Characterising the complexity of tissue P systems with fission rules

Alberto Leporati, Luca Manzoni, Giancarlo Mauri *, Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336/14, 20126 Milano, Italy

ARTICLE INFO

Accepted 20 June 2017

Keywords:
Membrane computing
Computational complexity
Tissue P systems
Counting complexity
Cell fission

ABSTRACT

We analyse the computational efficiency of tissue P systems, a biologically-inspired computing device modelling the communication between cells. In particular, we focus on tissue P systems with fission rules (cell division and/or cell separation), where the number of cells can increase exponentially during the computation. We prove that the complexity class characterised by these devices in polynomial time is exactly $\mathbf{P}^{\#\mathbf{P}}$, the class of problems solved by polynomial-time Turing machines with oracles for counting problems.

1. Introduction

Membrane computing is a branch of natural computing which defines and investigates parallel computing devices (called *P systems*) inspired by the internal working of biological cells and the way they communicate. The original model of P system [9] consists of a hierarchical nesting of membranes delimiting regions; each region contains a *multiset* of symbols describing the chemicals contained therein and their concentrations, and the P system evolves by means of multiset rewriting and communication rules, which move objects around the membrane structure.

Many further variants of P systems have been defined in the literature [11], and most of these variants have been proved to be Turing-equivalent in terms of computability. Furthermore, variants of P systems which are also interesting from a computational complexity perspective have been defined by allowing membranes to *divide*, as in the biological process of mitosis. This allows an exponential increase in the number of membranes during the computation; these membranes can then evolve independently in parallel, performing tasks such as exploring the whole solution space of an **NP**-complete problem in polynomial time. The variant called *P system with active membranes* [10] is even able to solve in polynomial time exactly the problems in the class **PSPACE** [15,18].

Another variant of P systems, called *tissue P systems* [5], consists of an ensemble of single-membrane communicating cells. Hence, this variant lacks the hierarchical nesting of cell-like P systems, and while an exponential number of cells may be created by using division [12] or separation rules [7] (these two kinds of rules differ only with respect to the behaviour of the contents of a cell when it performs binary fission), all the resulting cells are located at the same level, without

* Corresponding author.
E-mail addresses: leporati@disco.unimib.it (A. Leporati), luca.manzoni@disco.unimib.it (L. Manzoni), mauri@disco.unimib.it (G. Mauri), porreca@disco.unimib.it (A.E. Porreca), zandron@disco.unimib.it (C. Zandron).

parent-children distinction. This still allows them to solve **NP**-complete problems in polynomial time [12,7], although the known upper bound **PSPACE** to the complexity class they characterise [17,16] does not seem to be reachable.

In this paper we improve both the lower bound and the upper bound for the class of problems solvable in polynomial time by tissue P systems with cell fission (division and/or separation) rules, and provide an exact characterisation of their computing power. This solves the open problem proposed by Sosík and Cienciala [17] asking whether the above-mentioned **PSPACE** upper bound is indeed optimal. This class of problems is proved to be exactly $\mathbf{P}^{\#\mathbf{P}}$, a class consisting of the problems solved in polynomial time by Turing machines with access to an oracle for a **#P** counting problem [8] and conjectured to be properly included in **PSPACE**. Furthermore, this result holds for *deterministic* tissue P systems, as well as for the more commonly employed *confluent* tissue P systems, where the evolution of the system can be locally nondeterministic, but the computations are eventually all accepting or all rejecting. The upper bound is proved (Section 3, which is an extended version of [3]) by showing that an oracle for **#P** allows us to simulate in polynomial time the communication between exponentially many cells, without the need to explicitly store their individual configurations. The lower bound $\mathbf{P}^{\mathbf{PP}}$, which coincides with $\mathbf{P}^{\#\mathbf{P}}$ [8], is achieved (Section 4) by designing tissue P systems simulating an arbitrary polynomial-time deterministic Turing machine with a **#P** oracle; in particular, cell fission is exploited in order to simulate in parallel the nondeterministic computations of a **PP** machine.

2. Basic notions

We begin by recalling the definition of tissue P systems with division [12] and separation rules [7]; for a more detailed introduction on multiset processing and tissue P systems, we refer the reader to the original paper [5].

Definition 1. A *tissue P system* is a structure $\Pi = (\Gamma, E, w_1, \ldots, w_d, R)$, where:

- Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called *objects*;
- $E \subseteq \Gamma$ is the alphabet of objects initially located in the external environment, in *arbitrarily many* copies (i.e., they are never exhausted during the computation);
- $d \geq 1$ is the *degree* of the system, i.e., the initial number of cells;
- $w_1, \ldots, w_d$ are *finite* multisets over Γ, describing the initial contents of the d cells; here $1, \ldots, d$ are *labels* identifying the cells of the P systems, and 0 is the label of the external environment;
- R is a finite set of rules.

The rules of R are of the following types:

(a) *Communication rules*, denoted in this paper by $[u]_h \leftrightarrow [v]_k$ and in the literature by $(h, u/v, k)$, where h and k are distinct labels (including the environment), and u and v are multisets over Γ (at least one of them non-empty): these rules are applicable if there exists a region with label h containing u as a submultiset and a region k containing v as a submultiset; the effect of the application of the rule is to exchange u and v between the two regions. If $h = 0$ (resp., $k = 0$) then the rule is denoted by $u \leftrightarrow [v]_k$ (resp., $[u]_h \leftrightarrow v$), and in that case we require multiset u (resp., v) to contain at least an object from $\Gamma - E$, i.e., an object with finite multiplicity, if v (resp., u) is empty.[1] In this paper we consider two rules $[u]_h \leftrightarrow [v]_k$ and $[v]_k \leftrightarrow [u]_h$ to be the same, since they would exhibit the same behaviour in all circumstances.

(b) *Division rules*, of the form $[a]_h \rightarrow [b]_h\,[c]_h$, where $h \neq 0$ is a cell label and $a, b, c \in \Gamma$: these rules can be applied to a cell with label h containing at least one copy of a; the effect of the application of the rule is to divide the cell into two cells, both with label h; the object a is replaced in the two cells by b and c, respectively, while the rest of the original multiset contained in h is replicated in both cells.

(c) *Separation rules*, of the form $[a]_h \rightarrow [\Gamma_1]_h\,[\Gamma_2]_h$, where $h \neq 0$ is a cell label, $a \in \Gamma$, and $\{\Gamma_1, \Gamma_2\}$ is a partition of Γ: these rules can be applied to a cell with label h containing at least one copy of a; the effect of the application of the rule is to separate the cell into two cells, both with label h; the object a is consumed, while the objects from Γ_1 in the original multiset contained in h are placed inside one of the cells, and those from Γ_2 in the other. All separation rules in R must share the same partition $\{\Gamma_1, \Gamma_2\}$ of Γ.

A *tissue P system with cell division* only uses communication and division rules, while a *tissue P system with cell separation* only uses communication and separation rules. We use the name *tissue P system with fission* when communication rules are used together with both cell division and cell separation.

A *configuration* $\mathcal{C}$ of a tissue P system consists of a multiset over $\Gamma - E$ describing the objects appearing with finite multiplicity in the environment, and a multiset of pairs (h, w), where h is a cell label and w a finite multiset over Γ, describing the cells. A *computation step* changes the current configuration according to the following set of principles:

[1] Since communication rules are applied in a maximally parallel way, this restriction avoids the situation where infinitely many objects from the environment simultaneously enter a cell.

- Each object can be subject to at most one rule, and each cell can be subject to *any number* of communication rules or, *alternatively*, to a *single* division or separation rule.
- The application of rules is *maximally parallel*: each region is subject to a maximal multiset of rules (i.e., no further rule can be applied).
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.

A *halting computation* $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of the tissue P system Π is a finite sequence of configurations, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules are applicable in $\mathcal{C}_k$.

Tissue P systems can be used as language *recognisers* by employing two distinguished objects *yes* and *no*: we assume that all computations are halting, and that one of the objects *yes* or *no* (but not both) is released into the environment, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the tissue P system is said to be *confluent*. A confluent P system is said to *accept* if its computations are accepting, and to *reject* otherwise. Confluent tissue P systems may be locally nondeterministic, as long as all computations agree on the final result. As a special case, tissue P systems whose initial configuration generates a single computation are called *deterministic*.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser tissue P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a tissue P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [6].

Definition 2. A family of tissue P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be *(polynomial-time) uniform* if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:

- $F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common tissue P system for all inputs of length n, with a distinguished input cell.
- $E(x) = w_x$, where w_x is a multiset encoding the specific input x.
- Finally, Π_x is simply Π_n with w_x added to its input cell.

On the other hand, the family $\mathbf{\Pi}$ is said to be *(polynomial-time) semi-uniform* if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^\star$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of cells and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This is also called a *permissible encoding* [6].

The class of problems solved in polynomial time by uniform (resp., semi-uniform) families of confluent tissue P systems with cell division is denoted by $\mathbf{PMC}_{\mathcal{TDC}}$ (resp., $\mathbf{PMC}^\star_{\mathcal{TDC}}$); the corresponding classes for tissue P systems with separation are $\mathbf{PMC}_{\mathcal{TSC}}$ and $\mathbf{PMC}^\star_{\mathcal{TSC}}$, and those for tissue P systems with *both* kinds of fission rules are $\mathbf{PMC}_{\mathcal{TFC}}$ and $\mathbf{PMC}^\star_{\mathcal{TFC}}$. The inclusions

$$\mathbf{PMC}_{\mathcal{TDC}} \subseteq \mathbf{PMC}^\star_{\mathcal{TDC}} \qquad \mathbf{PMC}_{\mathcal{TSC}} \subseteq \mathbf{PMC}^\star_{\mathcal{TSC}} \qquad \mathbf{PMC}_{\mathcal{TFC}} \subseteq \mathbf{PMC}^\star_{\mathcal{TFC}}$$

hold by definition, since uniformity is a special case of semi-uniformity. Furthermore, we have

$$\mathbf{PMC}_{\mathcal{TDC}} \cup \mathbf{PMC}_{\mathcal{TSC}} \subseteq \mathbf{PMC}_{\mathcal{TFC}} \qquad \mathbf{PMC}^\star_{\mathcal{TDC}} \cup \mathbf{PMC}^\star_{\mathcal{TSC}} \subseteq \mathbf{PMC}^\star_{\mathcal{TFC}}$$

since adding further types of rules does not decrease the computational power. The complexity classes for families of *deterministic* tissue P systems working in polynomial time are $\mathbf{DPMC}_{\mathcal{TDC}}$, $\mathbf{DPMC}_{\mathcal{TSC}}$, and $\mathbf{DPMC}_{\mathcal{TFC}}$, together with their semi-uniform variants; these also enjoy inclusions similar to the confluent case.

Finally, we recall the definitions of the complexity classes **PP**, **#P**, $\mathbf{P^{PP}}$ and $\mathbf{P^{\#P}}$ [8].

Definition 3. The complexity class **PP** consists of decision problems solvable in polynomial time by *threshold Turing machines*, that is, nondeterministic Turing machines where the accepting condition is that the majority of the computations is accepting.

Definition 4. The complexity class **#P** consists of all the *functions* $f\colon \Sigma^\star \to \mathbb{N}$, also called *counting problems*, with the following property: there exists a polynomial time nondeterministic Turing machine N such that, for each $x \in \Sigma^\star$, the number of accepting computations of N on input x is exactly $f(x)$.

Definition 5. The complexity class $\mathbf{P^{\#P}}$ consists of all decision problems solvable in polynomial time by deterministic Turing machines with oracles for **#P** functions. These are Turing machines M^f, with $f \in \mathbf{\#P}$, having a distinguished *oracle tape* and a *query state* such that, when M^f enters the query state, the string x on the oracle tape is replaced in one step with the binary encoding of $f(x)$.

The class $\mathbf{P}^{\#\mathbf{P}}$ coincides with $\mathbf{P}^{\mathbf{PP}}$, the class of decision problems solved in polynomial time with **PP** oracles [19]; we shall take advantage of this equivalence in the rest of this paper.

3. Upper bound

We want to find an upper bound to the class of problems solved in polynomial time by semi-uniform families of confluent tissue P systems with fission rules. The most straightforward way to do so is to simulate tissue P systems by means of suitably limited Turing machines.

The trivial simulation, where the whole configuration of the tissue P system is stored explicitly, requires exponential space in the general case, due to the possible exponential increase in the number of cells generated by cell fission. A more sophisticated simulation, requiring only polynomial space (although exponential time), has been described for tissue P systems with cell separation [16] and those with cell division [17] by Sosík and Cienciala.

By definition, deciding whether a recogniser tissue P system accepts can ultimately be reduced to the question "Is the object *yes* (resp., *no*) ever sent out to the environment during the computation?". This question can be reformulated as the (supposedly simpler) question "Does there exist a time step t such that the object *yes* (resp., *no*) is sent out to the environment at time t?"; assuming that we can answer that question for each individual time step t, the general question can then be answered by iterating across a polynomial number of time steps.

The question "Is an instance of object a sent out to the environment at time t?" can, in principle, be answered by using an oracle for a problem defined in terms of nondeterministic Turing machines: indeed, the nondeterminism can be viewed as a form of parallelism, whereby we can simulate exponentially many cells by making a nondeterministic choice between the two resulting cells whenever a cell fission rule is applied. However, the cells simulated in parallel are not independent, i.e., in order to simulate one of them it is not sufficient to store its configuration; indeed, communication rules can change the contents of a cell according to the configuration of another cell. Thus, answering the above question requires, in general, simulating exponentially many cells per computation of the nondeterministic Turing machine.

This can be avoided by providing further input data. A suitable query is "Given information about the communication rules applied at each computation step, is an instance of object a sent out to the environment at time t?". The extra information allows us to simulate the variation of the configuration of a cell *due to communication rules* by simple table lookups. It is then, of course, necessary to compute the contents of such "communication table". First of all, when asking the query for time step t, the table must only contain the data related to communication rules applied up to time t. By reformulating once again the query, we can not only obtain information about the objects sent to the environment, but also the information we need in order to fill the entries of the communication table related to the next computation step. Except for a few technical details to be explored later, the final form of the query will thus be

Query Q (informal). *Given information about the communication rules applied before computation step t, how many times is communication rule $r = [u]_h \leftrightarrow [v]_k$ applied at time t?*

Notice how the original Boolean query has been replaced by a counting query, to which the former can be reduced. Our goal is to show that this query actually belongs to **#P**, i.e., that the answer can be computed as the number of accepting computations of a suitable nondeterministic Turing machine working in polynomial time.

The first technical question to tackle is how to describe succinctly the communication table to be given as input. This table maps pairs (r, t) to the set of cells where rule r is applied at time t; hence, we must be able to distinguish multiple cells sharing the same label (obtained via fission rules). Since at most 2^t cells per label can exist at time t, we can attach to each of them an integer from the half-open range $[0, 2^t)$, with the obvious restriction that two distinct cells sharing the same label have distinct identifiers at each time step. Concretely, we employ an identifier schema analogous to that proposed by Sosík and Cienciala [17], whereby

- the identifier of the unique cell with label h is 0 in the initial configuration (time $t = 0$), for each label h;
- if the identifier of a given cell at time t is id, then its identifier at time $t + 1$ is $2 \times id$; if the cell is subject to a fission rule, the two resulting cells have identifiers $2 \times id$ and $2 \times id + 1$, respectively, at time $t + 1$.

The cell identifiers can be exploited when simulating a confluent tissue P system in order to solve nondeterministic conflicts. A confluent tissue P system can always be simulated deterministically by arbitrarily selecting any computation (which, by definition, will ultimately give the same result as any other computation). In the rest of the paper we shall make the following assumptions, without loss of generality:

- There is a *linear priority* $\prec$ over the set of rules: in the simulated computation, each rule is assigned as many times as possible to the relevant cells before assigning any other rule with lower priority.
- Furthermore, there is also a linear priority over the sets of cells sharing the same label: in the simulated computation, whenever a communication rule $r = [u]_h \leftrightarrow [v]_k$ is assigned to a pair of cells having labels h and k and identifiers i and j, respectively, there is no other pair of instances of h and k with identifiers i' and j' such that $i' < i$ or $j' < j$ where rule r is applicable. In other words, each rule is assigned to cells in increasing order of identifier.

These two assumptions uniquely identify the computation to be simulated. Furthermore, this particular computation has the following useful property:

Lemma 6. *For each computation step t and each communication rule $r = [u]_h \leftrightarrow [v]_k$ there exists a 4-tuple $T(r,t) = (M_h, \Delta_h, M_k, \Delta_k)$ of non-negative integers such that:*

(i) *All cells with label h (resp., k) having identifier in the range $[0, M_h)$ (resp., $[0, M_k)$) apply rule r in a maximally parallel way at time t (taking into account the objects that have already been assigned to rules with higher priority),* unless *a fission rule with higher priority blocks the cell. Notice that "maximally parallel way" might mean that rule r is applied* zero *times, if the objects occurring in r are not available.*
(ii) *The cell with label h and identifier M_h (resp., label k and identifier M_k) applies rule r* exactly *Δ_h (resp., Δ_k) times at time t.*
(iii) *Cells having label h (resp., k) and identifier greater than M_h (resp., M_k) do* not *apply rule r at time t.*

This property is illustrated by the following example.

Example 7. Suppose that the tissue P system Π has the rules

$$r_1 = [d]_k \to [e]_k\,[f]_k \qquad r_2 = [b]_h \leftrightarrow [a]_k \qquad r_3 = [b\ c]_h \leftrightarrow [e]_k$$

in priority order, and the following configuration[2] at time $t = 3$:

$$[b\ c]_{h_0} \quad [b\ b\ b\ c]_{h_4} \quad [b\ c]_{h_6}$$
$$[a\ a]_{k_0} \quad [a\ d]_{k_1} \quad [b\ e]_{k_4}$$

where the cell identifiers are represented as subscripts of labels h and k.

According to the principles described above, we must first assign rule r_1 in a maximally parallel way. This rule can be applied to cell k_1; communication rules will then be inhibited in this cell at time $t = 3$. The next rule in priority order is r_2, which will be applied once between h_0 and k_0, once between h_4 and k_0, and once between h_4 and k_4. Hence, rule r_2 is maximally applied to h_0 but only twice (rather than 3 times) to h_4; it is also maximally applied to k_0, k_1 (where it can be applied 0 times, since the cell is blocked by r_1) and k_4. This is represented by $T_{r_2,3} = (4, 2, 5, 0)$, that is, rule r_2 is maximally applied to cells with label h and identifier less than 4, only twice to the cell h with identifier 4, maximally to cells with label k and identifier less than 5, and 0 times to the cell with identifier 5 (notice that this cell does not actually exist, and this identifier is only used as a placeholder). The next rule in priority order is r_3, which is applied once between h_4 and k_4. This is represented by $T_{r_3,3} = (5, 0, 5, 0)$, i.e., rule r_3 is applied maximally in all cells with identifier at most 4.

This example can be generalised in order to obtain a complete proof.

Proof of Lemma 6. Let t be any computation step of the tissue P system. We proceed by induction on the set of rules $r_1 \prec r_2 \prec \cdots \prec r_m$. Assume that rules $r_1, \ldots, r_i$ have already been assigned to cells and objects according to the semantics of tissue P systems *and* the priorities over the set of rules and over the cells sharing the same label. Then, by induction hypothesis, there exists a 4-uple $T(r,t)$ satisfying (i)–(iii) for all communication rules $r \preceq r_i$. These values $T(r,t)$ directly represent the assignment of communication rules, and indirectly the assignment of fission rules with priority lower than or equal to r at time t (a fission rule is assigned whenever the object on the right-hand side is present and no communication rule has been previously assigned).

Let us now consider rule r_{i+1}. If this is a division rule $[a]_h \to [b]_h\,[c]_h$ or a separation rule $[a]_h \to [\Gamma_1]_h\,[\Gamma_2]_h$, then it is assigned to all cells with label h containing at least one instance of a where no other rule among $r_1, \ldots, r_i$ has already been assigned at time t; any assignment of rule r_{i+1} will block further rule assignments (for rules $r_{i+2}, \ldots, r_m$) to these cells for this computation step.

If instead r_{i+1} is a communication rule $[u]_h \leftrightarrow [v]_k$, then consider the set of cells with label h and the set of cells with label k, respectively. Define N_h (resp., N_k) as

$$N_h = \sum \left\{ |w_{id}|_u : id \in [0, 2^t) \text{ and cell } h \text{ with identifier } id \text{ has no fission rule assigned at time } t \right\}$$
$$N_k = \sum \left\{ |w_{id}|_v : id \in [0, 2^t) \text{ and cell } k \text{ with identifier } id \text{ has no fission rule assigned at time } t \right\}$$

i.e., as the sum across all identifiers id of the number of times $|w_{id}|_u$ (resp., $|w_{id}|_v$) that multiset u (resp., v) is contained in the multiset w_{id}, where w_{id} is the multiset of objects in the cell with label h (resp., k) and identifier id which are still unassigned at time t. A maximally parallel application of rule r_{i+1} thus consists of exactly $N = \min\{N_h, N_k\}$ applications,

[2] Technically, rules r_1, r_2 and r_3 do not suffice to obtain the configuration described here; however, we may suppose that this configuration has been reached by using rules with lower priority that are never enabled in the rest of the computation.

since the multiset u might have a different number of occurrences among the unassigned objects in cells with label h than v does in cells with label k in the current configuration of the P system. Since rules are always assigned to cells with smaller identifiers first, there exists a *maximum* identifier $M_h \leq 2^t$ (resp., M_k) of cells with label h (resp., k) where rule r_{i+1} is assigned to *all* previously unassigned instances of u (resp., v) in all cells with identifier *strictly smaller* than M_h (resp., M_k).

Notice that the cells with identifier less than M_h (resp., M_k) do not necessarily apply r_{i+1} a total of N times: indeed, the rule might be applicable a few extra times in another *unique* cell, without however exhausting the occurrences of u (resp., v) inside that cell. In that case, that unique cell h (resp., k) with larger identifier *id* applies r_{i+1} the remaining number of times $\Delta_h > 0$ (resp., $\Delta_k > 0$). We can always choose M_h (resp., M_k) to be precisely the identifier of that unique cell. If, instead, the cells with identifier less than M_h (resp., M_k) do apply r_{i+1} *exactly* a total of N times, then we set $\Delta_h = 0$ (resp., $\Delta_k = 0$) and the actual value of M_h (resp., M_k) does not necessarily correspond to an actually existing identifier; for instance, the value of M_h (resp., M_k) can be chosen to be 2^t (which does not correspond to any cell actually existing at time t) when all instances of h (resp., k) apply r_{i+1} as many times as possible.[3]

By letting $T(r_{i+1}, t) = (M_h, \Delta_h, M_k, \Delta_k)$, the statement of the lemma follows. □

The advantage of storing the values $T(r,t) = (M_h, \Delta_h, M_k, \Delta_k)$ across all rules r and time steps t instead of storing the whole configuration of the tissue P system is that this communication table only has a polynomial number of entries, and each entry is an integer of polynomial size. Indeed, the identifiers M_h, M_k and the number of objects Δ_h, Δ_k are exponentially bounded (thus, representable in polynomial space) with respect to the size of the initial configuration of the tissue P system. Even if it is exponentially smaller than the configuration of the tissue P system, the communication table allows us to decide whether the simulated computation is accepting or rejecting simply by checking whether a communication rule sending *yes* or *no* into the environment is applied.

Having defined the communication table T, we are now finally able to rewrite query Q in a formal way.

Query Q. *Given a tissue P system Π with fission rules, a time step t in unary notation, a communication rule $r = [u]_h \leftrightarrow [v]_k$, and a communication table T for Π, with entries $T(\rho, \tau)$ filled for all $\tau < t$ and for $\tau = t$ if $\rho \prec r$, how many times is rule r applied at time t by cells with label h, assuming the availability of arbitrarily many copies of v in cells with label k?*

The reason why the two cells h and k are treated asymmetrically in the query is that this simplifies the analysis of its complexity. The actual number of times that rule r is applicable can be obtained by asking the query a second time with the reversed rule $[v]_k \leftrightarrow [u]_h$ (which only considers the availability of copies of v in cells with label k) and taking the minimum of the two answers, as with N_h and N_k in the proof of Lemma 6.

An oracle for query Q allows us to simulate tissue P systems with cell fission rules with a polynomial slowdown.

Lemma 8. $\mathbf{PMC}^{\star}_{\mathcal{TFC}} \subseteq \mathbf{P}^{Q}$, *hence* $\mathbf{PMC}^{\star}_{\mathcal{TDC}} \cup \mathbf{PMC}^{\star}_{\mathcal{TSC}} \subseteq \mathbf{P}^{Q}$.

Proof. Consider a language $L \in \mathbf{PMC}^{\star}_{\mathcal{TFC}}$ and let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^{\star}\}$ be a semi-uniform family of tissue P systems with fission rules deciding L in polynomial time. Algorithm 1 describes how each Π_x can be constructed and simulated, given the input string x, by a deterministic Turing machine with an oracle for Q.

In line 1 we obtain the description of Π_x by simulating the machine providing the semi-uniformity condition for $\mathbf{\Pi}$ on input x. This, by definition, can be carried out in polynomial time with respect to the length of x.

The loop of lines 2–14 is executed for each simulated time step t, hence, by hypothesis, a polynomial number of times. Inside this loop, the algorithm iterates across all communication rules $r = [u]_h \leftrightarrow [v]_k$ of Π in priority order (lines 3–9) while filling the corresponding entry $T(r,t)$ of the communication table.

We begin (line 4) by assuming that all existing copies of h, i.e., the full range of identifiers $[0, 2^t)$, are allowed to apply rule r, as if there were enough copies of multiset v among the copies of k; we make the same assumption for the cells with label k. We then ask the oracle for Q how many times rule r is applied to cells with label h (line 6) and k (line 7) under those assumptions; call N_h and N_k the two numbers of applications thus obtained. If $N_h \neq N_k$, then the number of copies of u in cells with label h differs from the number of copies of v in cells with label k. For the simulation to be consistent with the semantics of tissue P systems, we need to update one of the ranges of identifiers $[0, M_h)$ or $[0, M_k)$ and one of the values Δ_h, Δ_k in order to ensure that the updated values of N_h and N_k are identical; when this happens, it means that rule r is correctly assigned to cells with label h the same number of times as to cells with label k.

Suppose, for the sake of example, that $N_h < N_k$ (the argument is symmetric if $N_h > N_k$). Then, we reduce the range of cells with label k involved in the application of rule r by repeatedly adjusting the corresponding value M_k and re-evaluating N_k with further queries. By performing a binary search (line 8), we can find in polynomial time ($O(\log 2^t) = O(t)$ iterations) the largest range $[0, M_k)$ of identifiers among those maximising the value of N_k, with the constraint $N_k \leq N_h$. Indeed, there might exist several ranges $[0, M_k)$ maximising the value of N_k under the constraint $N_k \leq N_h$; this can happen

[3] Notice that, since either $N = N_h$ or $N = N_k$ hold, we always have either $\Delta_h = 0$ or $\Delta_k = 0$, respectively. Another consequence is that rule r_{i+1} is applied in a maximally parallel way in *all* cells with label h (resp., k), and thus we can always assume $M_h = 2^t$ (resp., $M_k = 2^t$), although this is not strictly necessary for proving Lemma 6.

```
1  construct Π_x = (Γ, E, w_1, ..., w_d, R) from x
2  for each time step t do
3      for each rule r = [u]_h ↔ [v]_k ∈ R in priority order do
4          T(r, t) := (2^t, 0, 2^t, 0)
5          repeat
6              N_h := number of applications of [u]_h ↔ [v]_k in h at time t according to T
7              N_k := number of applications of [u]_h ↔ [v]_k in k at time t according to T
8              update T(r, t) by binary search
9          until N_h = N_k
10     for each rule r = [u]_h ↔ v do
11         N_0 := number of applications of r in h at time t according to T
12         remove N_0 instances of v and add N_0 instances of u to the environment
13     if yes or no appear in the environment then
14         accept or reject accordingly
```

Algorithm 1: Simulation of semi-uniform families of tissue P systems with cell fission rules.

when not all identifiers in the range $[0, 2^t)$ correspond to actual cells with label k, and when there exist cells with that label where rule r cannot be applied, either because they do not contain v, or they are blocked by a fission rule. When these kinds of cells are close to the right endpoint of the interval $[0, M_k)$, there exist several values of M_k corresponding to the same N_k. If the maximisation of N_k by means of binary search does not lead to $N_k = N_h$, then the difference $N_h - N_k$ is finally recorded as $\Delta_k > 0$; this is the number of times r must be applied by the cell having label k and identifier M_k. This querying procedure is performed even if $h = 0$ or $k = 0$, i.e., one of them is the label of the environment.

The loop of lines 10–12 updates the configuration of the environment, which is the only region whose configuration we explicitly store, by asking the oracle the final number of applications of rules involving the environment, and adjusting the environment multiset accordingly. Notice that the rules not involving the environment are not directly simulated by Algorithm 1, but only indirectly via the querying procedure, since the configurations of the cells are not stored. In particular, the type of fission rules (division and/or separation) employed by $\mathbf{\Pi}$ is irrelevant.

Finally, in lines 13 and 14 the computation is halted when one of the result-objects *yes* or *no* finally appears in the environment.

Since the number of queries needed, as well as the number of bookkeeping operations, is polynomially bounded, the simulation can be performed in $\mathbf{P}^{Q}$. □

In order to give a more precise upper bound for the complexity of simulating tissue P systems, we can now analyse query Q in detail, proving that it can be answered in polynomial time by a counting machine.

Lemma 9. *Query Q is in* **#P**.

Proof. Given a query Q with parameters Π, t, r, and T, Algorithm 2 describes a nondeterministic procedure for the parallel simulation of all cells of Π having label h, where each computation (i.e., a parallel process) actually simulates a single cell. Without loss of generality, the simulation gives higher priority to communication rules, followed by division rules and finally by separation rules (the order within each group of rules is immaterial).

This algorithm manages the identifiers of the cells as described above: the identifier of the unique copy of cell h in the initial configuration is 0 (line 1); if the identifier of a copy of h at time τ is id, then in the next time step (line 23) the identifier is $2 \times id$ (line 3); if, furthermore, the cell divides or separates, then the new copy, simulated by the computation where $i = 1$, has identifier $2 \times id + 1$ (lines 15 and 20).

The algorithm simulates one by one all steps up to t (line 2). In line 4 it initialises an empty multiset *newmultiset* to collect the objects entering the cell via communication rules, or rewritten via division rules; since the rules applied at each step are simulated sequentially by Algorithm 2, we employ this auxiliary multiset (in addition to the actual contents of the cell, named *multiset* in the pseudocode) in order to avoid applying more than one rule to each object.

The loop of lines 5–12 iterates across all communication rules ρ involving h (on either side of the rule). In line 6 we read the values corresponding to the ranges of identifiers for cell labels h and k where rule ρ is applied in the current time step. If the identifier of the cell being simulated belongs to the range $[0, M_h)$, then we apply rule ρ as many times as possible (lines 7–9). On the other hand, if the identifier is exactly M_h, we only apply the rule Δ_h times (lines 10–12). The rule is not applied if the identifier is strictly greater than M_h.

If a division rule is applicable in the cell (this, in particular, requires that no communication rule was applied previously), then we apply the first one in priority order (line 13). This consists in nondeterministically choosing which of the two resulting cells the current computation will continue to simulate (line 14) and updating the identifier and contents of the selected cell (lines 15–17). Notice that this establishes a bijection between computations of the algorithm and instances of cell h.

```
1  id := 0
2  for each time step τ ∈ {1, ..., t} do
3      newid := 2 × id
4      newmultiset := ∅
5      for each communication rule ρ = [u]_h ↔ [v]_k involving label h, in priority order, do
6          (M_h, Δ_h, M_k, Δ_k) := T(ρ, τ)
7          if id < M_h then
8              remove as many copies of u as possible from multiset
9              add the same number of copies of v to newmultiset
10         else if id = M_h then
11             remove Δ_h copies of u from multiset
12             add the same number of copies of v to newmultiset
13     if a division rule [a]_h → [b_0]_h [b_1]_h is applicable then
14         nondeterministically guess a bit i
15         newid := newid + i
16         remove a from multiset
17         add b_i to newmultiset
18     else if a separation rule [a]_h → [Γ_0]_h [Γ_1]_h is applicable then
19         nondeterministically guess a bit i
20         newid := newid + i
21         remove a from multiset
22         remove all objects in Γ_{1−i} from multiset
23     id := newid
24     multiset := multiset ∪ newmultiset
25 accept as many times as the number of applications of r in step t
```

Algorithm 2: Nondeterministic simulation of the cells having label h, with computation of the number of applications of communication rule r at time t.

Finally, if a separation rule is applicable (this requires that no communication or division rule was applied), then we apply the first one in priority order (line 18). The procedure is analogous to the simulation of division rules, except that the contents of the cell being simulated must be updated according to the semantics of cell separation rules (lines 20–22).

We can then update the values of *id* and add to *multiset* the objects that entered or were rewritten inside the current copy of cell h during the computation step just simulated, if any (lines 23 and 24).

After having simulated t steps, we can check the number of times m that input rule r was applied to the cell during the last step. The algorithm can now "fork" m accepting computations by making at most $\lceil \log m \rceil$ nondeterministic choices (line 25). This value contributes to the total number of accepting computations of the algorithm, which will then correspond to the number of applications of rule r at time t, as required. □

By combining Lemmata 8 and 9, we finally obtain an upper bound to the computational power of semi-uniform families of tissue P systems with fission rules working in polynomial time.

Theorem 10. $\mathbf{PMC}^{\star}_{\mathcal{TFC}} \subseteq \mathbf{P}^{\#\mathbf{P}}$, *hence* $\mathbf{PMC}^{\star}_{\mathcal{TDC}} \cup \mathbf{PMC}^{\star}_{\mathcal{TSC}} \subseteq \mathbf{P}^{\#\mathbf{P}}$. □

4. Lower bound

In order to prove that $\mathbf{P}^{\#\mathbf{P}}$ is actually a characterisation of the power of tissue P systems with fission rules, we exploit the equivalence of that class with $\mathbf{P}^{\mathbf{PP}}$ [19]. Furthermore, we define a normal form for threshold Turing machines that guarantees some useful properties of their set of computations.

Lemma 11. *If $L \in \mathbf{PP}$, then L is accepted by a polynomial-time threshold Turing machine M such that*

(i) *M has an odd number of computations;*
(ii) *at least one computation of M is accepting;*
(iii) *at least one computation of M is rejecting;*
(iv) *all computations of M have the same length.*

Proof. Let $L \in \mathbf{PP}$ and let N be a polynomial-time threshold Turing machine such that $L(N) = L$ and having $c(x)$ computations on input $x \in \Sigma^{\star}$, where $a(x)$ of them are accepting. Define M to behave as follows: on input x, nondeterministically choose an integer $i \in \{1, \ldots, 5\}$ (this can be performed by using binary nondeterministic choices) then proceed according

to the value of i. If $i \in \{1,2\}$, then simulate N on input x; if $i \in \{3,4\}$, then reject immediately; finally, accept immediately if $i = 5$. The Turing machine M then has $c'(x) = 2 \times c(x) + 3$ computations (twice the computations of N, corresponding to $i \in \{1,2\}$, and three "dummy" ones), with $a'(x) = 2 \times a(x) + 1$ accepting ones (twice those of N and a "dummy" one, corresponding to $i = 5$). The Turing machine M accepts x if and only if more than half of its computations are accepting, that is

$$a(x) > \frac{c(x)}{2} \iff 2 \times a(x) > c(x) \iff 2 \times a(x) + 1 > c(x) + 1 \iff 2 \times a(x) + 1 > \frac{2 \times c(x) + 2}{2}$$

Since natural numbers are spaced by 1, we can add $1/2$ to the right-hand side of the last inequality and obtain a logically equivalent one:

$$2 \times a(x) + 1 > \frac{2 \times c(x) + 3}{2} \iff a'(x) > \frac{c'(x)}{2}$$

Therefore, M accepts the same language as N but (i) has an odd number of computations on each input x, with (ii) at least an accepting one and (iii) a rejecting one. All computations of M can then be easily padded to the same length with no asymptotic loss of efficiency [8, Proposition 7.1], ensuring (iv). □

4.1. Simulating Turing machines with **PP** oracles

Let M be a single-tape deterministic polynomial-time Turing machine deciding the language $L(M)$ by exploiting a **PP** oracle for the language $L(N)$ decided by the polynomial-time threshold Turing machine N; we assume that N satisfies the properties of Lemma 11. We also establish the following oracle query conventions: the oracle string is written by M between two delimiters on its tape; machine M then enters its query state $q_?$, and in the next computation step it finds itself in state q_{yes} (resp., q_{no}) if the answer to the query is positive (resp., negative); furthermore, after each oracle query the tape head of M is relocated to its initial position. These conventions are easily proved to be equivalent to the standard definition, that uses a distinct oracle tape.

We describe a uniform family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of tissue P systems with cell division simulating M in polynomial time. The algorithm is inspired by the simulation of Turing machines with **PP** oracles by means of P systems with elementary active membranes described in [2].

In the initial configuration, all P systems $\Pi_x \in \mathbf{\Pi}$ have two cells labelled by C and R, and the environment contains infinitely many instances of each object mentioned later (except, of course, for the objects *yes* and *no*). Suppose both M and N work within polynomial space $m = p(n)$ on inputs of length n, including the length of the query strings; such a polynomial p can always be found as an upper bound on the space complexity of both M and N. At the beginning of the simulation of each step of machine M, cell C contains a multiset encoding the current configuration of M. Suppose M is in state q, its tape head is located on the i-th cell (counting from 1) and the tape contains the string $x_1 x_2 \cdots x_m$; then, cell C contains the multiset $q_i, x_{1,1}, x_{2,2}, \cdots, x_{m,m}$, i.e., there is an object representing the current state and tape position (the *state-object*), and one object (a *symbol-object*) for each tape cell, consisting of the tape symbol in the j-th cell, further indexed by its position j.

Suppose the next computation step of M is described by its transition function $\delta(q,a) = (r,b,d)$, where a is the symbol under the tape head and $d \in \{-1,0,+1\}$ denotes the movement of the head. This computation step is simulated by the following communication rules:

$$[q_i\ a_i]_C \leftrightarrow (q_i, a_i) \qquad \text{for } 1 \le i \le m \tag{1}$$

$$[(q_i, a_i)]_C \leftrightarrow (r_i, b, d) \qquad \text{for } 1 \le i \le m \tag{2}$$

$$[(r_i, b, d)]_C \leftrightarrow r_{i+d}\ b_i \qquad \text{for } 1 \le i \le m \tag{3}$$

Rule (1) "packs" the state-object q_i and the symbol-object a_i into a single object (q_i, a_i) by exchanging the objects with the environment. The object (q_i, a_i) is then exchanged with the right-hand side of the transition (r_i, b, d) by rule (2), and the latter object is finally "unpacked" into the two objects r_{i+d}, b_i by rule (3). The latter operation updates the position of the simulated tape head and the contents of the overwritten tape cell; after its execution, the cell contains the multiset encoding the next configuration of M.

Let q_{acc} and q_{rej} denote the accepting and rejecting state of M, respectively. When M reaches one of these states, the corresponding state-object is swapped with an instance of *yes* or *no*, respectively, which is initially located inside cell R:

$$[q_{acc,i}]_C \leftrightarrow [yes]_R \qquad [q_{rej,i}]_C \leftrightarrow [no]_R \qquad \text{for } 1 \le i \le m \tag{4}$$

The object *yes* or *no* is then immediately sent out to the environment, thus establishing the result of the computation of Π_x:

$$[yes]_C \leftrightarrow \epsilon \qquad [no]_C \leftrightarrow \epsilon \tag{5}$$

Here ϵ denotes the empty multiset.

Rules (1)–(5) simulate the behaviour of M as a deterministic Turing machine without oracles. We now need to simulate the oracle querying procedure, as well as the oracle itself. Let $q_?$ be the query state of M, and let q_{yes} and q_{no} be the

states corresponding to a positive and negative oracle answer, respectively. Suppose M enters the query state with the transition $\delta(q,a) = (q_?, b, d)$. After having applied rule (1), instead of using rule (2), we apply cell division as follows:

$$[(q_i, a_i)]_C \to [(q_{?,i}, b, d)]_C\, [(p_i, b, d)]_C \qquad \text{for } 1 \le i \le m \tag{6}$$

Here p denotes the initial state of the threshold Turing machine N deciding the oracle language. Now rule (3) is applied to both resulting cells, and then the simulation of M in the leftmost cell is paused (i.e., there are no applicable rules), and the simulation of N begins in the rightmost cell.

The simulated Turing machine N can identify its input string by searching for the query string delimiters, and use the rest of the tape as a work tape. The simulation of a computation step of N is identical to the simulation of M described above, using rules (1)–(3), as long as N behaves deterministically. When N performs a nondeterministic choice, such as $\delta(q,a) = \{(r,b,d),(s,c,e)\}$, the current cell C is divided, and the two resulting cells continue to evolve in parallel, each simulating one of the two computations arising from the nondeterministic choice:

$$[(q_i, a_i)]_C \to [(r_i, b, d)]_C\, [(s_i, c, e)]_C \qquad \text{for } 1 \le i \le m \tag{7}$$

Eventually, all cells involved in the simulation of N reach a final state; this happens in all cells simultaneously, since all simulated steps proceed in parallel, each requiring exactly three steps of the tissue P system, and we can assume by Lemma 11 that all simulated computations have the same length. The final state-object of N, either $q_{yes,i}$ or $q_{no,i}$, is swapped with a "primed" version ($q'_{yes,i}$ or $q'_{no,i}$) and an auxiliary object $\dagger_3$, to be used later for cleaning up:

$$[q_{yes,i}]_C \leftrightarrow q'_{yes,i}\, \dagger_3 \qquad [q_{no,i}]_C \leftrightarrow q'_{no,i}\, \dagger_3 \qquad \text{for } 1 \le i \le m \tag{8}$$

All result-objects $q'_{yes,i}$ and $q'_{no,i}$ from all instances of cell C are then simultaneously sent to cell R:

$$[q'_{yes,i}]_C \leftrightarrow [\epsilon]_R \qquad [q'_{no,i}]_C \leftrightarrow [\epsilon]_R \qquad \text{for } 1 \le i \le m \tag{9}$$

Inside cell R we combine the result-objects according to the majority rule. This is performed by first pairing the objects $q'_{yes,i}$ and $q'_{no,j}$ and eliminating each pair by replacing it with pairs of α objects:

$$[q'_{yes,i}\, q'_{no,j}]_R \leftrightarrow \alpha\, \alpha \qquad \text{for } 1 \le i \le m, 1 \le j \le m \tag{10}$$

After having maximally applied rule (10), either only objects of the form $q'_{yes,i}$ for some i remain inside R, or only objects of the form $q'_{no,j}$ for some j. By Lemma 11, the total number of result-objects is odd, hence at least one such result-object remains. Furthermore, since N has at least one accepting *and* one rejecting computation, rule (10) is applied at least once, leading to the presence of at least one α object. Hence, exactly one between the following two rules is applicable (one or more times) in the next computation step:

$$[q'_{yes,i}\, \alpha]_R \leftrightarrow q_{yes,1}\, \beta \qquad [q'_{no,i}\, \alpha]_R \leftrightarrow q_{no,1}\, \beta \qquad \text{for } 1 \le i \le m \tag{11}$$

The objects β are used to preserve the size of the multiset contained inside R, in order to allow restoring the initial configuration of this cell at the end of the simulation of the oracle query.

The object $q_{yes,1}$ (resp., $q_{no,1}$) that now appears inside R represents the state and position of the tape head of M after having asked the query. Indeed, either $q'_{yes,i}$ or $q'_{no,j}$ are in the majority (since the number of computations is odd), and all those in the minority have been replaced by α by rules of type (10). Exactly one of the objects $q_{yes,1}$ (resp., $q_{no,1}$) is then sent to the original cell C, replacing the $q_{?,i}$ object:

$$[q_{?,i}]_C \leftrightarrow [q_{yes,1}]_R \qquad [q_{?,i}]_C \leftrightarrow [q_{no,1}]_R \qquad \text{for } 1 \le i \le m \tag{12}$$

In the mean time, the objects $\dagger_3$ count down inside all cells C simulating N:

$$[\dagger_3]_C \leftrightarrow \dagger_2 \qquad [\dagger_2]_C \leftrightarrow \dagger_1 \qquad [\dagger_1]_C \leftrightarrow \dagger \tag{13}$$

and are finally sent to R while rule (12) is applied:

$$[\dagger]_C \leftrightarrow [\epsilon]_R \tag{14}$$

When the objects $\dagger$ reach cell R, they trigger the deletion of all remaining objects $q_{yes,i}, q_{no,i}, q'_{yes,i}, q'_{no,i}, \alpha, \beta$, and $q_{?,i}$:

$$\begin{array}{lll} [q_{yes,i}\, \dagger]_R \leftrightarrow \epsilon & [q'_{yes,i}\, \dagger]_R \leftrightarrow \epsilon & [\alpha\, \dagger]_R \leftrightarrow \epsilon \\ [q_{no,i}\, \dagger]_R \leftrightarrow \epsilon & [q'_{no,i}\, \dagger]_R \leftrightarrow \epsilon & [\beta\, \dagger]_R \leftrightarrow \epsilon \end{array} \qquad [q_{?,i}\, \dagger]_R \leftrightarrow \epsilon \qquad \text{for } 1 \le i \le m \tag{15}$$

By construction, the number of $\dagger$ equals the number of $q_{yes,i}, q_{no,i}, q'_{yes,i}, q'_{no,i}, \alpha, \beta, q_{?,i}$. Therefore, cell R now only contains the objects *yes* and *no*, and is thus again available for the simulation of another oracle query. The simulation of M can thus resume.

The algorithm described above can thus perform the simulation of M and its oracle (implemented by machine N). By analysing the rules of each $\Pi_x \in \mathbf{\Pi}$, we can observe that the P systems are not only confluent, but actually *deterministic*.

First of all, it is easy to check by enumeration that there does not exist a pair of distinct rules $[u_1]_h \leftrightarrow [v_1]_{k_1}$, $[u_2]_h \leftrightarrow [v_2]_{k_2}$ having the same label and multisets $u_1 \subseteq u_2$ on one side; similarly, there does not exist a pair $[u]_h \leftrightarrow [v]_k$, $[a]_h \to [b]_h\,[c]_h$ with $a \in u$ or a pair $[a]_h \to [b_1]_h\,[c_1]_h$, $[a]_h \to [b_2]_h\,[c_2]_h$ where the nondeterminism is explicit.

The second possible source of nondeterminism is given by two distinct communication rules $[u_1]_h \leftrightarrow [v_1]_{k_1}$, $[u_2]_h \leftrightarrow [v_2]_{k_2}$ with $u_1 \cap u_2 \neq \varnothing$ to which objects contained in a cell with label h can be assigned in several possible ways; this does never happen in the P systems described above. Indeed, the rules of type (1) have non-disjoint left-hand sides; however, at most one pair q_i, a_i exists at each time step, since the simulated Turing machine has only one state, one tape position, and one symbol in each tape cell i. Several pairs of rules of type (10), (11) and (15) have intersecting left-hand sides; however, the rules of type (11) are enabled only after applying those of type (10), and the rules of type (15) are only enabled at a later time step; hence, during the simulation no multiset enabling nondeterminism ever appears in a cell with label R. The rules involving intersecting multisets of objects in the environment, those of types (8), (10) and (11), are never in conflict, since the environment contains infinitely many instances of those objects. Furthermore, there is never a conflict in cells with label C between a division rule of type (6) or (7) and any other rule, since each cell C contains at most one instance of an object of the form (q_i, a_i) at any given time, and when that object appears, there is exactly one applicable rule, i.e., the division rule involving it.

The third, and last, possible source of nondeterminism is given by multiple cells with label C (the only dividing cell) simultaneously communicating with another region. As mentioned above, communicating with the environment never poses a problem, given the infinite amount of objects it contains. The only potentially nondeterministic rules are thus those of type (12). However, recall that the tissue P system always contains at most one copy of object $q_{?,i}$, the one contained in the cell with label C which started the querying simulation process, and which is waiting for its result.

Since the tissue P systems of the family $\mathbf{\Pi}$ do not possess any of these three sources of nondeterministic behaviour, each one of them is globally deterministic. This allows us to prove the following result:

Theorem 12. $\mathbf{P^{PP}} \subseteq \mathbf{DPMC}_{\mathcal{TDC}}$.

Proof. Let $L \in \mathbf{P^{PP}}$, and let M be a polynomial-time deterministic Turing machine with an oracle for a **PP** problem deciding it. Let us consider the tissue P system Π_x with cell division described above, simulating M on input x for each $x \in \Sigma^\star$. It is easy to check that the resulting family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is uniform, since there is a constant number of rule schemata (1)–(15), and each schema only depends on a constant number of variables ranging over the (polynomial-sized) set of tape cell positions of M and the machine N deciding the oracle language, namely $1, \ldots, m = p(n)$. Furthermore, the input multiset, to be placed inside the initial cell C, simply consists of the symbols of the input string $x = x_1 \cdots x_n$ with a further index denoting their position. The remaining objects of the initial configuration of Π_x are blank symbols for the tape cells $n+1, \ldots, m$, and the result-objects *yes* and *no*, initially located in cell R; furthermore, the environment contains all objects with infinite multiplicity, except of course for the *yes* and *no* objects, which do not initially appear in the environment.

The simulation of each computation step of M requires a constant number of steps (three) of Π_x, with the exception of oracle query steps, which require the simulation (with a similar algorithm) of the threshold Turing machine N on a query string. Since N works in polynomial time, the family $\mathbf{\Pi}$ simulates M with a polynomial slowdown, and $L \in \mathbf{DPMC}_{\mathcal{TDC}}$ follows. □

4.2. Lower bound for separation

The $\mathbf{P^{PP}}$ lower bound we proved above also holds for tissue P systems with cell separation. We can prove this result by showing how cell division can, under suitable hypotheses, be simulated by means of cell separation. This requires replicating the cell contents before actually applying the separation rules. The main assumptions we make are that the simulated P systems are *deterministic* and that the number of objects appearing when cell division occurs is always known in advance.

Lemma 13. *In the absence of further conflicting rules, a cell division rule $r = [a]_h \to [b]_h\,[c]_h$ can be simulated in constant time by cell separation and communication rules if r is always applied to cells containing exactly m objects, for some $m \in \mathbb{N}$, including exactly one instance of object a.*

Proof. Let Γ be the original alphabet of the P system. The object a appearing on the left-hand side of r is exchanged with a "tilded" version $\tilde{a}$, together with a timer object σ_2 (which will ultimately trigger the separation rule) and m instances of η, one per original object in h:

$$[a]_h \leftrightarrow \tilde{a}\,\sigma_2\,\underbrace{\eta, \cdots, \eta}_{m \text{ times}} \tag{16}$$

The replication of the contents of h is triggered by the objects η, which exchange each object $d \in \Gamma$ with four "copies" of it:

$$[d\,\eta]_h \leftrightarrow d_1\,d_2\,d'\,d' \qquad \text{for } d \in \Gamma \tag{17}$$

A similar exchange is made with $\tilde{a}$, but bringing in two "copies" each of the objects b and c appearing on the right-hand side of r rather than replicating a itself:

$$[\tilde{a}\ \eta]_h \leftrightarrow b_1\ c_2\ b'\ c' \tag{18}$$

The simulation will now use a new, auxiliary cell k. All "primed" objects are immediately sent to this cell by the following rules:

$$[d']_h \leftrightarrow [\epsilon]_k \qquad \text{for } d \in \Gamma \tag{19}$$

In the mean time, the timer object σ_2 is decremented:

$$[\sigma_2]_h \leftrightarrow \sigma_1 \qquad [\sigma_1]_h \leftrightarrow \sigma \tag{20}$$

Upon reaching σ, the following separation rule is triggered:

$$[\sigma\,]_h \rightarrow [\Gamma_1]_h\,[\Gamma_2]_h \tag{21}$$

where Γ_1 contains all objects in Γ with the extra subscript 1, and Γ_2 the same with the extra subscript 2; the remaining objects can be distributed arbitrarily between Γ_1 and Γ_2, since they never appear inside h when the cell divides, except for σ, which is consumed by rule (21).

While rule (21) is applied, the objects in cell k are exchanged with an "unprimed" version of themselves:

$$[d']_k \leftrightarrow d \qquad \text{for } d \in \Gamma \tag{22}$$

In the next computation steps, these copies of d are each exchanged with a copy of d_1 or d_2:

$$[d_1]_h \leftrightarrow [d]_k \qquad [d_2]_h \leftrightarrow [d]_k \qquad \text{for } d \in \Gamma \tag{23}$$

This produces, inside the two cells resulting from the separation, the same contents that would have been produced by the cell division rule r. □

A single auxiliary cell k can be used even when multiple division rules are associated to cell h (as long as the tissue P system is deterministic) and when multiple instances of h divide simultaneously. The role of the objects d' is to wait outside cell h; indeed, the objects d_1, d_2 cannot wait themselves inside cell h, otherwise the communication rule implementing the waiting would conflict with the separation rule (21).

Notice that the hypotheses of Lemma 13 hold for the simulation of Turing machines with counting oracles by means of tissue P systems with cell division described in Section 4. Indeed, rules (6) and rule (7) are applied deterministically to instances of cell C, which always contain the same number $p(n)$ of objects when division is triggered. Simulating rule (6) with the procedure described in Lemma 13 is straightforward, since only one instance of cell C is active (i.e., has applicable rules).

On the other hand, when rule (7) is applicable there are, in general, several instances of cell C operating in parallel (simulating different computations of the threshold Turing machine N). Since the algorithm requires all cells simulating computations of N to halt the simulation at the same time step, we need to slow down the simulation of steps where no nondeterministic choice is made by N to account for the multiple-step implementation of rule (7) by means of cell separation and communication. This can be straightforwardly achieved by replacing rule (2), which is applied only when simulating deterministic steps of N, by the following rules, applied sequentially:

$$\begin{aligned} &[(q_i, a_i)]_C \leftrightarrow (r_i, b, d)_4 \qquad [(r_i, b, d)_4]_C \leftrightarrow (r_i, b, d)_3 \\ &[(r_i, b, d)_3]_C \leftrightarrow (r_i, b, d)_2 \qquad [(r_i, b, d)_2]_C \leftrightarrow (r_i, b, d)_1 \qquad \text{for } 1 \le i \le m \\ &[(r_i, b, d)_1]_C \leftrightarrow (r_i, b, d) \end{aligned} \tag{24}$$

This slowdown allows us to synchronise the simulation when using cell separation rules. By noticing that the simulation of division by means of separation of Lemma 13 preserves the determinism of the simulation of Section 4, we obtain the corresponding lower bound for tissue P systems with cell separation:

Theorem 14. $\mathbf{P}^{\mathbf{PP}} \subseteq \mathbf{DPMC}_{\mathcal{TSC}}$. □

5. Conclusions

The results of Theorems 10, 12, and 14 can be summarised as the main result of this paper, namely that uniform or semi-uniform families of confluent or deterministic tissue P systems with fission rules all characterise $\mathbf{P}^{\#\mathbf{P}}$ in polynomial time:

Theorem 15. *The following complexity classes coincide:*

$$[\mathbf{D}]\mathbf{PMC}^{[\star]}_{\mathcal{TFC}} = [\mathbf{D}]\mathbf{PMC}^{[\star]}_{\mathcal{TDC}} = [\mathbf{D}]\mathbf{PMC}^{[\star]}_{\mathcal{TSC}} = \mathbf{P}^{\#\mathbf{P}}$$

where [**D**] *denotes optional determinism, and* [⋆] *optional semi-uniformity.* □

This result shows that these computing models do not reach (under the usual complexity-theoretical assumptions) the power of polynomial-space Turing machines when working in polynomial time; rather, they can be deterministically simulated in polynomial time with an oracle for a **#P** problem. Thus, they are weaker (assuming $\mathbf{P}^{\#\mathbf{P}} \neq \mathbf{PSPACE}$) than other variants of membrane systems with a hierarchical membrane structure, such as P systems with active membranes [18]. Furthermore, we have proved that the $\mathbf{P}^{\#\mathbf{P}}$ upper bound is actually tight, and thus an exact characterisation of the computing power of these devices. This shows a remarkable similarity (except, as is currently known, for the deterministic case) to cell-like P systems with active membranes using membrane division rules only for *elementary* membranes (i.e., membranes not containing further membranes), which characterise the same complexity class in polynomial time [1]. Actually, the characterisation already holds for P systems with active membranes of depth 1, consisting of an outermost membrane containing only one level of children membranes. Indeed, both computing modes lack the ability to generate the complex hierarchical membrane structure that seems to be needed in order to solve **PSPACE**-complete problems, such as full binary trees of exponential size [15]; however, P systems with active membranes with membrane structures of *constant* depth $d > 1$, which are impossible in standard tissue P systems, are able to solve intermediate problems [2], namely those in the complexity class $\mathbf{P}^{\mathbf{C}_d\mathbf{P}}$, where $\mathbf{C}_d\mathbf{P}$ is the d-th level of the *counting hierarchy* [20]. We can identify this as the main limiting factor, rather than the form of the rules (which are more sophisticated in P systems with active membranes) since a variant of P systems with symport/antiport rules (similar to those of tissue P systems) and membrane division, but with a hierarchical cell-like structure, is actually able to reach **PSPACE** [14].

6. Discussion

We can identify several open problems that we feel deserve further investigation. First of all, we conjecture that **PSPACE** can be reached by tissue P systems with division or separation rules if the systems are allowed to be *non-confluent*, that is, if the same tissue P system can have both accepting and rejecting computations, and the final result is determined by the existence of at least one accepting computation, as for nondeterministic Turing machines. The reason why we believe this to be the case is that non-confluent P systems with active membranes of depth 1, using only elementary membrane division, do indeed have the ability to solve all **PSPACE** problems [4].

Another related question is whether confluence and determinism in other variants of P systems, in particular P systems with active membranes, characterise the same complexity classes. It seems plausible that, in the majority of cases, confluence can be reduced to strict determinism by sequentialising the application of conflicting rules by means of techniques such as timer subscripts for the objects. An alternative is to exhibit a deterministic normal form for a class of P systems, as it is done in this paper for tissue P systems with fission rules working in polynomial time. However, an actual proof of this result is still missing.

A further question involves the comparison of cell division and cell separation rules. The two kinds of rules seem to provide the same main ability, that is, generating exponentially many cells in polynomial time. Is there a resource bound that identifies distinct complexity classes for tissue P systems with cell division and cell separation?

Finally, it might be interesting to minimise the length of communication rules $[u]_h \leftrightarrow [v]_k$, that is, the value $|u| + |v|$. It is known that a maximum length of 2 across all communication rules suffices in order to solve **NP**-complete problems in polynomial time with cell division; furthermore, a length of 3 suffices for tissue P systems with cell separation (both values are actually necessary, assuming $\mathbf{P} \neq \mathbf{NP}$) [13]. It is, however, unknown whether these values are enough to solve $\mathbf{P}^{\#\mathbf{P}}$ problems; the algorithms described in Section 4 use length 4 for tissue P systems with cell division, and even polynomial length with respect to the input size for cell separation.

Acknowledgments

This work has been supported by Fondo d'Ateneo (FA) 2015 of Università degli Studi di Milano-Bicocca: "Complessità computazionale e applicazioni crittografiche di modelli di calcolo bioispirati".

References

[1] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Simulating elementary active membranes, with an application to the P conjecture, in: M. Gheorghe, G. Rozenberg, P. Sosík, C. Zandron (Eds.), Membrane Computing, 15th International Conference, CMC 2014, in: Lecture Notes in Computer Science, vol. 8961, Springer, 2014, pp. 284–299.

[2] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Membrane division, oracles, and the counting hierarchy, Fund. Inform. 138 (2015) 97–111, https://doi.org/10.3233/FI-2015-1201.

[3] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Tissue P systems can be simulated efficiently with counting oracles, in: G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (Eds.), Membrane Computing, 16th International Conference, CMC 2015, in: Lecture Notes in Computer Science, vol. 9504, 2015, pp. 251–261.

[4] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Shallow non-confluent P systems, in: A. Leporati, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, 17th International Conference, CMC 2016, in: Lecture Notes in Computer Science, vol. 10105, 2016, pp. 307–316.
[5] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón, Tissue P systems, Theoret. Comput. Sci. 296 (2003) 295–326, https://doi.org/10.1016/S0304-3975(02)00659-X.
[6] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Nat. Comput. 10 (2011) 613–632, https://doi.org/10.1007/s11047-010-9244-7.
[7] L. Pan, M.J. Pérez-Jiménez, Computational complexity of tissue-like P systems, J. Complexity 26 (2010) 296–315, https://doi.org/10.1016/j.jco.2010.03.001.
[8] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1993.
[9] Gh. Păun, Computing with membranes, J. Comput. System Sci. 61 (2000) 108–143, https://doi.org/10.1006/jcss.1999.1693.
[10] Gh. Păun, P systems with active membranes: attacking NP-complete problems, J. Autom. Lang. Comb. 6 (2001) 75–90.
[11] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010.
[12] Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Tissue P systems with cell division, Int. J. Comput. Commun. Control 3 (2008) 295–303, https://doi.org/10.15837/ijccc.2008.3.2397.
[13] M.J. Pérez-Jiménez, The P versus NP problem from the membrane computing view, Eur. Rev. 22 (2014) 18–33, https://doi.org/10.1017/S1062798713000598.
[14] B. Song, M.J. Pérez-Jiménez, L. Pan, Efficient solutions to hard computational problems by P systems with symport/antiport rules and membrane division, Biosystems 130 (2015) 51–58, https://doi.org/10.1016/j.biosystems.2015.03.002.
[15] P. Sosík, The computational power of cell division in P systems: beating down parallel computers?, Nat. Comput. 2 (2003) 287–298, https://doi.org/10.1023/A:1025401325428.
[16] P. Sosík, L. Cienciala, Computational power of cell separation in tissue P systems, Inform. Sci. 279 (2014) 805–815, https://doi.org/10.1016/j.ins.2014.04.031.
[17] P. Sosík, L. Cienciala, A limitation of cell division in tissue P systems by PSPACE, J. Comput. System Sci. 81 (2015) 473–484, https://doi.org/10.1016/j.jcss.2014.10.006.
[18] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: a characterization of PSPACE, J. Comput. System Sci. 73 (2007) 137–152, https://doi.org/10.1016/j.jcss.2006.10.001.
[19] S. Toda, Simple characterizations of P(#P) and complete problems, J. Comput. System Sci. 49 (1994) 1–17, https://doi.org/10.1016/S0022-0000(05)80082-0.
[20] K.W. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Inform. 23 (1986) 325–356, https://doi.org/10.1007/BF00289117.