lindenmayer systems ([11]). The generative power of this variant is still to be analyzed.

The second problem concerns the fact that, for the moment, no pre-dynamical rule can be defined in the skin membrane: up to now P systems have been isolated structures and no rules have ever been defined for letting an object entering the skin membrane from outside. Moreover, no object can be ejected from the system by mobile membranes gemmated from the skin membrane: the mobile membrane could never reach any other P system and the objects would remain forever inside it (no language would be generated in this way). Hence, it would be interesting to define either an *external ambient* for P systems, either *colonies* of P systems which can communicate by mobile membranes, by means of gap junctions or even by getting the objects from outside with a new kind of evolution rules.

Finally, we have seen that P systems which use gemmation of mobile membranes and no in/out communications can generate at least all languages in MAT, but it is still an open problem knowing if the family of generated languages can ever be enlarged using this new communication feature only.

References

1. J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, *IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000*, La Coruna, Spain, 64–74.

2. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. M. R. Garey, D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, 1979, W. H. Freeman and Company.
4. D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31 (1994), 719–728.
5. S. N. Krishna, R. Rama, P systems with replicated rewriting, submitted, 2000.
6. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).
7. Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.
8. Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266, and *Turku Center for Computer Science-TUCS Report* No 218, 1998 (www.tucs.fi).
9. I. Petre, L. Petre, Mobile ambients and P systems, *Workshop on Formal Languages*, FCT99, Iaşi, 1999, *J. Universal Computer Sci.*, 5, 9 (1999), 588–598 (see also *Turku Center for Computer Science-TUCS Report* No 293, 1999, www.tucs.fi).
10. J. E. Rothman, L. Orci, Budding vesicles in cells, March 1996, *Scientific American*.
11. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
12. D. Voet, J. G. Voet, *Biochemistry* (second edition), 1995, John Wiley and Sons, Inc.