Membrane dissolution and division in P

Damien Woods[1], Niall Murphy[2], Mario J. Pérez–Jiménez[1], Agustín Riscos–Núñez[1]

[1] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Seville, Spain
[2] Department of Computer Science, National University of Ireland Maynooth, Ireland

Abstract Membrane systems with dividing and dissolving membranes are known to solve **PSPACE** problems in polynomial time. However, we give a **P** upperbound on an important restriction of such systems. In particular we examine systems with dissolution, elementary division and where each membrane initially has at most one child membrane. Even though such systems may create exponentially many membranes, each with different contents, we show that their power is upperbounded by **P**.

1 Introduction

Bacteria and other cells provide an interesting avenue to consider in Natural Computing. Their native ability to make copies of themselves inspires us with visions of exponentially multiplying processors churning away, solving intractable problems. But can cells actually solve such hard problems? To answer this we should abstract away from the mind-boggling intricacy of a living cell and create a model that strips away everything but the features that we want to analyse.

The model we investigate is called active membrane systems [9], a variant of P systems [10]. Active membrane systems were introduced to analyse the computational complexity of nested membranes that have the ability to divide and dissolve. It was shown by Sosík [11] that the model was extremely powerful and could solve problems as quickly as parallel machines that satisfy the parallel computation thesis. Later Sosík and Rodríguez-Patón showed [12] that the model was no more powerful than such parallel machines. This result holds for a very general definition of active membrane systems and so can be interpreted as an upperbound on systems with dividing and dissolving membranes.

These results used non-elementary division, that is, when a membrane divides, all sub-membranes are copied. For several years a tighter upperbound (or lowerbound) has been sought for the model where only membranes with no sub-membranes may divide. This would close an open problem known as the P conjecture which states that active membrane systems without charges characterise **P** [8].

However, the proof has been elusive (see Mauri et al. [3] for a more detailed survey). While attempting to answer this conjecture it was discovered that membrane dissolution was a powerful aspect of the system when it was found that without dissolution the system was upperbounded by **P** [1]. It is also known that systems without division rules are upperbounded by **P** [13]. This highlights that membrane dissolution when combined with division are the most difficult aspects of the model.

A restriction of the model that uses symmetric cell division, where the two resulting membranes are identical, was shown to solve no more than **P** [5]. In this paper we analyse the more general division rule where two resulting membranes of a division may be different (in a manner similar to the mechanism of stem cells) combined with dissolution rules.

Our result gives a **P** upperbound on systems with dissolution and elementary division, where at the initial timestep, the depth of membrane nesting is equal to the total number of membranes. In the notation of membrane systems, this may be stated as follows.

Theorem 1. $\mathbf{PMC}^*_{\mathcal{D}} \subseteq \mathbf{P}$*, where* $\mathcal{D}$ *is the class of systems in* $\mathcal{AM}^0_{+d,-ne,-e,-c}$ *having an initial membrane structure that is a single path.*

While all the machinery is in place for a proof for a membrane structure that (initially) is a tree, we leave the full proof to a more complete version of the paper and only consider here the case where the structure is initially a single path.

Our simulation algorithm does not explicitly simulate all (of the exponential number of) membranes. Even though these systems may create exponentially many membranes, each with different contents, we observe that the elementary membranes are only important to simulate if they release a dissolution object to a parent membrane, or some other ancestor.

To prove our main result we use a number of simulation techniques. The main novelties in our approach are the following. We carefully choose a computation to simulate: the membrane systems we simulate are confluent (nondeterministic, but all computations produce the same answer), and our technique finds a sufficiently simple computation to simulate out of a large set of more complicated computations. Also, we model important aspects of the computation using special *division graphs* that are inspired by, and a generalisation of, dependency graphs. Finally, our simulation algorithm carefully selects a small number of important membranes to explicitly simulate, while ignoring up to exponentially many other membranes. This technique is nicer than a brute force approach as it highlights which aspects of the computation actually influence the answer, and which do not.

Although our main motivation is the P conjecture, our work has applications for membrane system simulators. For previous **P** upperbounds, this point could perhaps be more difficult to argue since such upperbounds on membrane systems often have large hidden constants. However given a membrane system that creates an exponential number of membranes, our algorithm actually explicitly simulates very few membranes (less than double the number of membranes that are in the initial configuration).

Unlike some previous techniques, we feel that the present approach will be fruitful for proving the P conjecture for systems that have dissolution, elementary division, evolution and communication rules. Our technique seems suitable for generalisation, and does not suffer from a combinatorial explosion of cases.

What is the lowerbound on the power of the systems that we consider? If **P** uniformity is used, then we get a trivial **P** lowerbound [6]. However for uniformity that is tighter than **P** (e.g. $\mathbf{AC}^0$ or **L**), then we conjecture that a **P** lowerbound can be found by improving a result in [6].

2 Membrane Systems

In this section we define membrane systems and some complexity classes. These definitions are based on those from Pérez-Jiménez et al. [2], Păun [9,10], Sosík and Rodríguez-Patón [12], and Murphy and Woods [4].

2.1 Active membrane systems

Active membrane systems are a class of membrane systems with membrane division rules. In this paper, division rules can act only on elementary membranes, which are membranes that do not contain other membranes (and are represented as leaves in the membrane structure tree).

Definition 2. *An active membrane system without charges is a tuple* $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$ *where,*

1. $m \geq 1$ *is the initial number of membranes;*
2. O *is the alphabet of objects;*
3. H *is the finite set of labels for the membranes;*
4. μ *is a tree that represents the membrane structure, consisting of* m *nodes, labelled with elements of* H*. The root node (representing the parent of all membranes) is called the "skin" and has label* $1 \in H$*;*
5. $w_1, \ldots, w_m$ *are strings over* O*, describing the multisets of objects placed in the* m *regions of* μ*.*
6. R *is a finite set of developmental rules, of the following forms:*
   (a) $[\,a \rightarrow u\,]_h$*, for* $h \in H,\ a \in O,\ u \in O^*$ *(object evolution)*
   (b) $a[\,]_h \rightarrow [\,b\,]_h$*, for* $h \in H,\ a, b \in O$ *(communication in)*
   (c) $[\,a\,]_h \rightarrow [\,]_h\ b$*, for* $h \in H,\ a, b \in O$ *(communication out)*
   (d) $[\,a\,]_h \rightarrow b$*, for* $h \in H,\ a, b \in O$ *(membrane dissolution)*
   (e) $[\,a\,]_h \rightarrow [\,b\,]_h\ [\,c\,]_h$*, for* $h \in H,\ a, b, c \in O$ *(elementary membrane division)*

The semantics of these rules are described elsewhere [10], however we note that they are applied according to the following principles:

- All the rules are applied in a maximally parallel manner. That is, in one step, one object of a membrane is used by at most one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must evolve.
- The rules associated with membranes labelled with h are used for membranes with that label. At one step, a membrane can be the subject of only one rule of types (b)–(e), although in this paper we use only rules of type (d) and (e).

2.2 Recogniser membrane systems

We recall [1] that a computation of a membrane system is a sequence of configurations such that each configuration (except the initial one) is obtained from the previous one by a transition (one-step maximally parallel application of the rules). Systems are nondeterministic, therefore on a given input there are multiple possible computations.

Definition 3 ([1]). *A* recogniser membrane system *is a membrane system where each computation outputs either object* `yes` *or* `no` *(to the membrane with label 1). When this output occurs, no other rules are applicable.*

2.3 Complexity classes

Consider a decision problem X, i.e. a set of instances $X = \{x_1, x_2, \ldots\}$ over some finite alphabet such that to each x_i there is a unique answer "yes" or "no". We say that a *family* of membrane systems solves a decision problem if each instance of the problem is solved by some family member. We let $|x|$ denote the length of x. In this paper, the functions classes E, F, H are each contained in the class of functions that are polynomial time computable on deterministic Turing machines.

Definition 4. *Let $\mathcal{D}$ be a class of membrane systems, let E and F be classes of functions, and let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a total function. The class of* problems solved by (E, F)-uniform families of membrane systems *of type $\mathcal{D}$ in time t, denoted (E, F)–$\mathbf{MC}_{\mathcal{D}}(t)$, contains all problems X such that:*

- *There exists an f*-uniform *family of membrane systems, $\mathbf{\Pi}_X = \{\Pi_X(1), \Pi_X(2), \ldots\}$ of type $\mathcal{D}$: that is, there exists a function $f \in F$, $f : \{1\}^* \rightarrow \mathbf{\Pi}_X$ such that $f(1^n) = \Pi_X(n)$.*
- *There exists an input encoding function $e \in E$, $e : X \rightarrow \Sigma^*$ where $e(x)$ represents the input multiset (as a word), which is placed in the input membrane of $\Pi_X(n)$.*
- *The pair $\mathbf{\Pi}_X, e$ is* sound *and* complete *with respect to X: $\Pi_X(n)$ given the encoded input $e(x)$, where $|x| = n$, accepts iff the answer to x is "yes".*
- *$\mathbf{\Pi}_X$ is t-efficient: $\Pi_X(n)$ always halts in at most $t(n)$ steps.*

Definition 4 describes (e, f)-uniform family (i.e. with input) and we generalise this to define (h)*-semi-uniform family of membrane systems* $\mathbf{\Pi}_X = \{\Pi_X(x_1), \Pi_X(x_2), \ldots\}$ where there exists a function $h \in H$, $h : X \rightarrow \mathbf{\Pi}_X$ such that $h(x) = \Pi_X(x)$. Here a single function (rather than two) is used to construct the semi-uniform membrane family, and so the problem instance is encoded using objects, membranes, and rules. In this case, for each instance of $x \in X$ we have a (possibly unique) membrane system. The resulting class of problems is denoted by (H)–$\mathbf{MC}^*_{\mathcal{D}}(t)$. We define

$$(E, F)\text{–}\mathbf{PMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} (E, F)\text{–}\mathbf{MC}_{\mathcal{D}}(n^k),$$

$$(H)\text{–}\mathbf{PMC}^*_{\mathcal{D}} = \bigcup_{k\in\mathbb{N}} (H)\text{–}\mathbf{MC}^*_{\mathcal{D}}(n^k).$$

We let $\mathcal{AM}^0_{+d,-ne,-e,-c}$ denote the class of active membrane systems without charges and using only dissolution and elementary division rules. Theorem 1 is shown for such (polynomial time) systems. For consistency with the membrane systems literature (which often uses **P** uniformity and semi-uniformity), we simply write $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^*_{\mathcal{D}}$ when E, F, H are classes of functions that are polynomial time computable on deterministic Turing machines. The results in this paper hold for **P**, or tighter, uniformity and semi-uniformity.

Remark 5. A recogniser membrane system is *confluent* if it is both sound and complete. That is a Π_X is *confluent* if all computations of Π_X, with the same encoded input, give the same result.

Therefore the following interpretation holds: in a confluent membrane system, given a fixed initial configuration, the system non-deterministically chooses one from a number of valid computations, but all computations must lead to the same result, either all accepting or all rejecting.

2.4 Notation

For a membrane system Π we define $\Delta_i = \{\delta \mid [\delta]_i \to o$ is a rule in $\Pi, o \in O, i \in H\}$. In other words, Δ_i is the set of objects that are on the left hand side of a rule that dissolves a membrane with label i. Also, $\Delta_1 = \{\texttt{yes}, \texttt{no}\}$, and $\Delta = \cup_{i\in H} \Delta_i$. The parent of a membrane with label i is denoted $p(i)$. We often explicitly write the contents of a multiset as a vector C of multiplicities, as shown in Section 4.4.

3 Object division graphs

We introduce object division graphs which represent all divisions that result from a given object and membrane label. We use the notion of dependency graph [1], a directed acyclic graph previously used to represent non-dissolution rules [i.e. types (a), (b), (c), and (e)]. Given a (properly encoded) set of rules for a membrane system Π, the dependency graph G_Π is created in logspace [6]. An *object division graph* is a directed acyclic graph $G_{\text{div}(o,h)}$ that is the dependency graph for division rules only (ignoring all other rules), and that contains only those nodes and edges that are on paths leading from (o, h). We can also define an object division graph as follows. For each division rule $[\, o_i \,]_h \to [\, o_j \,]_h [\, o_k \,]_h$ we define the three nodes $\{o_{i,h}, o_{j,h}, o_{k,h}\}$ and the two edges $(o_{i,h}, o_{j,h})$ and $(o_{i,h}, o_{k,h})$. Then, the object division graph $G_{\text{div}(o_\ell,h)}$ is the subgraph that is reachable from the node $o_{\ell,h}$.

For example, Figure 1 shows four object division graphs (each is contained in a 'cloud'). The three rules that gave rise to the leftmost object division graph (rooted at a) are as follows, $[\, a \,] \to [\, b \,][\, c \,], [\, b \,] \to [\, \delta_{p(h_{\max})} \,][\, e \,], [\, c \,] \to [\, e \,][\, \delta_3 \,]$. (All membranes are labeled $h_{\max}$ and this label is omitted for brevity.)

For each element of $O \times H$ the simulation algorithm creates such a graph. By Lemma 8, we are simulating a membrane system Π where division is deterministic in the sense that for a given (object, label) pair, there is at most one applicable division rule. From this, and Lemma 7 it can be seen that each non-leaf vertex in $G_{\mathrm{div}(o,h)}$ has out-degree 2. Also, from the fact that division rules do not change membrane labels, a single object division graph encodes a set of rules, where each rule in the set has the same label.

4 Simulation

Here we prove our main result, Theorem 1.

Theorem 1 $\mathbf{PMC}^*_{\mathcal{D}} \subseteq \mathbf{P}$*, where* $\mathcal{D}$ *is the class of systems in* $\mathcal{AM}^0_{+d,-ne,-e,-c}$ *having an initial membrane structure that is a single path.*

We give a polynomial time algorithm that takes as input a membrane system Π from the class $\mathcal{AM}^0_{+d,-ne,-e,-c}$, which has a membrane structure that is a single path, as well as Π's input. The simulation algorithm begins with the (relatively easy) task of simplifying Π using Lemmas 6, 7 and 8. Then, the step-by-step simulation uses the algorithms given from Section 4.2 onwards.

4.1 Normal Forms and simplifications

The algorithm begins the simulation by simplifying the membrane system as follows.

Lemma 6 (Normal form: unique labels at initial step). *Given a membrane system* Π *with multiple membranes of the same label at the initial configuration, this can be converted (in time quadratic in the size of* Π*) to a membrane system* Π' *where each membrane has a unique label that accepts* w *iff* Π *does.*

Proof. We can iterate the following algorithm until each membrane has a unique label. For each membrane of label h, give the ith membrane a new label h_i, and create a new set of rules that is identical to the set of rules that make use of label h, except h is replaced by h_i in each rule. The algorithm runs in polynomial time, since both the set of rules R and labels H are of polynomial size. $\square$

Thus we can assume that each membrane has a unique label from $\{1, \ldots, |H|\}$ in the initial configuration of the membrane system, labelled sequentially from the root 1, to leaf $|H|$.

Lemma 7 (Normal form: removing symmetric division rules). *Given a membrane system* Π*, it can be converted (in polynomial time in the size of* Π*) to a membrane system* Π' *where all division rules are asymmetric, that is each division rule is of the form* $[\, o_1 \,]_i \rightarrow [\, o_2 \,]_i [\, o_3 \,]_i$*, and* $o_2 \neq o_3$*.*

Proof. The rules in Π' are the same as those Π, with the following exception. Each symmetric division rule $[\, o_1 \,]_i \rightarrow [\, o_2 \,]_i [\, o_2 \,]_i$ in Π is replaced by an asymmetric division rule $[\, o_1 \,]_i \rightarrow [\, o_2 \,]_i [\, o_2' \,]_i$ in Π'. We add the following: for each rule with o_2 on the left hand side, we create an extra rule which is the same except it has o_2' on the left hand side. $\square$

Lemma 8 (Normal form: removing nondeterminism on left-hand sides of the rules). *Suppose there is more than one rule for a given object $o \in O$ and label $h \in H$ in membrane system Π. Let Π' be identical to Π, except that we have exactly one of these rules (arbitrarily chosen) for each pair (o, h).*

That is, given any $o \in O$, for each $h \in H$ there is at most rule: either one division rule of the form $[\, o \,]_h \rightarrow [\, b \,]_h [\, c \,]_h$ with o on the left hand side, or one dissolution rule of the form $[\, o \,]_h \rightarrow u$ with o on the left hand side.

The resulting membrane system Π' accepts input word w if and only if Π accepts w.

Proof. Assuming that Π is a recogniser, we know (by definition) that given an input w either all computations accept it (yielding "yes" as output), or else none of the computations accept it (the output is always "no"). The proof follows from the observation that all computations of Π' are also computations of Π, since every time that it is possible to apply several rules of types (d) or (e) on a membrane labelled by h we must choose exactly one such rule (see Definition 2). This means that when Π is running a computation and is in a situation where there is an object o in a membrane labelled by h, it nondeterministically chooses the single rule that Π' has for the pair (o, h). $\square$

The input membrane system has now been simplified (within polynomial time) and we begin the simulation assuming these normal forms.

4.2 Simulating, and avoiding, division

The simulation algorithm classifies the system into one of Cases 0, 1, 2, or 3, depending on the contents of the membrane with greatest label $h_{\max}$ ($h_{\max} = |H|$ in the first timestep).

If the system fits Cases 0 or 1 then we simulate a valid computation where all membranes with label $h_{\max}$ (eventually) dissolve, and moreover in a way that can be simulated. This reduces the depth of the membrane structure by 1. Then, the algorithm restarts, and once again classifies the system into one of the 4 cases. If we only ever encounter Cases 0 and 1 then a straightforward inductive argument can be used to give prove the correctness of our simulation.

In either of Cases 2 or 3, we select a computation where there are membrane(s) of label $h_{\max}$ that *never* dissolve. In these cases, the sub-system with labels $\{1, \ldots, p(h_{\max})\}$ is non-elementary for all time $\geq t$, and the algorithm in Section 4.4 is used to simulate this sub-system. For Case 2 simulation of the elementary membranes is relatively straightforward. However, for Case 3 a more sophisticated simulation algorithm for elementary membranes is given in Section 5.

These four cases give full coverage in the sense that any valid configuration is represented by one of the cases, and this can be seen by a careful examination of Lemmas 9 to 12.

4.3 Case 0

Lemma 9 (Case 0). *At timestep t, if the single membrane of label $h_{\max}$ contains an object δ where $\delta \in \Delta_{h_{\max}}$, then there is a polynomial time simulation of the system from time t to $t+1$.*

Proof. We simulate the computation where dissolution is chosen (has priority) over division. If there are multiple such δ, we choose the lexicographically first one. The algorithm given below in Section 4.4 is then used for the simulation of the dissolution rule on $h_{\max}$, and likewise for any applicable dissolution rule on membranes of label $< h_{\max}$ for one timestep.

Then the simulation algorithm returns to the beginning (of Section 4.2), to find which of Cases 0, 1, 2, or 3 is applicable for timestep $t+1$. $\square$

4.4 Simulation algorithm for dissolution only

In this section we give a polynomial time simulation for the case where there are only dissolution rules. This could also be shown via Theorem 3 from Zandron et al. [13], but we instead give an explicit simulation that is useful in our constructions.

Suppose there are h membranes with distinct labels $\{1, \ldots, h\}$ at some time t (initially, $t = 0$). At time t, the simulation algorithm encodes the contents of each membrane as a vector. Specifically, for each membrane of label $i \in \{1, \ldots, h\}$, the multiset contents are encoded explicitly as

$$C_{i,0} = \left\langle |i|_{o_1}, |i|_{o_2}, \ldots, |i|_{o_{|O|}} \right\rangle$$

where $|i|_{o_j}$ is the multiplicity of $o_j \in O$ in the membrane with label i. Also, for all $t' > t$, $C_{i,t'} = \langle 0, 0, \ldots, 0 \rangle$. For all time t, the system has a polynomial number of labels so the vectors $C_{i,t}$ can be written out by a polynomial time Turing machine.

The system uses only dissolution rules, thus it is straightforward to find the contents of $C_{i,t+1}$, for each i, in polynomial time, as follows. Initially, let $C_{i,t+1} = \langle 0, 0, \ldots, 0 \rangle$. Then, for each i, from h down to 1, check the vector $C_{i,t}$ to see if $|i|_\delta > 0$, where $\delta \in \Delta_i$ (i.e. check if $C_{i,t}$ contains a non-zero value for any $\delta \in \Delta_i$):

- if so, add $C_{i,t} - \delta + x$ to $C_{p(i),t+1}$ (where $[\delta]_i \rightarrow x$ is the simulated dissolution rule, and if there are multiple such δ, we choose the lexicographically first),[3]
- if not add $C_{i,t}$ to $C_{i,t+1}$.

[3] Here the notation $C_{i,t} - \delta + x$ means remove one copy of δ from C and add one copy of x to $C_{i,t}$ (i.e. we let δ, x be vectors that encode the symbols δ, x).

The algorithm should take care of the important detail where, in one step, the dissolution occurs of multiple membranes with consecutive labels. If the (skin membrane) vector $C_{1,t'}$, for any t', encodes the object `yes` (respectively, `no`) then the simulation is finished and we accept (respectively, reject).

The computation that we are simulating for these membranes should be clear from the algorithm, so we omit a proof of correctness (which would proceed by a simple inductive argument on membrane contents).

4.5 Cases 1 and 2

Lemma 10 (Case 1). *At timestep t, if the single membrane of label $h_{\max}$ contains an object o where the object division graph $G_{\mathrm{div}(o,h_{\max})}$ has the property that every sink node is in $\Delta_{h_{\max}}$, then there is a polynomial time simulation of the computation from time t to time $t+d+1$. Here d is the length of the longest path starting at the root o in $G_{\mathrm{div}(o,h_{\max})}$. Moreover, at time $t+d+1$, all of the label $h_{\max}$ membranes have been dissolved.*

Proof. If there are multiple such o, we choose the lexicographically first one. To simulate the computation of membranes of label $h_{\max}$, we choose the (valid) computation that is defined by the graph $G_{\mathrm{div}(o,h)}$. That is, starting at object o, we simulate the divisions defined by out-branchings in $G_{\mathrm{div}(o,h)}$, followed by dissolutions that are triggered by sink nodes in $G_{\mathrm{div}(o,h)}$.

The object division graph $G_{\mathrm{div}(o,h)}$ may have an exponential number of paths, so we do not *explicitly* simulate the membrane division and subsequent dissolution. Instead, we calculate the number of objects that are expelled (dissolved out) from $h_{\max}$ membranes at each timestep, as follows.

Let $d' \in \{1, 2, \ldots, d\}$. At time $t + d'$ we calculate the following. For the jth sink node in $G_{\mathrm{div}(o,h)}$, we calculate k_j which is the number of paths of length d' that begin at o and end in sink j (by multiplying the adjacency matrix of $G_{\mathrm{div}(o,h)}$ by itself k times). Let $\delta_j \in \Delta_{h_{\max}}$ be the label of the jth sink, and let $[\delta_j]_{h_{\max}} \to x_j$. At time $t + d'$ the vector

$$\sum_j (k_j \cdot (C_{h_{\max},t} + x_j - o))$$

is added to $C_{p(h_{\max}),t+d'+1}$. The algorithm given in Section 4.4 is then used for the remaining simulation of labels $< h_{\max}$ at timestep $t + d'$.

After d timesteps the simulation algorithm returns to the beginning (of Section 4.2), to find which of Cases 0, 1, 2, or 3 is applicable for time $t + d + 1$. □

Lemma 11 (Case 2). *At timestep t, for all objects o in the single membrane of label $h_{\max}$, if $o \notin \Delta_{h_{\max}}$ and if the division tree of (o, h) has no sinks in $\Delta_{h_{\max}}$, then there is a polynomial time simulation of the computation from time t until the computation finishes. Moreover, a simulation of this computation can ignore membranes of label $h_{\max}$, and assume all ancestors of $h_{\max}$ never divide.*

Proof. There is a computation where no membrane of label $h_{\max}$ ever dissolves. The simulation for this case ignores all membranes of label $h_{\max}$ for all time $> t$. The algorithm in Section 4.4 simulates the dissolution-only sub-system with labels $< h_{\max}$. $\square$

5 Simulation algorithm for Case 3

Lemma 12 (Case 3). *At timestep t, we look at all objects $o \in O$ in the single membrane of label $h_{\max}$. If both of the following hold:*

- *for all such o, there are ≥ 1 sink nodes of $G_{\mathrm{div}(o,h_{\max})}$ not in $\Delta_{h_{\max}}$, and*
- *there is at least one such o where ≥ 1 sink nodes of $G_{\mathrm{div}(o,h_{\max})}$ are in $\Delta_{h_{\max}}$,*

then there is a polynomial time simulation of the computation from time t until the computation finishes.

The proof is given in the remainder of Section 5.

5.1 Overview of simulation technique

We define a special graph called the division graph, and use that to choose a specific computation to simulate. In fact we simulate only a very small number of membranes explicitly, and ignore an exponentially large number of other membranes. Remarkably, we explicitly simulate at most $2h_{\max} - 2$ membranes, where $h_{\max}$ is the depth of the membrane structure. We prove that this small number of membranes is sufficient to correctly predict the output of the computation. Then, by confluence, the correct prediction of one computation implies a correct prediction of the answer to all computations.

5.2 Division graph

The simulation algorithm creates a (directed acyclic) *division graph*, $\mathcal{G}_{h_{\max}}$ using the following algorithm:

– Write out the object division graph $G_{\mathrm{div}(o,h_{\max})}$, for each object o in m (the graphs are written in lexicographical order by object o, multiplicities included).
– Add a directed edge from each sink vertex of graph i to the root of graph $i + 1$. These edges are called *dummy edges* because they do not correspond to actual rules.

An example division graph is illustrated in Figure 1.

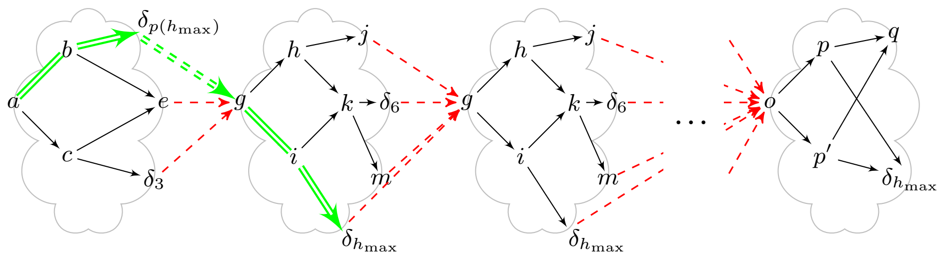

Figure 1. Example division graph $\mathcal{G}_{h_{\max}}$. The contents (in this case $a, g, g, \dots, o$) of the single membrane with label $h_{\max}$, and the $h_{\max}$ division rules, are used to define $\mathcal{G}_{h_{\max}}$. Dummy edges ($\dashrightarrow$) are dashed and go from sinks to roots. The shortest path to expel the object $\delta_{p(h_{max})}$ is highlighted ($\Rightarrow$).

5.3 Some observations about the division graph

We claim that the division graph is a good model of membrane division, in the sense that it accurately models the *division history* of any membrane (given a membrane of label $h_{\max}$, its division history is defined to be the sequence of division rules, with a left/right choice for each rule, that gave rise to that membrane from the time when there was exactly one membrane of label $h_{\max}$). In the following lemmas, the function length$(\cdot)$ does not count dummy edges.

Lemma 13. *Each path ρ in $\mathcal{G}_{h_{\max}}$ models the division history of a membrane with label $h_{\max}$, for exactly $\tau = \text{length}(\rho)$ consecutive division steps.*

Lemma 14. *The division history of any membrane of label $h_{\max}$, after τ consecutive divisions, is modelled by some path ρ in $\mathcal{G}_{h_{\max}}$, and* $\text{length}(\rho) = \tau$.

5.4 Chosen computation

The simulation proceeds by simulating some important aspects of a particular computation. This *chosen computation* is defined as follows.

Remark 15. Our chosen computation of Π satisfies all of the following criteria:

1. Division of $h_{\max}$ membranes: All $h_{\max}$ divisions occur in the order defined by the division graph $\mathcal{G}_{h_{\max}}$. The divisions may terminate early, but if a root object[4] starts dividing, then these divisions must continue until the sinks of $G_{\text{div}(o,h)}$ (thus if there is a non-root, non-leaf object o present in a $h_{\max}$ membrane we must immediately divide using o, and we cannot divide by another object, nor dissolve).

[4] By root object o, we mean an object that is a root vertex in one of the object division graphs $G_{\text{div}(o,h)}$ that $\mathcal{G}_{h_{\max}}$ is composed of. Analogously, a leaf bject is an object that is a leaf in some $G_{\text{div}(o,h)}$.

2. Dissolution of $< h_{\max}$ membranes: If a non-elementary membrane of label i is to dissolve, then this happens at the earliest possible time $t_{\min}$ (using some object $\delta \in \mathit{\Delta}_i$) that is consistant with 1 above. (If there are multiple such δ, then then any δ is chosen.)
3. Dissolution of $h_{\max}$ membranes: membranes of label $h_{\max}$ choose division of root objects over dissolution, except where this would contradict point 2.

Remark 16. If we dissolve a label $h_{\max}$ membrane then all of its object contents are roots or sinks, in object division graphs (in $\mathcal{G}_{h_{\max}}$).

5.5 Overview of Case 3 simulation algorithm

We do not simulate the division and dissolution of all label $h_{\max}$ membranes: However, a subset, of $\leq h_{\max} - 1$ membranes, are explicitly simulated. The following is an overview of our simulation algorithm for Case 3, the details are given in the remainder of the section.

At timestep t:

– if label $p(h_{\max})$ contains an element of $\mathit{\Delta}_{p(h_{\max})}$ then simulate the dissolution of $p(h_{\max})$,
– else if any label $h_{\max}$ membrane expels (dissolves) out a $\mathit{\Delta}_{p(h_{\max})}$ object in $< t$ timesteps, then explicitly simulate (only) the relevant label $h_{\max}$ membrane, and simulate the subsequent dissolution of $p(h_{\max})$,
– else $p(h_{\max})$ is not dissolved,
– membranes of label $< p(h_{\max})$ are simulated explicitly (as in Section 4.4).

5.6 Expel: simulating division and dissolution using shortest paths

Before giving the main simulation algorithm, we first give `expel`(S, τ), see Algorithm 1, which uses the division graph $\mathcal{G}_{h_{\max}}$. Given a set of objects S, and a time τ, the function `expel`(S, τ) finds whether or not the elementary membranes (of label $h_{\max}$) expel (release on dissolution) any element of S within τ timesteps. If so, `expel`(S, τ) returns the vector C, which represents the contents of the entire membrane m that expels some such $s \in S$. In particular, if there are multiple such membranes that release an element of S in time $\leq \tau$, then `expel`(S, τ) chooses the membrane which dissolves at the earliest timestep.

The `expel`(S, τ) algorithm computes the shortest path in the division graph $\mathcal{G}_{h_{\max}}$ that expels any $s \in S$ [shortest path is **NL**-complete [7], and at most $4|S|$ shortest paths need to found in each run of `expel`(S, τ)]. If no element of S is expelled by membranes of label $h_{\max}$ in $\leq \tau$ timesteps, then `expel`(S, τ) returns FALSE.

Algorithm 1 Find whether a membrane with label $h_{\max}$ expels any object from the set S in time $\leq \tau$.

```
divGraph.expel($S$: set of objects, $\tau$: time $\in \mathbb{N}$)
   // $s$ is a sink or root object, and $\delta_{h_{\max}} \in \Delta_{h_{\max}}$ is a sink object (in object division
   // graphs). Path lengths do not include dummy edges. We ignore ‘used’ paths. $O$ is the
   // alphabet of objects (i.e. nodes in $\mathcal{G}_{h_{\max}}$). All paths begin at the root of the first
   // object division graph in $\mathcal{G}_{h_{\max}}$.
   FOR all $s \in S$, find the shortest path $\rho$ (in $\mathcal{G}_{h_{\max}}$) of one of the following forms:
      $O^* s O^* \delta_{h_{\max}}$
      $O^* \delta_{h_{\max}} O^* s$
      $O^* \delta_{h_{\max}}$ and $\exists$ a rule: $[\delta_{h_{\max}}]_{h_{\max}} \rightarrow s$
      $O^* \delta_{h_{\max}}$ and $s$ is in $h_{\max}$ at the timestep where Case 3 began
   END FOR
   IF (length($\rho$) $\leq \tau$)     // Do not count dummy edges in path lengths
      Explicitly simulate the membrane induced by the path $\rho$ to give expelled objects $C$
      Mark the path $\rho$ as ‘used’ in $\mathcal{G}_{h_{\max}}$
      RETURN($C$)
   ELSEIF ( ($\nexists$ such $\rho$) OR (length($\rho$) $> \tau$) )
      RETURN FALSE
   END IF/ELSEIF
END expel
```

5.7 Case 3 simulation algorithm

Algorithm 2 simulates the computation described in Remark 15. Initially there is one membrane of label $h_{\max}$, however over time (at most) exponentially many such membranes are created. So Algorithm 2 does not explicitly simulate all label $h_{\max}$ membranes. At each timestep t, Algorithm 2 checks if it should dissolve the membrane of label $p(h_{\max})$, by (i) checking $C_{p(h_{\max}),t}$ for a $\Delta_{p(h_{\max})}$ object, and if not then (ii) checking if any $h_{\max}$ membrane expels a $\Delta_{p(h_{\max})}$ object in any timestep $< t$, using `expel`$(\Delta_{p(h_{\max})}, t-1)$.

When Algorithm 2 begins execution, we have just entered Case 3 at time t and there is at most one membrane of each label. Also, for all i, the simulation algorithm has already initialised the vectors $C_{i,t}$ to their correct contents, and for all $t' > t$, $C_{i,t'} = \langle 0, 0, \ldots, 0 \rangle$.

5.8 Case 3 proof of correctness

Lemma 17. *Algorithm 2 simulates the dissolution of membrane with label $p(h_{\max})$ at timestep t, using dissolution rule r, if and only if the chosen computation (Section 5.4) does so at the same timestep t.*

Proof (If). In the chosen computation defined in Section 5.4, if $p(h_{\max})$ is to dissolve at time t, then we show that Algorithm 2 simulates the dissolution of $p(h_{\max})$, using a valid rule. There are two cases, (i) and (ii).

(i) Some membrane of label $h_{\max}$ expels a $\Delta_{p(h_{\max})}$ object at time $< t$. Algorithm 2 uses Algorithm 1, via the call `expel`$(\Delta_{p(h_{\max})}, t-1)$, to find whether membranes of label $h_{\max}$ expel $\Delta_{h_{\max}}$ objects at any time $< t$. Let $C =$ `expel`$(\Delta_{p(h_{\max})}, t-1)$. If $C \neq$ `FALSE`, then C represents the contents of a membrane that expels a $\Delta_{p(h_{\max})}$ object.

Algorithm 2 Simulation of membranes with label $h_{\max}$ and $p(h_{\max})$. At time t we are in Case 3, and there is one membrane of label $p(h_{\max})$.

```
$C_{p(h_{\max}),t}$ := contents of the single membrane with label $p(h_{\max})$ at time $t$
WHILE ($C_{1,t}$ encodes neither yes nor no)
   IF (there is any $\delta \in \Delta_{p(h_{\max})}$ encoded in $C_{p(h_{\max}),t}$)
      Simulate the rule $[\delta]_{p(h_{\max})} \to x$ by adding ($C_{p(h_{\max}),t} + x - \delta$) to $C_{p(p(h_{\max})),t+1}$
      $\forall t' > t$ delete vectors $C_{p(h_{\max}),t'}$
      $p(h_{\max})$ := $p(p(h_{\max}))$
   ELSE
      // Check if any $h_{\max}$ membrane expels any $\delta \in \Delta_{p(h_{\max})}$ object in any timestep $< t$:
      $C$ := $\mathcal{G}_{h_{\max}}$.expel($\Delta_{p(h_{\max})}$, $t-1$)
      IF ($C \neq$ FALSE)
         Add $C$ to $C_{p(h_{\max}),t}$
         Simulate the rule $[\delta]_{p(h_{\max})} \to x$ by adding ($C_{p(h_{\max}),t} + x - \delta$) to $C_{p(p(h_{\max})),t+1}$
         $\forall t' > t$ delete vectors $C_{p(h_{\max}),t'}$
         $p(h_{\max})$ := $p(p(h_{\max}))$
      ELSEIF ($C =$ FALSE)
         // Do nothing, i.e. $p(h_{\max})$ is not dissolved
      END IF/ELSEIF
   END IF/ELSE
   Update all $C_{i,t+1}$, where $i < p(h_{\max})$, using algorithm in Section 4.4
   $t$ := $t + 1$
END WHILE
```

(ii) There exists a $\Delta_{h_{\max}}$ object in $p(h_{\max})$ at time t. If this is the case then that object was either there at the timestep where $p(h_{\max})$ was non-elementary (in which case it was encoded in $C_{p(h_{\max}),t}$ at an earlier timestep), or else was expelled by some membrane of label $h_{\max}$ (already shown in (i)).

(Only if) If $p(h_{\max})$ does not to dissolve at time t, then we are in one of two cases, (iii) or (iv).

(iii) There does not exist a $\Delta_{p(h_{\max})}$ object in $p(h_{\max})$ at time t. In Algorithm 2, the vector $C_{p(h_{\max}),t}$ encodes a subset of the contents of the membrane $p(h_{\max})$, therefore if there are no $\Delta_{p(h_{\max})}$ objects in the membrane $p(h_{\max})$ at time t, then there are no $\Delta_{p(h_{\max})}$ objects encoded in $C_{p(h_{\max}),t}$ at time t.

(iv) No membrane of label $h_{\max}$ expels a $\Delta_{p(h_{\max})}$ object at time $< t$. Algorithm 2 uses Algorithm 1, via the call `expel`$(\Delta_{p(h_{\max})}, t-1)$, to find whether membranes of label $h_{\max}$ expel $\Delta_{h_{\max}}$ objects at any time $< t$. If `expel`$(\Delta_{p(h_{\max})}, t-1) =$ `FALSE` then this does not occur on the chosen computation path, and Algorithm 2 does not simulate the dissolution of $p(h_{\max})$.

□

Lemma 18. *Algorithm 2 runs in polynomial time and outputs* `yes` *iff Π does, and outputs* `no` *otherwise.*

Proof. By Lemma 17 we dissolve the $p(h_{\max})$ membranes correctly (i.e. consistent with the chosen computation in Section 5.4). The membranes of label i, where $1 \leq i < p(h_{\max})$, are simulated using the algorithm given in Section 4.4. Finally, we note that $\Delta_1 = \{\texttt{yes}, \texttt{no}\}$, and so Algorithm 2 treats the search for $\{\texttt{yes}, \texttt{no}\}$ in the same way that the search for any Δ_i object is treated.

Algorithm 2 iterates a number of times that is less than or equal to the number of time steps of the chosen computation that is being simulated. The most significant operations in Algorithms 1 and 2 involve vector addition/subtraction, checking vector coordinates for 0, and finding shortest paths, all of which can be computed in polynomial time. □

Acknowledgement

This work is supported by a Project of Excellence TIC-581 from the Junta de Andalucía, project TIN 2006 13425 of the Ministerio de Educación y Ciencia of Spain, and the Irish Research Council for Science, Engineering and Technology. We thank the anonymous reviewers for helpful comments on the presentation.

References

1. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez, and F. J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, 83(7):593–611, 2006.
2. M. J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, 2003.
3. G. Mauri, M. J. Pérez-Jiménez, and C. Zandron. On a Păun's Conjecture in Membrane Systems. In *Bio-inspired Modeling of Cognitive Tasks*, pages 180–192, 2007.
4. N. Murphy and D. Woods. The computational complexity of uniformity and semi-uniformity in membrane systems. In preparation.
5. N. Murphy and D. Woods. Active membrane systems without charges and using only symmetric elementary division characterise P. In *Proceedings of the 8th Workshop on Membrane Computing, Thessaloniki, Greece*, volume 4860 of *LNCS*, pages 367–384. Springer, June 2007.
6. N. Murphy and D. Woods. A characterisation of NL using membrane systems without charges and dissolution. In *Seventh International Conference on Unconventional Computation*, volume 5204 of *LNCS*, pages 164–176, Vienna, 2008. Springer.
7. C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1995.
8. G. Păun. Further twenty six open problems in membrane computing. In *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain)*, pages 249–262, Jan. 2005.
9. G. Păun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
10. G. Păun. *Membrane Computing.* Springer-Verlag, Berlin, 2002.
11. P. Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, Aug. 2003.
12. P. Sosík and A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73(1):137–152, 2007.
13. C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In I. Antoniou, C. Calude, and M. Dinneen, editors, *UMC '00: Proceedings of the Second International Conference on Unconventional models of Computation*, pages 289–301, London, UK, Feb. 2000. Springer-Verlag.