Simulating counting oracles with cooperation

Alberto Leporati[1], Luca Manzoni[2], Giancarlo Mauri[1],
Antonio E. Porreca[3], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126, Milan, Italy
{alberto.leporati,giancarlo.mauri,claudio.zandron}@unimib.it
[2] Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste
Via Alfonso Valerio 12/a, 24127, Trieste, Italy
lmanzoni@units.it
[3] Université Publique

Abstract. Many variants of P systems with active membranes are able to solve traditionally intractable problems. Sometimes they also characterize well known complexity classes, depending upon the computational features they use. In this paper we continue the investigation of the importance of (minimal) cooperative rules to increase the computational power of P systems. In particular, we prove that monodirectional shallow chargeless P systems with active membranes and minimal cooperation working in polynomial time precisely characterise $\mathbf{P}_{\|}^{\#\mathbf{P}}$, the complexity class of problems solved in polynomial time by deterministic Turing machines with a polynomial number of parallel queries to an oracle for a counting problem.

1 Introduction

Membrane systems, also called *P systems*, have been introduced by Gh. Păun in 1998 as a framework for defining parallel models of computation inspired from the functioning of living cells. For an introduction to the subject we refer the reader to [17,19], whereas the latest information and an extensive bibliography can be found in the P systems web-page [1]; a useful reference is also *The Oxford Handbook of Membrane Computing* [18], that contains a comprehensive presentation of the state of knowledge in 2010.

Membrane Computing is currently an active field of research, where three types of models have been extensively studied: cell-like P systems [15], based on the hierarchical structure of the membranes within a living cell; tissue-like P systems [10], based on the intercommunication between the cells in a biological tissue; and spiking neural P systems [3] inspired by the electrical impulses that neurons emit as information.

The computing elements of P systems are usually arranged as the nodes of tree-like or graph-like architectures, and perform their computations by rewriting multisets of objects through transformation rules taken from appropriate formal grammars. Since their birth, P systems have stimulated a relevant number of investigations both on the theoretical side – dealing with computability and their computing power and efficiency in solving computationally difficult problems – and from the point of view of applications – studying how P systems can be applied to simulate several kinds of natural phenomena.

Focusing on cell-like P systems, one of the models that have attracted most attention, especially from the theoretical point of view, are P systems *with active membranes* [16]. In this kind of P systems the space is delimited in different (possibly nested) regions via membranes, mimicking the way cellular membranes separate the inner part of a cell from the external environment. The biological inspiration does not end here: inside each membrane multiple objects, representing chemical substances, are transformed by rewriting rules, mimicking biochemical reactions. Moreover, each membrane may have an associated electrical charge, which typically can be negative, positive, or neutral, that affects the applicability of the rules embedded in the membrane. The communication between different regions of space is ensured by the ability of substances to move in and out of membranes (also depending on the membrane charge) and, possibly, to even dissolve a membrane, making all its content "fall out" in the containing region. The name *P systems with active membranes* stems from the fact that membranes play an active role during the computations, either by influencing the rules to be applied through charges, or by modifying the membrane hierarchy through membrane division or dissolution.

During the two decades following their introduction, P systems with active membranes have been employed to solve classically intractable problems, like **NP**-complete ones [23] or, more recently, problems in the complexity class $\mathbf{P}^{\#\mathbf{P}}$ and in the entire counting hierarchy [5]. To reach these results, the simulation of Turing machines (TM) provide an important building block. In particular, the construction of P systems simulating TM using as few membranes (or cells) as possible and limiting the depth of the system is one of the "tricks" that allows the nesting of multiple machines in order to solve problems in large complexity classes. For example, nesting of non-deterministic machines (where the non-determinism was simulated by membrane division) and a counting mechanism allow to characterize $\mathbf{P}^{\#\mathbf{P}}$, the class of all problems solvable by a deterministic TM with access to a **#P** oracle [5,8]. The same ideas can be applied to tissue P systems, where the different communication topology makes even more important to keep TM simulations compact [7].

Considering the computational features used by P systems with active membranes, we have that uniform families of this kind of systems, with charges and bidirectional communication, are able to solve $\mathbf{P}^{\#\mathbf{P}}$-complete problems when only one level of membrane nesting (those kinds of P systems are usually called *shallow systems*) is allowed [4,5], and **PSPACE**-complete problems when this restriction is removed [21]. The presence of simple cooperation rules, like the ones provided by antimatter, where two opposite objects can annihilate each other, allows the systems to reach $\mathbf{P}^{\#\mathbf{P}}$ with a shallow membrane structure, also when the systems have no charges [8]. Even when the communication is severely restricted, as in monodirectional systems [6], where send-in is forbidden, uniform families of P systems with active membranes with charges characterize $\mathbf{P}^{\mathbf{NP}}$ or, if shallow, the class $\mathbf{P}^{\mathbf{NP}}_{\|}$, as shown in [6]. It is interesting to see that this is not the case for monodirectional systems with antimatter: the additional cooperation provided by object annihilation makes it possible to perform a counting operation once (in a destructive way), thus providing families of this kind of systems the ability to solve all problems in $\mathbf{P}^{\#\mathbf{P}}$ where only one oracle query suffices, even with only one level of nesting [8].

In this paper we continue the investigation of the importance of cooperation to increase the computational power of P systems. In particular, we show that monodirec-

tional systems with minimal cooperation [22,13,12] working in polynomial time also characterize the class of all decision problems solvable in polynomial time by a deterministic Turing machine with access to a *single* query to a **#P** oracle. This result shows that charges are not essential for monodirectional shallow P systems, thus helping in the understanding of the roles of depth and the direction of communication for reaching beyond **NP** [20].

The rest of this paper is organized as follows: Section 2 introduces the basic notions which are used in the following. Section 3 shows how a single **#P** query can be simulated, thus allowing to use P systems as oracles, and in Section 4 the main result is presented. Section 5 contains the conclusions, and proposes some directions for future research.

2 Basic Notions

In this paper we consider (semi)uniform families of P systems with active membranes without charges and using minimal cooperative evolution rules, send-out rules, and elementary division rules. For the technical details we refer the reader to Valencia-Cabrera et al. [22]. For an introduction on how P systems operate and the related notions of formal language theory and multiset processing, we refer the reader to *The Oxford Handbook of Membrane Computing* [18].

Definition 1. *A* polarizationless monodirectional P system with active membranes*, of initial degree* $d \geq 1$*, is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where:

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Λ *is a finite set of labels;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are multisets (finite sets with multiplicity) of objects in* Γ*, describing the initial contents of each of the* d *regions of* μ*;*
- R *is a finite set of rules.*

The rules in R considered in this paper are of the following types:

(a) *Minimal Cooperative Object evolution rules*, of the form $[u \to w]_h$.
They can be applied inside a membrane labelled by h and containing the multiset of objects u; all the objects in u are rewritten into the multiset w (i.e., the objects in u are removed from the multiset in h and replaced by the objects in w). Here we only allow *minimal* cooperative object evolution rules, i.e., such that the multiset u contains at most two objects (in symbols, $|u| \leq 2$).

(c) *Send-out communication rules*, of the form $[a]_h \to [\,]_h\, b$.
They can be applied to a membrane labelled by h and containing an occurrence of the object a; the object a is sent out from h to the parent region, becoming b.

(e) *Elementary division rules*, of the form $[a]_h \to [b]_h\,[c]_h$. They can be applied to a membrane labelled by h containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h; the object a is replaced, respectively, by b and c, while the other objects of the multiset contained in membrane h are replicated in both membranes.

A P system is *shallow* if the depth of its membrane structure is at most *one*. That is, the outermost membrane can contain only elementary membranes.

A computation step changes the current configuration of the system according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for cooperative object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel* [18]. Each object appearing on the left-hand side of evolution or communication rules must be subject to exactly one copy of them. Analogously, each membrane can only be subject to one send-out or division rule (types (c) and (e)) per computation step; for this reason, these rules will be called *blocking rules* in the rest of the paper. As a result, the only objects and membranes that do not evolve are those associated with no rule.
- 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 from a certain configuration.
- In each computation step, all the chosen rules are applied simultaneously in an atomic way. However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps whereby each membrane evolves only after its internal configuration (including, recursively, the configurations of the membrane substructures it contains) has been updated. Obviously, rules can be applied to objects coming from an inner membrane by a communication rule only at the next computation step.
- Any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ 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 of Π are applicable in $\mathcal{C}_k$.

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 either one copy of object yes or one of object no is sent out from the outermost membrane, 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 P system is said to be *confluent*.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\Pi = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x deciding the membership of x in a 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 [11].

Definition 2. *A family of P systems $\Pi = \{\Pi_x : x \in \Sigma^\star\}$ is* (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 P system for all inputs of length n, with a distinguished input membrane.*
- *$E(x) = w_x$, where w_x is a multiset encoding the specific input x.*
- *Finally, Π_x is simply Π_n with w_x added to a specific membrane, called the* input membrane.

Any explicit encoding of Π_x is allowed as output of the construction, as long as it is at most polynomially shorter than the one where the rules are listed one by one, the membrane structure is represented in such a way that all membranes are listed one by one and their content is encoded in unary. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [11] for further details on the encoding of P systems.

We also consider polynomial-time Turing machines with oracles for counting problems in the complexity class #**P** [14] and, in particular, the two complexity classes $\mathbf{P}^{\#\mathbf{P}[1]}$, where only one query is allowed, and $\mathbf{P}_{\parallel}^{\#\mathbf{P}}$, when any polynomial number of queries is allowed, but they must all be carried out *in parallel*, that is, all query strings are prepared in advance before actually interrogating the oracle (in other words, later queries are not adaptive with respect to the answers to previous ones). The two classes $\mathbf{P}^{\#\mathbf{P}[1]}$ and $\mathbf{P}_{\parallel}^{\#\mathbf{P}}$ actually turn out to be equivalent, as proved in [8]:

Proposition 1 (Leporati et al. [8]). *A polynomial number of parallel* #**P** *queries can be simulated by a single* #**P** *query in polynomial time (which can be expressed in symbols as $\mathbf{P}_{\parallel}^{\#\mathbf{P}} = \mathbf{P}^{\#\mathbf{P}[1]}$).*

Proof. A single query does never depend on the results of previous queries, thus the inclusion $\mathbf{P}^{\#\mathbf{P}[1]} \subseteq \mathbf{P}_{\parallel}^{\#\mathbf{P}}$ holds by definition.

Conversely, let M be a deterministic Turing machine running in polynomial time $p(n)$ with parallel oracle queries for a function $f \in$ #**P**, and let N be a nondeterministic Turing machine having $f(x)$ accepting computations for each input string x of length n and running in polynomial time $q(n)$.

Then $f(x) \leq 2^{q(|x|)}$ for each input string x, since $2^{q(n)}$ is the maximum number of computations of N on an input of length n (assuming binary nondeterministic choices). Clearly, due to its running time, the machine M can only ask queries with query strings of length bounded by $p(n)$, which means that each query answer is an integer bounded by $2^{q(p(n))}$, and M can ask up to $p(n)$ queries.

Let $x_1, x_2, \ldots, x_{p(n)}$ be the query strings of M on a run on a given input, putting $x_j = \varepsilon$ for $j \geq i$ if M asks less than i queries, and let $g\colon \Sigma^\star \to \mathbb{N}$, with Σ the union of the query alphabet and the separator symbol \$, be defined as

$$g(x_1\$x_2\$\ldots\$x_{p(n)}) = \sum_{i=1}^{p(n)} B^i \times f(x_i)$$

where $B = 2^{q(p(n))} + 1$; this corresponds to encoding all the query answers as a base-B integer. Then, a single query to g contains all the information that can be obtained by asking up to $p(n)$ parallel queries to f, since each value $f(x_i)$ can be recovered in polynomial time by computing

$$f(x_i) = \left\lfloor \frac{g(x_1\$x_2\$\ldots\$x_{p(n)})}{B^{i-1}} \right\rfloor \bmod B.$$

The function g is also in **#P**, since this class is closed under summations and products [2], and this proves that $\mathbf{P}^{\#\mathbf{P}}_{\|} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$. □

3 Simulating a single #P query monodirectionally

It is quite easy to simulate efficiently (actually, in linear time) a deterministic Turing machine working in polynomial time by means of a uniform family of P systems [9]. A configuration of the Turing machine can be encoded as a multiset of objects as follows:

that is, each symbol (including blanks) is subscripted by an index corresponding to the number of the tape cell, and the state of the machine is also represented as an object, subscripted by the index of the cell currently under the tape head. Blank tape cells are represented by the $\Box_i$ objects. Then, each transition of the machine, say $\delta(q,b) = (r,a,d)$ with $d = \pm 1$, is simulated by a set of cooperative evolution rules replicated for each legitimate tape position:

$$[q_i\, b_i \to r_{i+d}\, a_i]_h \qquad \text{for } 0 \le i < s(n) \tag{1}$$

where $s(n)$ is the polynomial space bound of the machine tape. These rules replace the symbol b_i under the tape head by a_i, and update the state symbol q_i to r_{i+d}, which also updates the position of the tape head.

A *nondeterministic* Turing machine can be simulated by dividing elementary membranes, replacing the rules (1) by

$$[q_i\, b_i \to \langle q_i, b_i\rangle]_h \qquad \text{for } 0 \le i < s(n) \tag{2}$$
$$[\langle q_i, b_i\rangle]_h \to [\langle r_{i+d}, a_i\rangle']_h\, [\langle s_{i+e}, c_i\rangle']_h \qquad \text{for } 0 \le i < s(n) \tag{3}$$
$$[\langle r_{i+d}, a_i\rangle' \to r_{i+d}\, c_i]_h \qquad \text{for } 0 \le i < s(n) \tag{4}$$
$$[\langle s_{i+e}, b_i\rangle' \to s_{i+e}\, c_i]_h \qquad \text{for } 0 \le i < s(n) \tag{5}$$

in the case of the nondeterministic transition $\delta(q,b) = \{(r,a,d),(s,c,e)\}$. Notice that, without loss of generality, we may assume that every nondeterministic transition produces exactly two configurations; this can always be ensured at the possible expense of further computation steps. The rules of type (2) "pack" the head-state object and the object

representing the symbol under the tape head into a single object; this is necessary in order to respect the membrane division rule format. The rules of type (3) then perform the transition rule by dividing the membrane and rewriting the packed object into the two objects representing the two possible evolutions of the configuration, one in each of the resulting membranes; these objects are primed, in order to signal that they are the *result* of the transition and not the left-hand side. Finally, the two resulting objects are "unpacked" by the rules of type (4) and (5), thus obtaining the two possible Turing machine configurations inside the divided membranes. In order to avoid synchronisation issues due to the three-step simulation of a nondeterministic transition vs the one-step simulation of deterministic ones, we also slow down the latter accordingly, using the rules

$$[q_i\ b_i \to \langle q_i, b_i \rangle]_h \qquad \text{for } 0 \le i < s(n) \tag{6}$$
$$[\langle q_i, b_i \rangle \to \langle r_{i+d}, a_i \rangle']_h \qquad \text{for } 0 \le i < s(n) \tag{7}$$
$$[\langle r_{i+d}, a_i \rangle' \to r_{i+d}\ a_i]_h \qquad \text{for } 0 \le i < s(n) \tag{8}$$

in the case of the deterministic transition $\delta(q,b) = (r,a,d)$.

Thus, a single membrane (or a number of membranes obtained by division of a single initial one, in the case of nondeterminism) can efficiently simulate a polynomial-size tape Turing machine and, in particular, a Turing machine working in polynomial time. On the other hand, by using several nested membranes it is possible to efficiently simulate oracle queries [9]. With bidirectional P systems (i.e., standard P systems using both send-in and send-out rules) the simulation of the Turing machine is paused, then one usually sends the query string into a child membrane, where another Turing machine for the oracle language is simulated, possibly using membrane division; the answer is then sent out and the simulation of the original Turing machine is resumed.

With monodirectional P systems we proceed in the opposite direction: we first duplicate and send out the multiset encoding the configuration of the Turing machine being simulated, then the oracle machine is simulated in the innermost membrane, and the result is sent out, where the simulation of the original Turing machine can then resume [9]. This process is depicted in Figure 1.

When the simulated Turing machine answering the query is nondeterministic, several result-objects yes are sent out from the divided membranes; they can be counted and operated upon by the Turing machine simulated in the external membrane by converting all objects of the same type to the binary number representing their multiplicity. This can be accomplished by using cooperative evolution rules as follows:

$$[\mathsf{yes} \to 1_0]_k \tag{9}$$
$$[1_i\ 1_i \to 1_{i+1}]_k \qquad \text{for } 0 \le i < m \tag{10}$$

where m is the maximum number of bits for the answer, which can be computed as in the proof of Proposition 1. These rules produce the multiset of 1_i for all positions i where the number of yes objects produced as the answer to the query expressed in binary notation, contains a 1. By combining this with the multiset $0_m\ 0_{m-1}\ \cdots\ 0_1\ 0_0$ using the rules

$$[0_i\ 1_i \to 1_i]_k \qquad \text{for } 0 \le i \le m \tag{11}$$

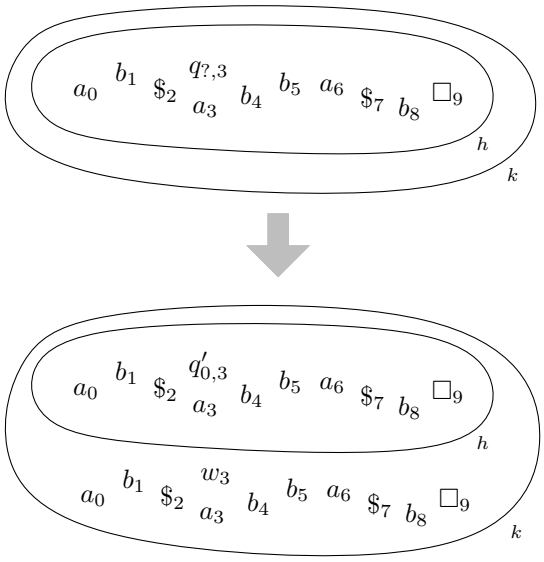

Fig. 1. Simulating an oracle query by means of monodirectional P systems. The portion of configuration delimited by \$ corresponds to the oracle query string, and the tape head is located on its first symbol. When the Turing machine being simulated inside membrane h enters the query state $q_?$, its configuration is duplicated and sent out. The head-state object inside membrane h now represents the state q'_0, the initial state of the Turing machine to be simulated in order to answer the oracle query, while the head-state object outside (in membrane k) represents a "dummy" symbol waiting for state w.

i.e., by deleting the objects 0_i corresponding to the existing objects 1_i, we obtain the binary notation for the answer to the query, which can then be processed by the original Turing machine (now being simulated inside membrane k) as part of its tape.

The existence of the simulation described here, together with Proposition 1, proves the following result:

Theorem 1. *Deterministic polynomial time Turing machines making a polynomial number of queries to a* **#P** *problem can be simulated in polynomial time by shallow chargeless monodirectional P systems with active membranes and minimal cooperation rules.* □

4 Simulating P systems with a single #P query

Here we show how it is possible to formulate an oracle query in **#P** such that we can use it to decide if a confluent P system working in polynomial time with the set of rules described above accepts or rejects.

The query used by the Turing machine that simulates a shallow chargeless P system with active membranes and minimal cooperation, working in polynomial time, is the following:

Query 1. *Given the description of a P system Π, an elementary membrane label h, an object type a, and a time t expressed in unary notation, how many objects of type a are collectively sent out by membranes with label h at time t?*

It is now necessary to prove that the answer to the query can be actually computed by a function in **#P**, i.e., it is a "valid" oracle query for the Turing machines that we are considering. Here we only give a sketch of the proof; all the details can be found in [8].

Lemma 1. *Query 1 is in* **#P**.

Proof. Query 1 can be answered by essentially simulating a single-membrane P system; indeed, if Π is monodirectional, no object can enter membrane h from the parent membrane, and if h is elementary, neither can objects from children membranes. By the Milano Theorem [23], a P system without division (thus, in particular, a single membrane without division), even with cooperative evolution rules, can be simulated in deterministic polynomial time by a Turing machine. By allowing nondeterminism, the divisions $[a]_h \rightarrow [b]_h\ [c]_h$ of membrane h can be simulated in polynomial time by nondeterministically choosing whether to simulate the "left" (where a is rewritten as b) or the "right" membrane (where a is rewritten as c) resulting from the division. After simulating t steps, this results in a nondeterministic computation tree with a leaf for each instance of membrane h. Each computation must accept if and only if an object of type a is sent out at time t, which gives us a number of accepting computations identical to the number of objects a that are collectively sent out at time t by membranes labelled by h, proving that the query is in **#P**. □

We are now ready to prove the main theorem of this paper:

Theorem 2. *A family of (semi)uniform shallow chargeless monodirectional P systems with active membranes and minimal cooperation running in polynomial time can be efficiently simulated by a single* **#P** *query.*

Proof. Given an input string x, the corresponding P system Π_x can be constructed in polynomial time by a deterministic Turing machine M that simulates the (two) Turing machine(s) establishing the (semi)uniformity condition.

Before beginning the actual simulation of Π_x, we can ask a number of queries to an oracle for Query 1; in particular, we ask Query 1 for each possible value of h (labels of elementary membranes), of a (symbols of the alphabet of Π_x), and of t (time steps between 0 and $p(|x|)$, where p is the polynomial running time of the family). Clearly, this is a polynomial number of parallel queries. These queries can then be combined into a single **#P** query by means of Proposition 1.

Then, the external membrane of Π_x can be simulated by Turing machine M. Since this membrane does not divide, by the Milano Theorem [23] it can be simulated deterministically in polynomial time, except for the objects coming from the children membranes, which are allowed to divide. But these have already been precomputed by

asking the oracle queries, and can simply be added with the corresponding multiplicity in the correct time step to the configuration of the outermost membrane. The simulation can then be carried out correctly until the result object (yes or no) is sent out to the environment, and the simulation algorithm accepts or rejects correspondingly. □

5 Conclusions

We have shown that minimal cooperation for monodirectional, shallow P systems with active membranes without charges is sufficient to reach and characterize $\mathbf{P}_{\|}^{\#\mathbf{P}}$. This minimal amount of cooperation seems actually necessary and it might be expressed either explicitly (as done here), or implicitly (as with antimatter [8]). However, the minimal amount of cooperation actually required to "count", thus allowing the construction of a **#P** oracle, is still an open research avenue. In fact, while minimal cooperation can simulate annihilation rules in P systems with antimatter, it is unclear if there exist ways of performing cooperative actions that are even weaker, while still attaining the ability to "count" and perform **#P** queries.

Acknowledgement

Antonio E. Porreca was funded by his salary of French public servant, affiliated to Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France.

References

1. The P systems webpage. http://ppage.psystems.eu/
2. Fortnow, L.: Counting complexity. In: Hemaspaandra, L.A., Selman, A.L. (eds.) Complexity Theory Retrospective II, pp. 81–107. Springer (1997)
3. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae **71**(2-3), 279–308 (2006), https://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-08
4. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014. Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2014), https://doi.org/10.1007/978-3-319-14370-5_18
5. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae **138**(1–2), 97–111 (2015), https://doi.org/10.3233/FI-2015-1201
6. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing **15**(4), 551–564 (2016), https://doi.org/10.1007/s11047-016-9565-2
7. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Characterising the complexity of tissue P systems with fission rules. Journal of Computer and System Sciences **90**, 115–128 (2017), https://doi.org/10.1016/j.jcss.2017.06.008
8. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theoretical Computer Science **701**, 161–173 (2017), https://doi.org/10.1016/j.tcs.2017.03.045

9. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Subroutines in P systems and closure properties of their complexity classes. Theoretical Computer Science **805**, 193–205 (2020), https://doi.org/10.1016/j.tcs.2018.06.012
10. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science **296**(2), 295–326 (2003), https://doi.org/10.1016/S0304-3975(02)00659-X
11. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing **10**(1), 613–632 (2011), https://doi.org/10.1007/s11047-010-9244-7
12. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. Journal of Membrane Computing **1**(2), 85–92 (2019), https://doi.org/10.1007/s41965-018-00004-9
13. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: A path to computational efficiency through membrane computing. Theoretical Computer Science **777**, 443–453 (2019), https://doi.org/10.1016/j.tcs.2018.12.024
14. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
15. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000), https://doi.org/10.1006/jcss.1999.1693
16. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)
17. Păun, Gh., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science **287**(1), 73–100 (2002), https://doi.org/10.1016/S0304-3975(02)00136-6
18. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
19. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002)
20. Sosík, P.: P systems attacking hard problems beyond NP: a survey. Journal of Membrane Computing **1**, 198–208 (2019), https://doi.org/10.1007/s41965-019-00017-y
21. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences **73**(1), 137–152 (2007), https://doi.org/10.1016/j.jcss.2006.10.001
22. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Polarizationless P systems with active membranes: Computational complexity aspects. Journal of Automata, Languages and Combinatorics **21**(1–2), 107–123 (2016), https://doi.org/10.25596/jalc-2016-107
23. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001), https://doi.org/10.1007/978-1-4471-0313-4_21