The Computational Complexity of Uniformity and Semi-uniformity in Membrane Systems

Niall Murphy[1] and Damien Woods[2]

[1] Department of Computer Science,
National University of Ireland, Maynooth, Ireland
http://www.cs.nuim.ie/∼nmurphy/
nmurphy@cs.nuim.ie
[2] Department of Computer Science and Artificial Intelligence,
University of Seville, Spain
http://www.cs.us.es/∼dwoods/
dwoods@us.es

Summary. We investigate computing models that are presented as families of finite computing devices with a uniformity condition on the entire family. Examples include circuits, membrane systems, DNA computers, cellular automata, tile assembly systems, and so on. However, in this list there are actually two distinct kinds of uniformity conditions.

The first is the most common and well-understood, where each input length is mapped to a single computing device that computes on the finite set of inputs of that length. The second, called semi-uniformity, is where each input is mapped to a computing device for that input. The former notion is well-known and used in circuit complexity, while the latter notion is frequently found in literature on nature-inspired computing models, from the past 20 years or so.

Are these two notions distinct or not? For many models it has been found that these notion are in fact the same, in the sense that the choice of uniformity or semi-uniformity leads to characterisations of the same complexity classes. Here, we buck this trend and show that these notions are actually distinct: we give classes of uniform membrane systems that are strictly weaker than their semi-uniform counterparts. This solves a known open problem in the theory of membrane systems.

1 Introduction

In his famous 1984 paper on DNA computing [1], Adleman mapped a specific instance of the travelling salesman problem (TSP) to a set of DNA strands, and then used well-known biomolecular techniques to solve the problem. To assert generality for his algorithm, one would define a (simple) mapping from arbitrary TSP instances to sets of DNA strings. Then, in order to claim that this mapping is not doing the essential computation, it would have to be easily computable

(e.g. logspace computable). Circuit uniformity provides a well-established framework where we map each input length $n \in \mathbb{N}$ to a circuit $c_n \in C$, with a suitably simple mapping. However, Adleman did something different, he mapped a specific *instance* of the problem to a computing device. We call this notion *semi-uniformity*, and in fact a large number of computation models use semi-uniformity. This raises the immediate question of whether the notions of uniformity and semi-uniformity are equivalent.

It has been shown in a number of models that whether one chooses to use uniformity or semi-uniformity does not affect the power of the model. However, in this paper we show that these notions are not equivalent. We prove that choosing one notion over another gives characterisations of completely different complexity classes, including known distinct classes.

We prove this result for a computational model called membrane systems (also known as P-systems) [15]. Membrane computing is a branch of natural computing which defines computation models that are inspired by the structure and function of living cells. The membrane computing model is sufficiently formal that this question can be clearly stated, e.g. it is stated as Open Problem C in [16].

Why is this result surprising? Every class of problems solved by a uniform family of devices is contained in the analogous semi-uniform class, since one is a restriction of the other. However, in all membrane system models studied to date, the classes of problems solved by semi-uniform and uniform families turned out to be equal [4, 10, 19]. Specifically, if we want to solve some problem, by specifying a family of membrane systems (or some other model), it is often much easier to first use the more general notion of semi-uniformity, and then subsequently try to find a uniform solution. In almost all cases where a semi-uniform family was given for some problem [3, 11, 13, 19], at a later point a uniform version of the same result was published [2, 4, 13]. Here we prove that this improvement is not always possible.

Since our main result proves something general about families of finite devices we would hope that, in the future, it can be applied to other computational models, besides membrane systems. Why? Firstly, our results are proved by converting the membrane system into a directed acyclic graph. Input acceptance is then rephrased as a graph reachability problem and this gives a very general tool that can be applied to other computational models (where we can find analogous graph representations). Secondly, the result concerns a general concept (uniformity/semi-uniformity) that is independent of particular formalisms. Besides membrane systems and circuits, some other models that use notions of uniformity and semi-uniformity include families of neural networks, molecular and DNA computers, tile assembly systems and cellular automata [6, 8, 12, 17, 18]. Our results could conceivably be applied to these models.

We now briefly observe what happens when we relate the notion of semi-uniformity to circuit complexity. We can easily define semi-uniformity for circuits. If the complexity class of the semi-uniformity function contains the prediction problem for circuits in the resulting family, then the semi-uniformity condition

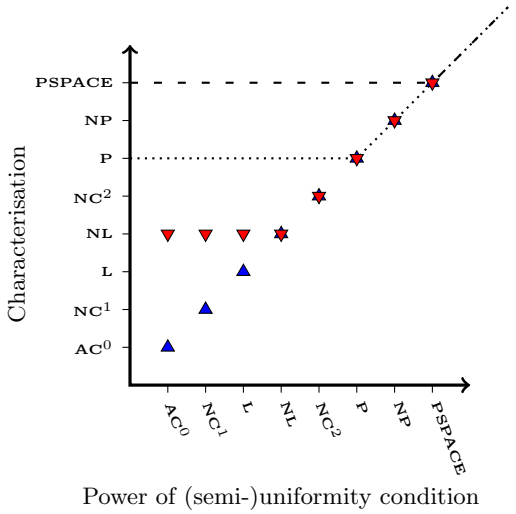

Fig. 1. Complexity classes that are characterised by the membrane systems studied in this paper. Characterisations by uniform systems are denoted by ▲, and semi-uniform by ▼. For example, Theorem 1 is illustrated by the fact that $\mathbf{AC}^0$-uniform systems characterise $\mathbf{AC}^0$, and that $\mathbf{AC}^0$-semi-uniform systems characterise **NL**. The previously known **P** [10] (indicated by ·····) and **PSPACE** [4, 20] (indicated by - -) results, where semi-uniform and uniform classes have the same power are also shown.

characterises the power of the model. If the semi-uniformity function is computable in the class that is characterised by the prediction problem for circuits in the resulting family, then we get the known characterisations for the analogous uniformity condition. However, in the uniform case it is not obvious what happens when we increase the uniformity beyond the power of the circuit, for example **P**-uniform $\mathbf{AC}^0 = \mathbf{AC}^0$ is an open problem [5]. Furthermore we should note that the uniformity condition in membrane systems preprocesses the input (as well as creating the device) and so is a seemingly different notion than circuit uniformity. If we add an analogous preprocessing step to circuits we see similar results as proven here for membrane systems: as soon as the preprocessing goes beyond the power of the circuit, we can ignore the circuit and let the preprocessing solve the problem. With preprocessing below the power of the circuit, the answer depends on the particular circuit model. In fact, if we restrict ourselves to polynomially sized circuits with only OR gates, we would see analogous results to those presented here, (i.e. our work shows that this circuit model is computationally equivalent to the $\mathcal{AM}^0_{-d}$ membrane systems discussed in this work).

1.1 Statement of result

We show that a class of problems, that is characterised by $\mathbf{AC}^0$-uniform membrane systems of a certain type, is a strict subset of another class that is characterised by $\mathbf{AC}^0$-semi-uniformity systems of the same type. Besides their respective use of uniformity and semi-uniformity, both models are identical, so this shows that for the membrane systems we consider, semi-uniformity is a strictly stronger notion than uniformity. Specifically, we show that the uniform systems characterise $\mathbf{AC}^0$ and the semi-uniform systems characterise **NL**, two classes known to be distinct. In the notation of membrane systems this is written as follows (explanations of notation are found in Section 2).

Theorem 1. $\mathbf{AC}^0 = (\mathbf{AC}^0, \mathbf{AC}^0)\text{–}\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subsetneq (\mathbf{AC}^0)\text{–}\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

The left hand equality is proved in this paper, while the right hand equality was given in [11]. In Figure 1, Theorem 1 is illustrated by the leftmost pair of triangles. Essentially, the figure shows that if we use $\mathbf{AC}^0$ uniformity, the systems characterise $\mathbf{AC}^0$, while with $\mathbf{AC}^0$ semi-uniformity they characterise **NL**.

In fact we can also state a more general result for a number of complexity classes below **NL**, for brevity we keep the list short.

Theorem 2. *Let* $C \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}\}$ *and assuming* $\mathbf{NC}^1 \subsetneq \mathbf{L} \subsetneq \mathbf{NL}$ *then* $C = (C, C)\text{–}\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subsetneq (C)\text{–}\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

This shows that, roughly speaking, uniform membrane systems are essentially powerless, they are as weak and as strong as their uniformity condition. In Figure 1, Theorem 2 is illustrated by the triangles to the left of (and including) the uniformity condition **L**.

The essential ideas behind the proof of these theorems are as follows. First, we convert the (complicated looking) membrane systems into a directed acyclic graph called a dependency graph. Acceptance of an input word in some membrane system is equivalent to reachability in the corresponding dependency graph. We observe that for the class of systems that we consider, it is possible to make a number of simplifications to the model (and the dependency graph) without changing the power. In the semi-uniform case, even with these simplifications, the membrane systems have **NL** power. We then go on to prove that in the uniform case, the systems are severely crippled. We show this by proving that even though an arbitrary membrane system's dependency graph may have an **NL**-complete reachability problem, in fact there is an equivalent membrane system where reachability on the dependency graph is in $\mathbf{AC}^0$. This, along with some other tools, is used to show that if the power of the uniformity notion is $\mathbf{AC}^0$ or more, then the power of the entire family of systems is determined by the power of the uniformity.

2 Preliminaries

In this section we define membrane systems and some complexity classes, these definitions are based on those from [9, 13, 14, 15, 20].

The set of all multisets over a set A is denoted by $\mathrm{MS}(A)$. Let $G = (V, E)$ be a directed graph with $x, y, z \in V$. Then let $\mathsf{path}(x, y)$ be true if $x = y$, or $\exists\ z$ s.t $\mathsf{path}(x, z)$ and $\mathsf{path}(z, y)$. Otherwise $\mathsf{path}(x, y)$ is false.

2.1 Active membrane systems

Active membrane systems are a class of membrane systems with membrane division rules. Here division rules act only on elementary membranes, which are membranes that do not contain other membranes (i.e. leaves in the membrane structure).[3] To prove the results in this paper, we convert membrane systems into directed graphs. Thus, in this section, we provide some necessary membrane system definitions, but omit specific example of membrane systems.

Definition 3. *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,* Σ *is the input alphabet,* $\Sigma \subset O$*;*
3. H *is the finite set of labels for the membranes;*
4. μ *is a membrane structure in the form of a tree, consisting of* m *membranes (nodes), labelled with elements of* H*. The parent of all membranes (the root node) is called the "environment" and has label* $env \in H$*;*
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).*

These rules 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.

[3] The more complicated non-elementary membrane division rule is also considered in the literature (where membranes containing other membranes can divide and replicate all of their substructure). All results in this paper hold when we permit non-elementary division, however we omit this detail as it adds unnecessary complications to our definitions and proofs.

- If at the same time a membrane labelled with h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. This process takes only one step.
- 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).

2.2 Recogniser membrane systems

We recall [9] 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). Membrane systems are non-deterministic, therefore on a given input there are multiple possible computations. A computation that reaches a configuration where no more rules are applicable to the existing objects and membranes is called a halting computation.

Definition 4 ([9]). *A* recognizer membrane system *is a membrane system that, on each computation, outputs either object* `yes` *or object* `no` *(but not both), this occurs only when no further 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 an 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 denote by $|x| = n$ the length of any instance $x \in X$. Throughout this paper, $\mathbf{AC}^0$ circuits are **DLOGTIME**-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fanin [7].

Definition 5. *Let $\mathcal{D}$ be a class of membrane systems and let $t : \mathbb{N} \to \mathbb{N}$ 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)–$\mathrm{MC}_{\mathcal{D}}(t)$, 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 $f : \{1\}^* \to \mathbf{\Pi}_X$ such that $f(1^n) = \Pi_X(n)$.*
- *There exists an input encoding function $e : X \to \mathrm{MS}(\Sigma)$ such that $e(x)$ is the input multiset, where $|x| = n$, and the input multiset is placed in a specific (input) membrane of $\Pi_X(n)$.*
- *$\mathbf{\Pi}_X$ is* sound *and* complete *with respect to problem X: $\Pi_X(n)$ starting with the encoding $e(x)$ of input $x \in X, |x| = n$, 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 5 describes (e, f)-uniform families (i.e. with input) and we generalise this to define (h)-*semi-uniform families of membrane systems* $\mathbf{\Pi}_X = \{\Pi_X(x_1), \Pi_X(x_2), \ldots\}$ where there exists a function $h : X \to \mathbf{\Pi}_X$ 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. Also, for each instance of $x \in X$ we have a (potentially unique) membrane system, a clear departure from the spirit of circuit uniformity. The resulting class of problems is denoted by (h)–$\mathrm{MC}^*_{\mathcal{D}}(t)$.

We often refer to $\mathbf{AC}^0$ uniform or logspace uniform (or semi-uniform) families of membrane systems which indicates that the functions e and f (or h) are $\mathbf{AC}^0$ or logspace computable functions.

We define (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{D}}$ (and (h)–$\mathbf{P}\mathrm{MC}^*_{\mathcal{D}}$) as the class of problems solvable by (e, f)-uniform (respectively (h)-semi-uniform) families of membrane systems in polynomial time. We let $\mathcal{AM}^0$ denote the class of membrane systems that obey Definitions 3 and 4. We let $\mathcal{AM}^0_{-d}$ denote the class of membrane systems that obey Definition 3 but where rule (d) is forbidden, and Definition 4.

We let $(\mathbf{AC}^0)$–$\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}}$ denote the class of problems solvable by $\mathbf{AC}^0$-semi-uniform families of membrane systems in polynomial time with no dissolution rules.

Remark 6. A membrane system is said to be *confluent* if it is both sound and complete. That is, a membrane system Π is *confluent* if all computations of Π with the same input x (properly encoded) give the same result; either always accepting or else always rejecting.

In a confluent membrane system, given a fixed initial configuration, the system non-deterministically chooses one from a number of valid configuration sequences, but all of the reachable configuration sequences must lead to the same result, either all accepting or all rejecting.

3 Dependency graphs

A *dependency graph* (first introduced in [9]) represents the rules of membrane systems as a directed acyclic graph (DAG). For many proofs, this representation is significantly simpler and as such is an indispensable tool for characterising the computational complexity of membrane systems (without type (d) dissolution rules).

The dependency graph for a membrane system Π (without type (d) dissolution rules) is a directed graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathtt{in}, \mathtt{yes}, \mathtt{no})$ where $\mathtt{in} \subseteq V_{\mathcal{G}}$ represents the input multiset, and $\mathtt{yes}, \mathtt{no} \in V_{\mathcal{G}}$, represent the accepting and rejecting objects respectively. Each vertex $a \in V_{\mathcal{G}}$ is a pair $a = (o, h) \in O \times H$, where O is the set of objects in Π, and H is the set of membrane labels in Π. An edge (a, b)

exists in $E_{\mathcal{G}}$ if there is a developmental rule in Π such that the left hand side of the rule has the same object-membrane pair as a and the right hand side has an object-membrane pair matching b. In this paper, no membrane dissolution (type (d)) rules are allowed, and so the parent/child relationships of membranes in the structure tree cannot change during the computation. Thus when creating the edges for communication rules (types (b) and (c)) we can find the parent and child membranes for these rules and these choices remain correct for the entire computation (for example, to represent the rule $a[\ ]_h \to [a]_h$, that communicates an object a into a membrane of label h, it is only necessary to calculate the parent of h *one time* in the construction of the dependency graph).

For a number of previous results, it was sufficient to construct the graph $\mathcal{G}$ from Π in polynomial time [9]. For the results in this paper, we make the observation that $\mathcal{G}$ can be constructed from Π in $\mathbf{AC}^0$ (see Appendix A).

4 Proof of main result

The equality on the right hand side of Theorem 1 states that certain $(\mathbf{AC}^0)$-semi-uniform systems characterise **NL**. This was shown in [11], we quote the result:

Theorem 7 ([11]). $(\mathbf{AC}^0)$–$\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

In rest of this paper, we prove the left hand side equality of Theorem 1, that is, we show that the analogous $(\mathbf{AC}^0, \mathbf{AC}^0)$-uniform systems characterise $\mathbf{AC}^0$. We begin by giving two normal forms for the membrane systems that are considered in this paper.

4.1 Normal forms

Lemma 8. *Any confluent $\mathcal{AM}^0_{-d}$ membrane system Π, with m membranes, is simulated by a $\mathcal{AM}^0_{-d}$ membrane system Π', that (i) has exactly one membrane and (ii) uses only rules of type (a). (By* simulate *we mean that the latter system accepts x iff the former does.)*

Proof (sketch). Given membrane system Π we construct its dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathtt{in}, \mathtt{yes}, \mathtt{no})$. We observe that we can convert $\mathcal{G}$ into a new membrane system $\Pi' = \Pi_{\mathcal{G}}$ by simply converting the edges of the graph into object evolution rules. Specifically, the set of objects of $\Pi_{\mathcal{G}}$ is $O_{\mathcal{G}} = V_{\mathcal{G}}$, and there is a single (environment) membrane of label *env*. The rules of $\Pi_{\mathcal{G}}$ are $\{[\, v \to \mathrm{str}(v)]_{env} \mid\ v \in V_{\mathcal{G}}\}$ where $\mathrm{str}(v)$ is the string formed by concatenating the elements of the set $\{s \mid (v, s) \in E_{\mathcal{G}}\}$. The vertices $\mathtt{yes}, \mathtt{no}$ are mapped to the `yes` and `no` objects respectively, and the set of vertices `in` becomes the input multiset of objects (actually an input *set*). This construction of $\Pi_{\mathcal{G}}$ (from $\mathcal{G}$) is $\mathbf{AC}^0$-computable.

For correctness, notice that the dependency graphs of Π and $\Pi_{\mathcal{G}}$ are isomorphic, so one accepts iff only the other does. Furthermore, $\Pi' = \Pi_{\mathcal{G}}$ has exactly one membrane with label *env*, and uses only type (a) rules (object evolution). □

Lemma 9. *Any confluent $\mathcal{AM}^0_{-d}$ membrane system Π, which has, as usual, multisets of objects in each membrane is simulated by another $\mathcal{AM}^0_{-d}$ membrane system Π', which has sets of objects in each membrane. (By* simulate *we mean that the latter system accepts x iff the former does.)*

Proof (sketch). We observe that in a dependency graph, $\mathcal{G}$, the multiset of objects is encoded as a *set* of vertices, so no information is stored regarding object multiplicities. Thus if we convert $\mathcal{G}$ into a new membrane system, Π' (as in the proof of the previous lemma), there are no rules in Π' with a right hand side with more than one instance of each object. The resulting system Π' accepts iff Π accepts since the dependency graphs of both systems are isomorphic. □

4.2 Uniformity is not equal to semi-uniformity

The following theorem is key to the proof of our main results (Theorems 1 and 2). Roughly speaking, Theorem 10 states that in uniform membrane systems of the type we consider, the uniformity condition dominates the computational power of the system. By letting $E = F = \mathbf{AC}^0$, the statement of Theorem 10 gives us the left hand side equality in Theorem 1. By letting $E = F \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}\}$ we get the left hand side of Theorem 2. The remaining classes quoted in the theorem serve to illustrate Figure 1.

Theorem 10. *Let $E, F \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}, \mathbf{NL}, \mathbf{NC}^2, \mathbf{P}, \mathbf{NP}, \mathbf{PSPACE}\}$ and let $F \subseteq E$. Then (E, F)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} = E$.*

Proof. Let $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ be the dependency graph of confluent recogniser membrane system Π from the class $\mathcal{AM}^0_{-d}$. We define the following subsets of the vertices of $V_\mathcal{G}$. Let $V_{\mathcal{G}_{\mathtt{yes}}} = \{v \mid v \in V_\mathcal{G}, \mathsf{path}(v, \mathtt{yes})\}$, $V_{\mathcal{G}_{\mathtt{no}}} = \{v \mid v \in V_\mathcal{G}, \mathsf{path}(v, \mathtt{no})\}$, and $V_{\mathcal{G}_{\mathrm{other}}} = V_\mathcal{G} \backslash (V_{\mathcal{G}_{\mathtt{yes}}} \cup V_{\mathcal{G}_{\mathtt{no}}})$.

We claim that $V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}} = \emptyset$. Assume otherwise, and let vertex $v \in V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}}$. This implies that $\mathsf{path}(\mathtt{in}, \mathtt{yes})$ and $\mathsf{path}(\mathtt{in}, \mathtt{no})$ are both true, which contradicts Definition 4 which states that *only* a `yes` or *only* a `no` object may be output by the system Π.

Next we claim that for confluent recogniser membrane systems Π from the class $\mathcal{AM}^0_{-d}$, a size-two input alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ is both necessary and sufficient, in the sense that this restriction does not alter the computing power of the system Π. Again, consider $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, the dependency graph of Π. The "input" vertices $\mathtt{in} \subseteq V_\mathcal{G}$, represent the input objects of Π. On the one hand, it is necessary that $|\ \mathtt{in}\ | \geq 2$. This follows from the fact that it is necessary that both `yes` and `no` are reachable, and the fact that $V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}} = \emptyset$. Thus we need one vertex in $V_{\mathcal{G}_{\mathtt{yes}}}$ and another vertex in $V_{\mathcal{G}_{\mathtt{no}}}$. On the other hand, it can be seen that a single vertex from each set $V_{\mathcal{G}_{\mathtt{yes}}}, V_{\mathcal{G}_{\mathtt{no}}}$ is sufficient as follows. Given a set S of input vertices in $V_{\mathcal{G}_{\mathtt{yes}}}$, there is another system with an extra vertex, where all edges from this extra vertex lead to all vertices in S. A vertex can be analogously added for $V_{\mathcal{G}_{\mathtt{no}}}$. So even though membrane system Π may have multiple input objects, there is a

Π' that is equivalent in all respects except that there are exactly two input objects some extra rules. In particular, Π' accepts input x iff Π does.

The previous argument permits us to consider only those systems that have two input objects $\{\mathtt{a}, \mathtt{b}\}$. Thus we restrict attention to the case that the input encoding function is of the form $e : X \to \{\mathtt{a}, \mathtt{b}\}$. We say that e is a characteristic function with range $\{\mathtt{a}, \mathtt{b}\}$.

Let (e, f)–$\mathfrak{P}$ be a (e, f)-uniform family of confluent recogniser membrane systems from the class $\mathcal{AM}^0_{-d}$ that solves problem X. We claim that there exists a family (e, f')–$\mathfrak{P}$ that also solves X but uses a uniformity function f' that produces membrane systems of a very restricted form. Consider the dependency graph of the membrane system $f(n) = \Pi_X(n)$. In terms of reachability from $\{\mathtt{a}, \mathtt{b}\}$ to $\{\mathtt{yes}, \mathtt{no}\}$ in this graph (which corresponds to accepting/rejecting in the membrane system), the essential property is whether $\mathtt{path}(\mathtt{a}, \mathtt{yes})$ is true or $\mathtt{path}(\mathtt{a}, \mathtt{no})$ is true. However, this essential property is captured by the following (extremely simple) pair of dependency graphs: $\mathcal{G}_1 = (V_\mathcal{G}, E_{\mathcal{G}_1})$ and $\mathcal{G}_2 = (V_\mathcal{G}, E_{\mathcal{G}_2})$ where $V_\mathcal{G} = \{\mathtt{a}, \mathtt{b}, \mathtt{yes}, \mathtt{no}\}$, $E_{\mathcal{G}_1} = \{(\mathtt{a}, \mathtt{yes}), (\mathtt{b}, \mathtt{no})\}$, and $E_{\mathcal{G}_2} = \{(\mathtt{a}, \mathtt{no}), (\mathtt{b}, \mathtt{yes})\}$. Therefore if there is a family (e, f)–$\mathfrak{P}$ that solves X, where f represents valid, but arbitrary, dependency graphs, then there is another family of the form (e, f')–$\mathfrak{P}$ that also solves X and is identical in every way except that f' represents dependency graphs of the restricted form just described.

Now we prove the upper bound (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subseteq E$, where e, f are respectively computable in E, F, with $F \subseteq E$, and the classes E, F are as given in the statement. As we have just shown, for each problem X in the class (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}}$ there is a family of the restricted form (e, f')–$\mathfrak{P}$ that solves X. To simulate (e, f')–$\mathfrak{P}$ with a given input $x \in X$ we compute the pair $\left(e(x), f'(1^{|x|})\right)$. The range of the function f' is two membrane systems, which correspond to the two (restricted) dependency graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ from above, and whose reachability problems are (trivially) in the weakest E that we consider ($E = \mathbf{AC}^0$). Furthermore, since we only consider the case where $F \subseteq E$ then we know that f' itself is computable in E and we have (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subseteq E$.

The lower bound $E \subseteq (e, f)$–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}}$ is easy to show. We use the fact, shown above, that e is a characteristic function with access to the input word. Thus the following simple family computes any problem from E: function $e(x)$ outputs $\mathtt{a}$ if x is a positive instance of X and $\mathtt{b}$ if x is a negative instance of X, and f simply maps $\mathtt{a}$ to $\mathtt{yes}$ and $\mathtt{b}$ to $\mathtt{no}$. □

Acknowledgements

Niall Murphy is funded by the Irish Research Council for Science, Engineering and Technology. Damien Woods is supported by a Project of Excellence from the Junta de Andaluca, grant number TIC-581.

References

1. Leonard Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
2. Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P Systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
3. Artiom Alhazov and Linqiang Pan. Polarizationless P Systems with active membranes. *Grammars*, 7:141–159, 2004.
4. Artiom Alhazov and Mario J. Prez-Jimnez. Uniform solution to QSAT using polarizationless active membranes. In Jrme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations and Universality (MCU)*, volume 4664 of *LNCS*, pages 122–133, Orlans, France, September 2007. Springer.
5. Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In Osamu Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, chapter 1, pages 4–22. Springer, 1992.
6. Martyn Amos. *Theoretical and Experimental DNA Computation*. Natural Computing Series. Springer, 2005.
7. David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
8. Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. Technical report, Caltech Parallel and Distributed Systems Group [http://caltechparadise.library.caltech.edu/perl/oai2] (United States), 2008. In submission.
9. Miguel A. Gutirrez-Naranjo, Mario J. Prez-Jimnez, Agustn Riscos-Nez, and Francisco J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, 83(7):593–611, 2006.
10. Niall Murphy and Damien Woods. Active membrane systems without charges and using only symmetric elementary division characterise P. *8th International Workshop on Membrane Computing, LNCS*, 4860:367–384, 2007.
11. Niall Murphy and Damien Woods. A characterisation of NL using membrane systems without charges and dissolution. *Unconventional Computing, 7th International Conference, UC 2008, LNCS*, 5204:164–176, 2008.
12. Ian Parberry. *Circuit complexity and neural networks*. MIT Press, 1994.
13. Mario J. Prez-Jimnez, Alvaro Romero-Jimnez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, 2003.
14. Gheorghe Pun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
15. Gheorghe Pun. *Membrane Computing*. Springer-Verlag, Berlin, 2002.
16. Gheorghe Pun. Further twenty six open problems in membrane computing. In *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain)*, pages 249–262. Fnix Editoria, January 2005.
17. David Soloveichik and Erik Winfree. The computational power of Benenson automata. *Theoretical Computer Science*, 344:279–297, 2005.
18. David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM J. Comput.*, 36(6):1544–1569, 2007.
19. Petr Sosk. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, August 2003.

20. Petr Sosk and Alfonso Rodrguez-Patn. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73(1):137–152, 2007.

Appendix A

A.1 Constructing dependency graphs in $\mathbf{AC^0}$

We are given a binary string x that encodes a membrane system, Π. To make a dependency graph from a membrane system requires a constant number of parallel steps that are as follows. First, a row of circuits identifies all communication (type (b) and (c)) rules and uses the (static) membrane structure to determine the correct parent membranes, then writes out (a binary encoding of) edges representing these rules. Next, a row of circuits writes out all edges representing division (type (e)) rules, for example a rule $[\ a\ ]_h \rightarrow [\ b\ ][\ c\ ]$ becomes the edges $(a_h, b_h), (a_h, c_h)$ in the dependency graph. In the final step we deal with evolution (type (a) rules) where it is possible to have polynomially many copies of polynomially many distinct objects on the right hand side of a rule (e.g. $[\ a\ ]_h \rightarrow [\ bcbbcdee\cdots z\ ]_h$). To write out edges for these rules in constant time we take advantage of the fact that we require at most one edge for each object-membrane pair in $O \times H$. We have a circuit for each element of $\{o_h \mid o \in O, h \in H\}$. The circuit for o_h takes as input (an encoding of) all rules in R whose left hand side is of the form $[o]_h$. The circuit then, in a parallel manner, masks (an encoding of) the right hand side of the rule (for example $[bbcdc]_h$) with the encoding of each object in O, (in the example, masking for (encoded) b would produce (encoded) $bb000$). All encoded objects in the string are then ORed together so that if there was at least one copy of that object in the system we obtain a single instance of it. The circuit being unique for a specific left hand side $[o]_h$ now writes out an encoding of the edge (o_h, b_h) and an encoding of all other edges for objects that existed on the right hand side of this rule in parallel.