P Systems with Elementary Active Membranes: Beyond NP and coNP

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{porreca,leporati,mauri,zandron}@disco.unimib.it

Abstract. We prove that a uniform family of P systems with active membranes, where division rules only operate on elementary membranes and dissolution rules are avoided, can be used to solve the following **PP**-complete decision problem in polynomial time: given a Boolean formula of m variables in 3CNF, do at least $\sqrt{2^m}$ among the 2^m possible truth assignments satisfy it? As a consequence, the inclusion $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ holds: this provides an improved lower bound on the class of languages decidable by this kind of P systems.

1 Introduction

P systems with active membranes [11] are a variant of P systems where a particularly important role in the computation is performed by the membranes themselves: they possess an electrical charge that can inhibit or activate the rules that govern the evolution of the system, and they can also increase exponentially in number via division rules. The latter feature makes them extremely efficient from a computational complexity standpoint: using exponentially many membranes that evolve in parallel, they can be used to solve **PSPACE**-complete problems [12, 2] in polynomial time.

When the ability of dividing membranes is limited, the efficiency apparently decreases. The so-called Milano theorem [14] tells us that no **NP**-complete problem can be solved in polynomial time without using division rules, unless $\mathbf{P} = \mathbf{NP}$ holds.

On the other hand, the computing power of polynomial-time P systems with division rules operating only on *elementary* membranes (that is, membranes not containing other membranes) has not been yet characterised precisely. It is a known fact that elementary division rules suffice to efficiently solve **NP**-complete problems (and, due to closure under complement, also **coNP**-complete ones). This result dates back to 2000 in the semi-uniform case [14], where each input is mapped to a specific P system solving the problem for that particular input, and to 2003 in the uniform case [9], where a single P system solves the problem for all inputs of the same size. In terms of complexity classes, this is written $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})} \subseteq \mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{n})}$, where the star denotes

semi-uniformity. Since these results do not require membrane dissolution rules, we also have the (possibly stronger) inclusion $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$; the systems of type $\mathcal{AM}(-\mathrm{d},-\mathrm{n})$ are sometimes called *P systems with restricted elementary active membranes* [2].

No significant improvement on the $\mathbf{NP} \cup \mathbf{coNP}$ lower bound for the complexity classes $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ and $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$, or the corresponding semi-uniform classes, has been found since then, although a **PSPACE** upper bound was proved in 2007 [13].

In 2008, Alhazov et al. [1] proved that P systems with elementary active membranes can be used to solve **PP**-complete problems, but their result is not directly related to $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$, since it requires either cooperative evolution rules, a very strong feature which is not a part of standard P systems with active membranes, or post-processing data of exponential size (when expressed in unary).

The complexity class **PP** appears to be larger than **NP**, since it contains **NP** as a subset and it is closed under complement: thus $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PP}$. In this paper we prove that a **PP**-complete problem (and, as a consequence, the totality of problems in **PP**) can indeed be solved in polynomial time using standard P systems with restricted elementary active membranes.

2 Definitions

We begin by recalling the definition of P systems with restricted elementary active membranes.

Definition 1. *A* P system with restricted elementary active membranes [2]*, in symbols* $\mathcal{AM}(-\mathrm{d},-\mathrm{n})$*, of the initial degree* $d \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$$

where:

- Γ *is a finite alphabet of symbols, also called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted unordered tree) consisting of* d *membranes enumerated by* $1, \ldots, d$*; furthermore, each membrane is labelled by an element of* Λ*, not necessarily in a one-to-one way;*
- $w_1, \ldots, w_d$ *are strings over* Γ*, describing the multisets of objects placed in the* d *initial regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses a further attribute, named *polarization* or *electrical charge*, which is either neutral (represented by 0), positive (+) or negative (−) and it is assumed to be initially neutral.

The rules are of the following kinds:

- *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 multiset w).
- *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.
- *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.
- *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; the membrane is divided into two membranes having label h and charge 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.

A *configuration* in a P system with active membranes is described by its current membrane structure, together with its charges and the multisets of objects contained in its regions. The *initial configuration* is given by μ, all membranes having charge 0 and the initial contents of the membranes being $w_1, \ldots, w_m$. A *computation step* changes the current configuration according to the following principles:

- Each object and each membrane can be subject to only one rule during a computation step.
- The rules are applied in a *maximally parallel way*: each object which appears on the left-hand side of applicable evolution, communication, or elementary division rules must be subject to exactly one of them; the same holds for each membrane which can be involved in a communication or division rule. The only objects and membranes which remain unchanged are those associated with no rule, or with rules that are not applicable in that particular step (due to the charge of the membrane).
- When more than one rule can be applied to an object or membrane, the actual rule to be applied is chosen nondeterministically; hence, in general, multiple configurations can be reached from the current one.
- When division rules are applied to a membrane, the multiset of objects to be copied is the one resulting *after* all evolution rules have been applied.
- The skin membrane cannot be divided. Furthermore, every object which is sent out from the skin membrane cannot be brought in again.

A *halting computation* $\boldsymbol{\mathcal{C}}$ of a P system Π is a finite sequence of configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration of Π, every $\mathcal{C}_{i+1}$ can be reached

from $\mathcal{C}_i$ according to the principles just described, and no further configuration can be reached from $\mathcal{C}_k$ (i.e., no rule can be applied). P systems might also perform *non-halting* computations; in this case, we have infinite sequences $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ of successive configurations.

We can use families of P systems with active membranes as language recognisers, thus allowing us to solve decision problems.

Definition 2. *A* recogniser P system with active membranes Π *has an alphabet containing two distinguished objects* YES *and* NO*, used to signal acceptance and rejection respectively; every computation of* Π *is halting and exactly one object among* YES, NO *is sent out from the skin membrane during each computation.*

In what follows we will only consider *confluent* recogniser P systems with active membranes, in which all computations starting from the same initial configuration agree on the result.

Definition 3. *Let* $L \subseteq \Sigma^\star$ *be a language and let* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *be a family of recogniser P systems. We say that* $\boldsymbol{\Pi}$ decides L*, in symbols* $L(\boldsymbol{\Pi}) = L$*, when for each* $x \in \Sigma^\star$*, the result of* Π_x *is acceptance iff* $x \in L$*.*

Usually some uniformity condition, inspired by those applied to families of Boolean circuits, is imposed on families of P systems. Two different notions of uniformity have been considered in the literature; they are defined as follows.

Definition 4. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* semi-uniform *when the mapping* $x \mapsto \Pi_x$ *can be computed in polynomial time, with respect to* $|x|$*, by a deterministic Turing machine.*

Definition 5. *A family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to be* uniform *when there exist two polynomial-time deterministic Turing machines* M_1 *and* M_2 *such that, for each* $n \in \mathbb{N}$ *and each* $x \in \Sigma^n$

- M_1*, on input* 1^n *(the unary representation of the length of* x*), outputs the description of a P system* Π_n *with a distinguished input membrane;*
- M_2*, on input* x*, outputs a multiset* w_x *(an encoding of* x*);*
- Π_x *is* Π_n *with* w_x *added to the multiset located inside its input membrane.*

In other words, the P system Π_x *associated with string* x *consists of two parts; one of them,* Π_n*, is common for all strings of length* $|x| = n$ *(in particular, the membrane structure and the set of rules fall into this category), and the other (the input multiset* w_x *for* Π_n*) is specific to* x*. The two parts are constructed independently and, only as the last step,* w_x *is inserted in* Π_n*.*

Time complexity classes for P systems [9] are defined as usual, by restricting the amount of time available for deciding a language. By $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ (resp., $\mathbf{PMC}^\star_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$) we denote the class of languages which can be decided by uniform (resp., semi-uniform) families $\boldsymbol{\Pi}$ of confluent P systems with restricted elementary active membranes where each computation of $\Pi_x \in \boldsymbol{\Pi}$ halts in polynomial time with respect to $|x|$. These classes are known to be closed

under complement and polynomial-time reductions. Since uniformity is a special case of semi-uniformity, the inclusion $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} \subseteq \mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ holds; we only consider uniform families in the rest of the paper.

The complexity class **PP** (Probabilistic **P**) was first introduced to characterise those decision problems which can be solved efficiently by a probabilistic Turing machine, whose probability of error on every input is strictly less than 1/2 [6]. An equivalent definition of **PP** is usually given in terms of nondeterministic Turing machines by altering the notion of acceptance [8].

Definition 6. *The complexity class* **PP** *consists of all languages $L \subseteq \Sigma^{\star}$ which can be decided in polynomial time by a nondeterministic Turing machine N with the following acceptance criterion: N accepts $x \in \Sigma^{\star}$ iff more than half of the computations of N on input x are accepting.*

3 Solving a PP-Complete Problem

One of the standard **PP**-complete problems is Majority-SAT [4, 8]: given a Boolean formula φ of m variables in conjunctive normal form, determine whether more than half of the 2^m possible truth assignments satisfy it. However, it is not easy to provide a polynomial-time *uniform* solution for this problem, since the clauses of φ may contain any number of literals between 1 and $2m$. The usual solution in membrane computing is to require the input formula to have exactly three different literals per clause (see, e.g., [10]). Unfortunately, the resulting decision problem Majority-3SAT is not known to be **PP**-complete. In particular, the standard reduction from SAT to 3SAT [5] is not applicable here, as it requires the addition of "dummy" variables, which increase the number of possible assignments without necessarily increasing the number of satisfying ones: this can decrease the ratio of satisfying assignments over total assignments from above 1/2 to a value less than or equal to this threshold.

There is, however, yet another slight variation of the problem that is suitable for our purposes.

Definition 7. Sqrt-3SAT[1] *is the following decision problem: given a Boolean formula of m variables in 3CNF, determine whether the number of truth assignments satisfying it is at least $\sqrt{2^m}$.*

The problem Sqrt-3SAT is known to be **PP**-complete [3], and it is very close in spirit to Majority-3SAT. Our solution to this problem follows the canon for **NP**-complete problems in membrane computing [11, 14], but with an additional intermediate phase (numbered 3 in the following algorithm).

Algorithm 1. *Solving* Sqrt-3SAT *on input φ, a 3CNF formula of m variables.*

1. *Generate 2^m membranes using elementary division, each one containing a different truth assignment to the variables occurring in φ.*

[1] This problem is denoted by $\#3\mathrm{SAT}(\geq 2^{m/2})$ in the original paper [3].

2. *Evaluate φ under the 2^m assignments, in parallel, and send out from each membrane an object t whenever the formula is satisfied by the corresponding assignment.*
3. Erase $\lceil\sqrt{2^m}\rceil - 1$ instances of t (or all of them, if less than $\lceil\sqrt{2^m}\rceil - 1$ occur).
4. *Output* YES *if at least one instance of t remains; otherwise, output* NO.

Notice that, by removing Phase 3, we obtain the standard membrane computing algorithm for SAT. The additional phase was first proposed by Alhazov et al. [1] for checking the value of the permanent of a matrix, but the authors used cooperative object evolution rules, that are not part of standard P systems with active membranes. In Section 3.2 we show how to implement this phase using elementary division and communication rules, together with all the other steps of Algorithm 1.

3.1 Encoding of Formulae

Formulae in 3CNF are easy to encode as binary strings [7, 10]. Given m variables, only $8\binom{m}{3}$ clauses without repeated variables exist: we have $\binom{m}{3}$ sets of three out of m variables, and each one of them can be either positive or negated. Once an easily-computable enumeration of the clauses has been fixed (e.g., under a lexicographic order, the i-th clause can be computed from i in polynomial time) a formula φ can be represented by a string $\langle\varphi\rangle$ of $n = 8\binom{m}{3}$ bits, where the i-th bit is set iff the i-th clause occurs in φ.

Under this encoding, a string in $\{0,1\}^n$ is a valid formula iff $n = 8\binom{m}{3}$ for some integer $m \geq 3$. The number of variables m can be easily recovered in polynomial time, given n in unary notation, by finding the unique positive integer root of the polynomial $p(m) = 8\binom{m}{3} - n = \frac{4}{3}m^3 - 4m^2 + \frac{8}{3}m - n$. If no such root exists, we can deduce that the input is not well-formed with respect to our encoding.

3.2 Solution to Sqrt-3SAT

The implementation of Algorithm 1 is a uniform variant of the solution described by Zandron et al. [14]. To all strings $x \in \{0,1\}^n$ with $n = 8\binom{m}{3}$, representing Boolean formulae φ of m variables, we associate a P system with restricted elementary active membranes Π_n. The initial configuration of Π_n (excluding the input multiset) is the following one:

$$\mathcal{C}_0 = \left[q_0 r_0 \;\; [p_0 x_1 x_2 \cdots x_m]_1^0 \;\; [b_{i_1}]_2^0 [b_{i_2}]_2^0 \cdots [b_{i_h}]_2^0 \right]_0^0$$

Here the objects $x_1, \ldots, x_m$ represent the variables of φ, while p_0, q_0, and r_0 are objects used to implement three timers, counting from zero.

The number of membranes having label 2 and their contents are determined as follows. Let $k = \lceil\sqrt{2^m}\rceil - 1$, and consider the binary representation of k: for each $i = 0, \ldots, \lfloor \log k \rfloor$, if the i-th least significant bit of k is 1, then we add to $\mathcal{C}_0$ a copy of membrane 2, containing the single object b_i; otherwise we

add nothing. In other words, h and $i_1 < i_2 < \cdots < i_h$ are the unique integers such that $k = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_h}$. Clearly h is bounded by k, which is in turn bounded by $\frac{m}{2}$; hence the configuration $\mathcal{C}_0$ can be costructed in polynomial time with respect to n.

The input multiset, obtained from $\langle\varphi\rangle$, is placed inside membrane 1, and contains all the objects c_i such that the i-th clause does *not* occur in φ.

For instance, suppose $m = 3$, hence $n = 8\binom{3}{3} = 8$. The eight (up to reordering of literals) clauses over three variables x_1, x_2, x_3 can be enumerated as

$$\begin{array}{llll} x_1 \vee x_2 \vee x_3 & x_1 \vee x_2 \vee \neg x_3 & x_1 \vee \neg x_2 \vee x_3 & x_1 \vee \neg x_2 \vee \neg x_3 \\ \neg x_1 \vee x_2 \vee x_3 & \neg x_1 \vee x_2 \vee \neg x_3 & \neg x_1 \vee \neg x_2 \vee x_3 & \neg x_1 \vee \neg x_2 \vee \neg x_3 \end{array}$$

and the formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ is then encoded as $\langle\varphi\rangle = 0010\ 0110$. The corresponding input multiset is $c_1c_2c_4c_5c_8$.

Starting from the initial configuration, including the input multiset, the computation proceeds as follows.

Phase 1 (Generate). Each variable object x_i is used to divide membrane 1, and is replaced by a "true" object t_i on one side, and by a "false" object f_i on the other, denoting the two possible truth values that can be assigned to variable x_i. The corresponding rules are

$$[x_i]_1^0 \to [t_i]_1^0[f_i]_1^0 \qquad \text{for } 1 \le i \le m.$$

While the membranes having label 1 divide, thus generating 2^m copies (each one containing a different assignment), the timer p_0 is incremented, one step at a time, up to m:

$$[p_j \to p_{j+1}]_1^0 \qquad \text{for } 0 \le j \le m-1.$$

The object p_m is then sent out of membrane 1, while changing the charge of the membrane to positive, using the rule $[p_m]_1^0 \to [\ ]_1^+ p_m$.

The object p_m is immediately brought back in (and renamed to u_0) via $p_m\ [\ ]_1^+ \to [u_0]_1^+$. Simultaneously, each object t_i and f_i is replaced by a set of objects denoting the clauses that are satisfied when the variable x_i is true or false, respectively, i.e.:

$$\begin{array}{ll} [t_i \to c_{i_1} \cdots c_{i_s}]_1^+ & \text{for } 1 \le i \le m \text{ and } x_i \text{ occurs in clauses } i_1, \ldots, i_s; \\ [f_i \to c_{i_1} \cdots c_{i_s}]_1^+ & \text{for } 1 \le i \le m \text{ and } \overline{x}_i \text{ occurs in clauses } i_1, \ldots, i_s. \end{array}$$

Notice that computing these sets of clauses does *not* require the input formula φ, but only its size n (this is consistent with a uniform construction). The clause-objects c_j are produced in the $(m+2)$-th step.

While all these events described above occur inside the membranes labelled by 1, we also divide the membranes with label 2 until we have $\lceil\sqrt{2^m}\rceil - 1$ copies. Indeed, each object b_i is used to create 2^i copies, according to the following rules:

$$[b_i]_2^0 \to [b_{i-1}]_2^0[b_{i-1}]_2^0 \qquad \text{for } 1 \le i \le \lfloor \log k \rfloor.$$

Producing all copies of membrane 2 requires a number of steps bounded by

$$\lfloor \log k \rfloor = \lfloor \log(\lceil \sqrt{2^m} \rceil - 1) \rfloor \leq \log \lceil \sqrt{2^m} \rceil \leq \frac{m}{2} + 1 \leq m + 2.$$

Hence, Phase 1 requires a total of $m + 2$ steps.

Phase 2 (Evaluate). In this phase, the object u_j inside each copy of membrane 1 behaves as a counter for the number of satisfied clauses, and initially it has the value u_0.

Now consider the contents of the membranes having label 1. If the i-th clause occurs in φ *and* it is satisfied by the truth assignment corresponding to the particular copy of membrane 1 under consideration, then one or more instances of object c_i have been generated in Phase 1. If this clause does *not* occur in φ, then the object c_i has been placed in membrane 1 as part of the input multiset: the clause is then considered to be satisfied[2]. Finally, if this clause *does* occur in φ but it is not satisfied, then no instance of c_i occurs inside membrane 1.

We find out whether the clauses are satisfied, one by one in the order established in Section 3.1, by checking whether an instance of the object c_1 occurs, then decrementing the subscript of all the other objects c_j by one; this procedure is repeated until an unsatisfied clause is found, or all of them are found to be satisfied.

If c_1 does indeed occur, then it is sent out and changes the charge of 1 to negative, using the rule $[c_1]_1^+ \to [\,]_1^-\ c_1$. While membrane 1 is negative, the other subscripts are decremented:

$$[c_j \to c_{j-1}]_1^- \qquad \text{for } 2 \leq j \leq n.$$

Simultaneously, u_j increments its subscript via

$$[u_j \to u_{j+1}]_1^- \qquad \text{for } 0 \leq j < n,$$

and c_1 re-enters membrane 1 (not necessarily the same instance of membrane 1, but any negatively charged one) as the "junk" object #, and sets its charge back to positive via $c_1\ [\,]_1^- \to [\#]_1^+$.

Phase 2 now restarts, with all clause-objects having their subscript decremented by one. If one of the objects c_j is missing for some $j = 1, \ldots, n$, then the computation in that copy of membrane 1 halts prematurely, and u_n is never reached. On the other hand, if all objects $c_1, \ldots, c_n$ exist inside a certain copy of membrane 1, the object u_n is reached in $2n$ steps: we can then conclude that the formula is completely satisfied, and send out a t object to signal it, using the rule $[u_n]_1^+ \to [\,]_1^+\ t$. Notice that all t objects are sent out simultaneously from all copies of membrane 1.

The total number of steps required for Phase 2 is $2n + 1 = 16\binom{m}{3} + 1$.

Phase 3 (Erase). When the instances of objects t reach the skin membrane, labelled by 0, each copy of membrane 2 absorbs one of them, if any is available,

[2] This is consistent with the "true" value being the identity of conjunction.

using the communication rule $t\,[\,]_2^0 \to [\#]_2^+$. After this computation step, one or more copies of t remain inside membrane 0 iff the number of instances of t was at least $\sqrt{2^m}$, that is, iff φ is a positive instance of SQRT-3SAT.

Phase 4 (Output). The sequences of objects q_j and r_j, which begin with q_0 and r_0 and whose behaviour we have not described yet, are meant to count the number of steps across Phases 1, 2, and 3, that is, $\ell = (m+2) + \left(16\binom{m}{3} + 1\right) + 1$. This is accomplished by using the following evolution rules:

$$[q_j \to q_{j+1}]_0^0 \qquad \text{for } 0 \le j \le \ell;$$
$$[r_j \to r_{j+1}]_0^\alpha \qquad \text{for } 0 \le j \le \ell + 2 \text{ and } \alpha \in \{+, 0, -\}.$$

When the subscript of q reaches ℓ, Phase 3 has just finished. This object is sent out in order to change the charge of membrane 0 to positive, using the rule $[q_\ell]_0^0 \to [\,]_0^+ \#$; this enables any remaining instance of t inside membrane 0 to exit and change again the charge of the skin to negative, using the rule $[t]_0^+ \to [\,]_0^- \#$. If no object t exists inside membrane 0, the charge remains positive.

During the next computation step, the subscript of r is $\ell + 2$, and this object is finally sent out, either as YES or NO depending on the charge of membrane 0:

$$[r_{\ell+2}]_0^+ \to [\,]_0^+ \text{ NO} \qquad [r_{\ell+2}]_0^- \to [\,]_0^- \text{ YES}.$$

According to the argument above, the object emerging from membrane 0 corresponds to the correct answer to the problem.

The sizes of the sets of rules (hence, also the size of the alphabet) described in each step of the algorithm are clearly bounded by a polynomial in n and computable efficiently from 1^n; thus, we can conclude that SQRT-3SAT has a polynomial time uniform solution.

Theorem 1. SQRT-3SAT $\in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$, *hence* $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ *via polynomial-time reductions.* □

4 Conclusions

We improved one of the earliest results related to P systems with active membranes, namely that elementary division is sufficient to solve **NP**-complete problems in polynomial time, by proving that **PP** problems can also be solved efficiently by the same class of P systems, without the need for dissolution rules. The method is a generalisation of the classic membrane computing algorithm schema for **NP**-complete problems, where all candidate solutions are generated and then tested in parallel.

This result does still not provide, however, a characterisation of the complexity classes $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ and $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ in terms of Turing machines; furthermore, neither the **PP** lower bound, nor the **PSPACE** upper bound are known to be strict. We think that the question is worth further investigation, with the goal of finally establishing whether nonelementary division rules are a redundant feature of P systems with active membranes, or a fundamental one.

References

1. Alhazov, A., Burtseva, L., Cojocaru, S., Rogozhin, Y.: Solving PP-complete and #P-complete problems by P systems with active membranes. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 9th International Workshop, WMC 2008, Edinburgh, UK, July 28–31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5391, pp. 108–117. Springer (2009)
2. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
3. Bailey, D.D., Dalmau, V., Kolaitis, P.G.: Phase transitions of PP-complete satisfiability problems. Discrete Applied Mathematics 155(12), 1627–1639 (2007)
4. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity I. Springer, 2nd edn. (1995)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co. (1979)
6. Gill, J.T.: Computational complexity of probabilistic Turing machines. In: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 91–95 (1974)
7. Lagoudakis, M.G., LaBean, T.H.: 2D DNA self-assembly for satisfiability. In: Winfree, E., Gifford, D.K. (eds.) DNA Based Computers V. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54, pp. 139–152 (1999)
8. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
9. Pérez-Jiménez, M.J., Romero Jiménez, A., Sancho Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
10. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing, in press. DOI: 10.1007/s11047-010-9189-x
11. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
12. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
13. 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)
14. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K: Proceedings of the Second International Conference, pp. 289–301. Discrete Mathematics and Theoretical Computer Science, Springer (2001)