A Survey on Space Complexity of P Systems with Active Membranes

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

leporati/luca.manzoni/mauri/porreca/zandron@disco.unimib.it

Abstract. P systems with active membranes are a variant of P systems where membranes play an active role during the computation, for example by dividing existing membranes in order to create new ones. In this way, an exponential number of membranes can be obtained in polynomial time, and then used in parallel to attack computationally hard problems. Many interesting questions arise concerning the trade–off between time and space needed to solve various classes of computational problems by means of such membrane systems. In this paper we overview the main results presented in the literature concerning this subject.

1 Introduction

P systems with active membranes have been introduced in [14] as a variant of P systems where the membranes play an active role in the computation: an electrical charge, that can be positive (+), neutral (0), or negative (−), is associated with each membrane, and the application of the rules can be controlled by means of these electrical charges. Moreover, new membranes can be created during the computation by division of existing ones. A very interesting feature of such systems is that, using these operations, one can create an exponential number of membranes in polynomial time, and use them in parallel to solve computationally hard problems. Characterisation of classic time complexity classes has been obtained in this way, like **P** ([24]), **NP** (see, e.g., [4], [14] or [22]) and **PSPACE** ([23]). By considering P systems with active membranes using various features and having different limitations, concerning, e.g., communication, rules, or membrane structure, relations with other complexity classes were also investigated ([5], [6], [19], [25]).

This possibility raises many interesting questions concerning the trade–off between the time and space needed to solve various classes of computational problems by means of membrane systems. In order to clarify such relations, a definition of space complexity for P systems has been proposed in [16], on the basis of an hypothetical implementation of P systems by means of real biochemical materials: every single object and every single membrane requires some constant physical space.

Research on the space complexity of P systems with active membranes has shown that these devices, when using a polynomial amount of space, exactly characterize the complexity class **PSPACE**, as shown in [17] and [18]. This result has then been generalized, showing that any Turing machine working in space $\Omega(n)$ can be simulated with a polynomial space overhead [1].

A natural research topic that follows immediately is to clarify what are the classes of problems solved by P systems which make use of logarithmic space. The first natural approach, when considering the use of sublinear space in the framework of membrane systems, is to compare logarithmic space P systems with Turing machines using the same amount of space. It has been shown [20] that **DLOGTIME**–uniform (a standard, weak uniformity condition usually considered for families of Boolean circuits) P systems with active membranes, using a logarithmic amount of space, are able to simulate logarithmic-space deterministic Turing machines, and thus to solve all problems in the class **L**. In [10] it is pointed out that, while logarithmic-space Turing machines can only generate a polynomial number of distinct configurations, P systems working in logarithmic space have *exponentially* many potential ones, and thus they can be exploited to solve computational problems that are harder than those in **L**. In particular, polynomial-space Turing machines can be simulated by means of P systems with active membranes using only logarithmic auxiliary space, thus obtaining a characterization of **PSPACE**.

However, an even lower amount of space suffices: P systems using only a *constant* amount of space have also been considered; in this case, it turned out [9] that, quite surprisingly, a constant amount of space is sufficient (and trivially necessary) to solve all problems in **PSPACE**. This result challenges our intuition of space, formalized in the definition of space complexity for P systems adopted so far. Thus, a more accurate estimate of the space required by a configuration of a P system was proposed. Using the new space definition, all the results involving at least a polynomial amount of space, according to the first definition, still hold. The difference appears only when P systems with severely tight bounds on the amount of space used during computations are considered.

2 Basic Notions

For a comprehensive introduction to P systems we refer the reader to *The Oxford Handbook of Membrane Computing* [15]. A recent list of open problems and research topics on various aspects of membrane computing can be found in [3]. The definition of space complexity for P systems has been introduced in [16].

In order to consider general space complexity classes in the framework of P systems (i.e., including sublinear and, possibly, constant space P systems), we need to define a meaningful notion of space inspired by sublinear space definition for Turing machines: for this reason, when necessary in the following we will consider, instead of a single generic alphabet Σ, two distinct alphabets: an *INPUT* alphabet and a *WORK* alphabet, the difference being that the input objects cannot be rewritten and do not contribute to the size of the configura-

tion of a P system. Indeed, this allow us to define, as usually done for Turing machines, the size of a configuration as the sum of the number of membranes in the current membrane structure and the total number of working objects they contain. Let us recall here the basic definitions related to P systems with active membranes with an input alphabet [20]:

Definition 1. *A* P system with (elementary) active membranes *having initial degree* $d \geq 1$ *is a tuple* $\Pi = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$*, where:*

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Δ *is another alphabet, disjoint from* Γ*, called the* input alphabet*;*
- Λ *is a finite set of labels for the membranes;*
- 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 strings over* Γ *describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules over* $\Gamma \cup \Delta$*.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

A description of the available kinds of rule follows. This description differs from the original definition [14] only in that new input objects may not be created during the computation.

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$
  They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w). At most one input object $b \in \Delta$ may appear in w, and only if it also appears on the left-hand side of the rule (i.e., if $b = a$).
- *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β. If $b \in \Delta$ then $a = b$ must hold.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are released into the surrounding region unaltered, except that an occurrence of a becomes b. If $b \in \Delta$ then $a = b$ must hold.

– *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
They can be applied to a membrane labelled by h, having charge α, 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 and charges β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes. If $b \in \Delta$ (resp., $c \in \Delta$) then $a = b$ and $c \notin \Delta$ (resp., $a = c$ and $b \notin \Delta$) must hold.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

– Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane, several evolution rules can be applied simultaneously).
– The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved in communication, dissolution, or elementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
– 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.
– 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 microsteps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (the outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
– The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 $\mathcal{C}_k$. A *non-halting* computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many

configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as language *recognisers* (see, e.g. [2]) by employing two distinguished objects `yes` and `no`; exactly one of these must be sent out from the outermost membrane, and only in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists. Unless otherwise specified, the P systems in this paper are to be considered confluent.

In order to solve decision problems (i.e., decide languages over an alphabet Σ), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a 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 each input length [12].

Definition 2. *Let $\mathcal{E}$ and $\mathcal{F}$ be classes of functions. A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* $(\mathcal{E}, \mathcal{F})$-uniform *if and only if*

- *There exists a function $F \in \mathcal{F}$ such that $F(1^n) = \Pi_n$, i.e., mapping the unary representation of each natural number to an encoding of the P system processing all inputs of length n, and defining a specific membrane as the* input membrane.
- *There exists a function $E \in \mathcal{E}$ mapping each string $x \in \Sigma^\star$ to a multiset $E(x) = w_x$ (represented as a string) over the input alphabet of Π_n, where $n = |x|$.*
- *For each $x \in \Sigma^\star$ we have $\Pi_x = \Pi_n(w_x)$, i.e., Π_x is Π_n with the multiset encoding x placed inside the input membrane.*

Definition 3. *If the mapping $x \mapsto \Pi_x$ is computed by a* single *polynomial-time Turing machine, the family $\boldsymbol{\Pi}$ is said to be $\mathcal{F}$–*semi-uniform *(where $\mathcal{F}$ is a class of functions). In this case, inputs of the same size may be associated with P systems having possibly different membrane structures and rules.*

Generally, the above mentioned classes of functions $\mathcal{E}$ and $\mathcal{F}$ are complexity classes; in the most common uniformity condition $\mathcal{E}$ and $\mathcal{F}$ denote polynomial-time computable functions, although weaker complexity classes are used for some results presented in this paper.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. 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 [12] for further details on the encoding of P systems.

Finally, we describe how space complexity for families of recogniser P systems is measured, and the related complexity classes [16, 20].

Definition 4. *Let $\mathcal{C}$ be a configuration of a recogniser P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects from Γ (i.e, the non-input objects) they contain. If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a computation of Π, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}.$$

The space required by Π *itself is then obtained by computing the space required by all computations of Π and taking the supremum:*

$$|\Pi| = \sup\{|\boldsymbol{\mathcal{C}}| : \boldsymbol{\mathcal{C}} \text{ is a computation of } \Pi\}.$$

Finally, let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recogniser P systems, and let $s\colon \mathbb{N} \to \mathbb{N}$. We say that $\boldsymbol{\Pi}$ operates within space bound s *iff $|\Pi_x| \leq s(|x|)$ for each $x \in \Sigma^\star$.*

By $(\mathcal{E}, \mathcal{F})$-$\mathbf{MC}_{\mathcal{D}}(f(n))$ (resp. $(\mathcal{E}, \mathcal{F})$-$\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$) we denote the class of languages which can be decided by $(\mathcal{E}, \mathcal{F})$-uniform families of confluent P systems of type $\mathcal{D}$ (in the following we will mainly refer to P systems with active membranes, and we denote this by setting $\mathcal{D} = \mathcal{AM}$), where each $\Pi_x \in \boldsymbol{\Pi}$ operates within time (resp. space) bound $f(|x|)$. The corresponding class when we consider semi-uniform families is denoted by $(\mathcal{E}, \mathcal{F})$-$\mathbf{MC}^\star_{\mathcal{D}}(f(n))$ (resp. $(\mathcal{E}, \mathcal{F})$-$\mathbf{MCSPACE}^\star_{\mathcal{D}}(f(n))$).

The class of problems that can be solved in [semi-uniform] $(\mathcal{E}, \mathcal{F})$-logarithmic (respectively polynomial) space is denoted by $(\mathcal{E}, \mathcal{F})$-$\mathbf{LMCSPACE}^{[\star]}_{\mathcal{D}}$ (respectively $(\mathcal{E}, \mathcal{F})$-$\mathbf{PMCSPACE}^{[\star]}_{\mathcal{D}}$).

3 Space Complexity Results

In [17] it has been shown that recognizer P systems with active membranes, using three polarizations, are able to solve all problems in **PSPACE** working in polynomial space and exponential time.

Theorem 1. $\mathbf{PSPACE} \subseteq \mathbf{PMCSPACE}_{\mathcal{D}}$.

Proof. (Sketch) The **PSPACE**-complete problem Q3SAT is solved by a P system working in polynomial space. The solution is uniform, in the sense that a fixed P system is able to solve all the instances of Q3SAT of a given size. □

In [18] it has been shown that such P systems can be simulated by Turing machines with only a polynomial increase in space requirements.

Theorem 2. $[\mathbf{N}]\mathbf{PMCSPACE}^{[\star]}_{\mathcal{D}} \subseteq \mathbf{PSPACE}$, *where* $[\mathbf{N}]$ *denotes optional non-confluence, and* $[\star]$ *optional semi-uniformity.*

Proof. (Sketch) The inclusion $\mathbf{PMCSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{PSPACE}$ is proved by simulating a nondeterministic P system working in polynomial space by a Turing machine working in polynomial nondeterministic space, which can then be reduced to polynomial deterministic space by using Savitch's theorem [13]. □

Together, the previous results give a precise characterization of the class **PSPACE** in terms of space complexity classes for membrane systems.

This result was then generalized in [1], by showing that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. As a consequence, the classes of problems solvable by P systems with active membranes and by Turing machines coincide up to a polynomial with respect to space complexity.

Theorem 3. *Let M be a single-tape deterministic Turing machine working in time $t(n)$ and space $s(n)$, including the space required for its input. Then there exists a uniform family of confluent P systems $\boldsymbol{\Pi}$ with active membranes operating in time $O\big(t(n)s(n)\log s(n)\big)$ and space $O\big(s(n)\log s(n)\big)$ such that $L(\boldsymbol{\Pi}) = L(M)$.*

Proof. (Sketch) The techniques used in [17] and [18] to simulate Turing machines via uniform families of P systems do not seem to apply when the space bound is super-exponential, because membranes are identified by binary numbers. In fact, when dealing with a super-exponential number of different membrane labels, such numbers would be made of a super-polynomial number of digits, and such systems cannot be built in a polynomial number of steps by a deterministic Turing machine, as required by the notion of polynomial-time uniformity usually employed.

Instead, multiple copies of a single "dot" object are used to represent the cell numbers in unary notation, and all membranes representing cells of the Turing machine have the same label. A configuration of the Turing machine M where, $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{K-1}\}$, q is the state of the machine, the visited portion of the tape has length m, the string on the visited portion of the tape is $w = w_1 \ldots w_m \in \Sigma^m$, and the head is placed on tape cell $p \in \{1, \ldots, m\}$, is encoded as follows:

- Three membranes labelled by q, p, and m contain, respectively, the unary encoding of q, p, and m, that is, as many copies of the dot object as the corresponding value.
- The i–th cell of the Turing machine M containing the j–th symbol of the alphabet, is simulated by means of a membrane containing the value $K \times i + j$ in unary notation.

To simulate a computation step of the Turing machine M we first need to identify, among all membranes labelled by t, the one corresponding to the cell located under the tape head of M. Notice that these membranes are externally indistinguishable, and they differ only in the unary value contained in it. A

nondeterministic guess among all these membranes is performed, and the chosen membrane is then checked to verify if it is indeed the right one; if this is not the case, then the membrane is marked, and the process is repeated until the correct membrane is eventually found.

Once correctly identified the involved membrane, the computation step is simulated, working on numbers in unary notation through a subroutine that simulates a register machine to update the configuration of the P system, according to the transition step of the Turing machine. □

From Theorem 3 we obtain inclusions of complexity classes for Turing machines and P systems when the space bound is at least linear (since we are dealing with single-tape Turing machines). In particular, for every function $f(n) \in \Omega(n)$ the following inclusions hold: $\mathbf{TIME}(f(n)) \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MC}_{\mathcal{AM}}(O(f(n)^3))$ and $\mathbf{SPACE}(f(n)) \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(f(n)^2))$.

Moreover, by combining the previous results, we can prove *equality* between space complexity classes for P systems and Turing machines under some (not very restrictive) assumptions on the set of space bounds we are interested in.

Theorem 4. *Let $\mathcal{F}$ be a class of functions $\mathbb{N} \to \mathbb{N}$ such that*

- *$\mathcal{F}$ contains the identity function $n \mapsto n$.*
- *If $s(n) \in \mathcal{F}$ and $p(n)$ is a polynomial, then there exists some $f(n) \in \mathcal{F}$ with $f(n) \in \Omega(p(s(n)))$.*

Then $\mathbf{SPACE}(\mathcal{F}) = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F})$. In particular, we have the following equalities:

$$\mathbf{PSPACE} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{AM}}$$
$$\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{EXPMCSPACE}_{\mathcal{AM}}$$
$$2\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}2\mathbf{EXPMCSPACE}_{\mathcal{AM}}$$
$$k\mathbf{EXPSPACE} = (\mathbf{L}, \mathbf{L})\text{-}k\mathbf{EXPMCSPACE}_{\mathcal{AM}}.$$

Another consequence of the possibility of P systems to simulate Turing machines with a polynomial overhead and vice versa is that we can translate theorems about the space complexity of Turing machines into theorems about P systems. As an example, the Savitch's theorem and the Space hierarchy theorem for Turing machines can be proved almost immediately for large enough space complexity bounds.

4 Simulating Logarithmic–Space Turing Machines

Considering membrane systems that work in logarithmic space is one of the first natural research topics that have been addressed once the results described in the previous section have been obtained. We first recall a result from [20] showing that P systems with active membranes, using a logarithmic amount of space, are able to simulate logarithmic-space deterministic Turing machines, and thus to solve all problems in the class **L**.

In order to consider such systems, we need to define a uniformity condition for the families of P systems that is weaker than the usual **P** uniformity, to avoid the possibility to solve a problem directly by using the Turing machine that builds the P systems we use to compute. One such possibility is to consider **DLOGTIME**-uniformity, defined on the basis of **DLOGTIME** Turing machines [11]. Another problem that the efficient simulation of logarithmic space Turing machines (or other equivalent models) has to face, is that it cannot use a polynomial number of working objects, to avoid violating the logarithmic space condition.

It has been shown in [20] that such problems can be avoided by a simulation that uses membrane polarization both to communicate objects through membranes as well as to store some information.

Theorem 5. *Consider a deterministic Turing machine M, having an input tape of length n, and with a work tape of length $O(\log n)$. Then there exists a* (**DLOGTIME**, **DLOGTIME**)*-uniform family $\boldsymbol{\Pi}$ of confluent recogniser P systems with active membranes, that works in logarithmic space, such that $L(M) = L(\boldsymbol{\Pi})$.*

Proof. (sketch) Consider a Turing machine M working in logarithmic space. The P system Π_n that simulates M on inputs of length n is composed of:

- A skin membrane containing a *state object* $q_{i,w}$ to indicate that M is currently in state q and its tape heads are on the i-th and w-th symbols of the input and work tape, respectively.
- $O(\log n)$ nested membranes (INPUT tape membranes) containing, in the innermost one, the input symbols of M, and $O(\log(n))$ membranes to store the work tape of M (WORK tape membranes).
- Two sets of membranes, whose sizes depend on the dimensions of the input and the working alphabet of M (SYMBOL membranes), respectively.

To simulate a computation step of M, the state object enters the INPUT membranes, storing the bits corresponding to the actual position of the INPUT head of M in their polarizations. Only one object (corresponding to the INPUT symbol actually read) can travel to the outermost membrane by using send-out rules; the other objects stop moving because they have the wrong charges. Then, the state object identifies the symbol actually under the WORK head (using the WORK tape membranes) and proceeds to simulate the transition of M using the SYMBOLS membranes.

Each P system Π_x (simulating each $M(x)$ such that $|x| = n$) only requires $O(\log |x|)$ membranes and objects besides the input objects; moreover, the family $\boldsymbol{\Pi}$ is (**DLOGTIME**, **DLOGTIME**)-uniform. The time required by the simulation is $O\big(n \cdot t(n)\big)$, where $t(n)$ is the maximum number of steps performed by M on inputs of length n. □

An immediate corollary of Theorem 5 is that the class of problems solved by logarithmic-space Turing machines is contained in the class of problems solved by (**DLOGTIME**, **DLOGTIME**)-uniform, logarithmic-space P systems with active membranes.

Corollary 1. $\mathbf{L} \subseteq (\mathbf{DLOGTIME}, \mathbf{DLOGTIME})\text{-}\mathbf{LMCSPACE}_{\mathcal{AM}}$. □

5 Simulating Polynomial-Space Turing Machines in Logarithmic Space

The result presented in the previous section only represents a lower bound for the power of logarithmic-space P systems; as a matter of fact, already in [20] it was conjectured that such a bound could be improved, as P systems working in logarithmic space have an *exponential* number of different configurations, which could possibly be used to efficiently solve harder problems than those in the class **L**. It turned out [10] that this is the case, and that polynomial-space deterministic Turing machines can be simulated by means of P systems with active membranes using only logarithmic auxiliary space, thus characterising **PSPACE**.

The simulation was based on two key ideas. First, input objects (of the form τ_i, where τ is a symbol from the alphabet of the Turing machine, and index i indicates a position on its tape) are distributed, during the computation, in various substructures. Apart from an initial phase, the value of τ is disregarded: the symbol σ written on the i-th tape cell of the Turing machine being simulated can be inferred from the label of the substructure that contains τ_i. The second idea is applied when querying the symbol under the tape head: the position i of the head is written in binary in the electrical charges of the membranes composing the substructure where the object τ_i is placed, so that the only input object having the correct subscript can leave the substructure corresponding to the sought symbol, and reach the skin membrane. The depth of each substructure is logarithmic, thus allowing to represent a polynomial number of possible head positions. As a result, we can simulate any polynomial space computation of a deterministic Turing machine with only a logarithmic number of symbols (plus a polynomial number of read-only input symbols) and membranes.

Theorem 6. *Let M be a single-tape deterministic Turing machine working in polynomial space $s(n)$ and time $t(n)$. Then, there exists an $(\mathbf{L}, \mathbf{L})$-uniform family $\boldsymbol{\Pi}$ of P systems with active membranes using object evolution and communication rules that simulates M in space $O(\log n)$ and time $O(t(n)s(n))$.*

Proof. (sketch) Let $x \in \Sigma^n$ be an input string, and let $m = \lceil \log s(n) \rceil$ be the minimum number of bits needed to represent the tape cell indices $0, \ldots, s(n)-1$ in binary notation. The P system Π_n, associated with the input length n, has a membrane structure consisting of an external skin membrane that contains, for each symbol of the tape alphabet of M, the following set of membranes, linearly nested and listed from the outside in:

- a *symbol-membrane*;
- a *query-membrane*;
- for each $j \in \{0, \ldots, m-1\}$, a membrane labelled by j_σ.

An arbitrary configuration of M on input x is encoded by a configuration of Π_x as follows:

- the outermost membrane contains the *state-object* q_i, (where q is the current state of M, and i is the current tape head position);
- if membrane $(m-1)_\sigma$ contains the input object τ_i, then the i-th tape cell of M contains the symbol σ.

The symbol written on the i-th tape cell of M can thus be inferred from the label of the substructure which contains the corresponding input symbol τ_i. Notice that a logarithmic depth membrane structure allows to represent a polynomial number of possible head positions.

The state-object q_i queries each membrane substructure, which encodes in binary the tape position i on the electrical charges of the membranes. Only the symbol whose subscript is i can reach the skin membrane and be used to conclude the simulation of a computation step.

The family $\boldsymbol{\Pi}$ described above is $(\mathbf{L}, \mathbf{L})$-uniform, and each P system Π_x uses only a logarithmic number of membranes and a constant number of objects per configuration, besides the input objects, which are never rewritten. Π_x works in space $O(\log n)$ and in time $O(t(n)s(n))$. □

As a consequence, we have the following:

Theorem 7. *For each class $\mathcal{D} \subseteq \mathcal{AM}$ of P systems with active membranes using object evolution and communication among their rules we have*

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}} = \mathbf{PSPACE}.$$

Proof. The inclusion $\mathbf{PSPACE} \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ follows immediately from Theorem 6. By definition, the class $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ is included in $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$. Finally, to prove the inclusion of $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$ in **PSPACE** it suffices to simulate P systems by means of Turing machines, which can be carried out with just a polynomial space overhead, as shown in [17, 1]. □

This was the first case where the space complexity of P systems and that of Turing machines differ by an exponential amount. Since, as previously said, **PSPACE** had already been proved to be characterised by *polynomial*-space P systems, these results also highlight a gap in the hierarchy of space complexity classes for P systems: super-polynomial space is required in order to exceed the computational power of logarithmic space.

6 Constant–Space P systems

After considering P systems with active membranes working in logarithmic space, a natural question arises concerning the power of such systems using only a constant amount of space. Surprisingly, it turned out that constant space is sufficient to simulate polynomial-space bounded deterministic Turing machines, as proved in [9]:

Theorem 8. $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(1)) = \mathbf{PSPACE}$.

Proof. (sketch) Let $L \in \mathbf{PSPACE}$, and let M be a Turing machine deciding L in space $p(n)$. We can construct a family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ such that $L(\boldsymbol{\Pi}) = L$ by letting $F(1^n) = \Pi_n$, where Π_n is the P system simulating M on inputs of length n, and

$$E(x_0 \cdots x_{n-1}) = x_{1,1} \cdots x_{n-1,n-1} \sqcup_n \cdots \sqcup_{p(n)-1},$$

i.e. by padding the input string x with $p(n) - n$ blank symbols $\sqcup$ before indexing the result with the positions of the symbols on the tape.

The simulation relies on two main ideas. As in the previous proof of Theorem 6, input objects of the form τ_i are distributed in substructures, and the symbol written on the i-th tape cell of M can be inferred from the label of the substructure where the corresponding input symbol τ_i is placed. The second idea is that it is possible to "read" a subscript of an input object τ_i without rewriting it and by using only a constant number of additional objects and membranes: in particular, a timer object is used to change the charge of a membrane after exactly i steps. Any other object that counts together with the timer is able to keep track of the number of elapsed steps, obtaining in this way the value i.

Since at each computation step only a constant number of working objects and membranes are present, then the simulation requires, according to definition 4, a constant amount of space. Moreover, both functions F and E can be computed in logarithmic space by Turing machines, since they only require adding subscripts having a logarithmic number of bits to rules or strings having a fixed structure, and the membrane structure is fixed for all Π_n. This proves the inclusion of **PSPACE** in $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$, while the reverse inclusion is proved in [17]. □

7 Rethinking the Definition of Space

The result of Theorem 8 shows that all problems in **PSPACE** can be solved by constant-space P systems with active membranes. This rises some natural questions about the definition of space complexity for P systems adopted before the appearance of [9]. Does counting each non-input object and each membrane as unitary space really capture an intuitive notion of the amount of space used by a P system during a computation? Is it fair to allow a polynomial padding of the input string when encoding it as a multiset?

In [9], it was highlighted that the constant number of non-input objects appearing in each configuration of the simulation actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet Γ of polynomial size. According to the original definition of space recalled in Section 2, each of these objects would only require unitary space, whereas the binary representation of the subscript i requires $\log p(n) = \Theta(\log n)$ bits. It may be argued that this amount of information needs a proportional amount of physical storage space. Similarly, each membrane label contains $\Theta(\log |\Lambda|)$ bits of information, which must also have a physical counterpart.

The information stored in the *positions* of the objects within the membrane structure is also not taken into account by Definition 4. However, the information on the location of the objects is part of the system and it is *not* stored elsewhere, exactly as the information on the location of the tape head in a Turing machine, which is not counted as space.

Due to the above considerations, in [9] an alternative definition of space was proposed:

Definition 5. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the number of membranes in the current membrane structure* multiplied by $\log |\Lambda|$, *plus the total number of objects from Γ (i.e, the non-input objects) they contain* multiplied by $\log |\Gamma|$.

Adopting this stricter definition does not significantly change space complexity results involving polynomial or larger upper bounds, i.e., the complexity classes $\mathbf{PMCSPACE}_{\mathcal{AM}}$, $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$, and larger ones remain unchanged.

As for padding the input string, one may argue that this operation provides the P system with some "free" storage, since input objects are not counted by Definition 4. The proof of Theorem 8 exploits the ability to encode an input string of length n as a polynomially larger multiset in a substantial way, as allowed by the most common uniformity conditions, including **P** and **LOGSPACE**-uniformity, but also weaker ones such as $\mathbf{AC}^0$ or **DLOGTIME**-uniformity.

The simulation described in the previous section would require logarithmic space according to Definition 5. Also the space bounds of the simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes described in Section 5 also increase to $\Theta(\log n \log\log n)$, since in that case each configuration of the P systems contains $\Theta(\log n)$ membranes with distinct labels and $O(1)$ non-input objects. Both simulations would be limited to *linear*-space Turing machines, rather than polynomial-space ones, if input padding were disallowed.

8 Final Remarks

In this paper we gave a survey on recent results concerning complexity of P systems with active membranes. The results showed that such P systems can be simulated by Turing machines with only a polynomial increase in space requirements and, moreover, that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. This leads to prove equalities among space complexity classes for P systems and Turing machines (as long as the sets of space bounds satisfy some reasonable properties). In particular, complexity classes defined in terms of polynomial, exponential, double exponential, ..., n-fold exponential space coincide for the two kinds of device.

Further, it has been shown that the class **PSPACE** can also be characterized by P systems with active membranes using logarithmic space or even a constant

amount of space. In view of the last result, a new definition of space for P systems has been proposed, that also takes into account the number of bits necessary to encode the non-input objects and the labels of the membranes. While the new definition does not change any result involving an amount of space which is polynomial or larger, it changes the results involving sublinear amounts of space. In particular, according to the new definition the simulation used to prove that constant space P systems with active membranes characterize **PSPACE** would now require logarithmic space.

Various problems can be considered in this framework and are still to be investigated. In general, they are related to features of the systems proposed to approach the different complexity classes: apart from time and space resources required by those systems, we can consider uniform versus semi–uniform conditions, confluent versus deterministic systems, deep membrane structures versus shallow membrane structures, whether or not is necessary to use features such as polarization or dissolution, and restrictions on the communication protocols (one way, limited communication, etc.). Another interesting recent topic, related to space in a broader sense, concerns the power of different (Tissue) P systems when considering bounds on the cell volume, or the shape of the cell structure embedded in the Euclidean space. We refer the reader to [7] and [8] for details.

References

1. A. Alhazov, A. Leporati, G. Mauri, A.E. Porreca, C. Zandron, Space complexity equivalence of P systems with active membranes and Turing machines, Theoretical Computer Science, 529, 2014, 69–81.
2. E. Csuhaj-Varju, M. Oswald, Gy. Vaszil, P automata, Handbook of Membrane Computing. Gh. Paun et al. (Eds.), Oxford University Press, 2010, 144–167.
3. M. Gheorghe, Gh. Păun, M.J. Perez-Jimenez, G. Rozenberg, Frontiers of membrane computing: Open problems and research topics. Intern. J. Found. Computer Sci. 24(5), 2013.
4. S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, Romanian J. of Information Science and Technology, 2, 4, 1999, 357–367.
5. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Simulating elementary active membranes with an application to the P conjecture, In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.), CMC 2014, LNCS 8961, Springer Heidelberg, 2014, 284–299.
6. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae, 138(1-2), 2015, 97–111.
7. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Trading Geometric Realism for Efficiency in Tissue P Systems, Romanian Journal of Information Science and Technology, 19(1-2), 2016, 17–30.
8. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Tissue P Systems with Small Cell Volume. Fundamenta Informaticae, 154(1-4), 2017, 261–275.
9. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Constant–space P systems with Active Membranes, Fundamenta Informaticae, Fundamenta Informaticae, 134 (1-2), 2014, 111–128.

10. A. Leporati, G. Mauri, A.E. Porreca, C. Zandron, A Gap in the space hierarchy of P systems with active membranes, Journal of Automata, Languages and Combinatorics, 19 (1–4), 2014, 173–184.
11. D.A. Mix Barrington, N. Immerman, H. Straubing, On uniformity within NC^1. Journal of Computer and System Sciences 41(3), 1990, 274–306.
12. N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Natural Computing 10(1), 2011, 613–632.
13. Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1993.
14. Gh. Păun, P systems with active membranes: Attacking NP-complete problems, Journal of Automata, Languages and Combinatorics 6(1), 2001, 75–90.
15. Gh. Păun, G. Rozenberg and A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, 2010.
16. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Introducing a space complexity measure for P systems, International Journal of Computing, Communication & Control 4(3), 2009, 301–310.
17. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P Systems with Active Membranes: Trading Time for Space, Natural Computing 10(1), 2011, 167–182.
18. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with active membranes working in polynomial space, International Journal of Foundations of Computer Science, 22(1), 2011, 65–73.
19. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems simulating oracle computations. In: Gheorghe, M., Păun, Gh., Salomaa, A., Rozenberg, G., Verlan, S. (eds.) Membrane Computing, 12th International Conference, CMC 2011, Lecture Notes in Computer Science, vol. 7184, Springer, 2012, 346–358.
20. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Sublinear Space P systems with Active Membranes, Membrane Computing: 13th International Conference, CMC 2012, LNCS 7762, Springer, Berlin, 2013, 342–357.
21. A.E. Porreca, G. Mauri, C. Zandron, Complexity classes for membrane systems, RAIRO-Theoretical Informatics and Applications, 40(2), 2006, 141–162.
22. A.E. Porreca, G. Mauri, C. Zandron, Non-confluence in divisionless P systems with active membranes, Theoretical Computer Science, 411 (6), 2010, 878–887.
23. P. Sosik, A. Rodriguez-Paton, Membrane computing and complexity theory: A characterization of PSPACE, Journal of Computer and System Sciences, 73, 2007, 137–152.
24. C. Zandron, C. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, Proceedings of the Second International Conference on Unconventional Models of Computation, Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.), UMC'2K, Springer, 2001, 289–301.
25. C. Zandron, A. Leporati, C. Ferretti, G. Mauri, M.J. Perez-Jimenez, On the Computational Efficiency of Polarizationless Recognizer P Systems with Strong Division and Dissolution, Fundamenta Informaticae, 87(1), 2008, 79–91.