Open problems in membrane computing and how not to solve them

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

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

Abstract. We present some high-level open problems in the complexity theory of membrane systems, related to the actual computing power of confluence vs determinism, semi-uniformity vs uniformity, deep vs shallow membrane structures, membrane division vs internal evolution of membranes. For each of these problems we present some reasonable approaches that are, however, unable to be employed "as-is" to provide a complete solution. This will hopefully sparkle new ideas that will allow tackling these open problems.

1 Introduction

It is common in most of the literature, not only of P systems but of science in general, to highlight the successful approaches, not only the ones that gave positive results, but also the ones that were successful in proving the negative ones. It is considerably rarer to write about the unsuccessful ideas, the roads that, while initially promising, were unable to give a full and satisfactory answer to a research question under examination. Unfortunately, this leads multiple people to follow the same path multiple times without the insight that might have been gained by sharing previous failed attempts. Here we want to take the opportunity to share some problems that we consider of particular interest for the membrane computing community but, since they are still open problems, we are not able to share a solution to them. What we want to share is, instead, a collection of promising attempts that were ultimately unsuccessful in solving these open problems. While unsuccessful, those attempts were far from useless: we were able to gain a deeper understanding of the problems and to solve some special cases. We think that, by sharing our non-working approaches we might be able to help the community in gaining a better understanding of these open problems and, hopefully, produce inspiration on how to solve them.

2 Confluence vs determinism

P systems solving decision problems (*recogniser P systems* [16]) are usually required to be *confluent* [16] rather than strictly deterministic. That is, they are

allowed to result in multiple computations, as long as all of them agree on the final result, which is acceptance or rejection.

This sometimes simplifies the presentation of some algorithms. For instance, a classic membrane computing technique [14, 22] consists in generating all 2^n truth assignments of n variables by using membrane division rules of the form $[x_i]_h \to [t_i]_h\ [f_i]_h$, with $1 \leq i \leq n$. The membrane division is triggered separately in each membrane with label h by one of the objects x_i, nondeterministically chosen at each computation step. Irrespective of all such nondeterministic choices, the final result is invariably a set of 2^n membranes, each containing a different truth assignment.

Notice, however, that this kind of nondeterminism can be completely avoided by serialising the generation of truth assignments for each variable: first all instances of x_1 trigger the division, then all instances of x_2, and so on. This can be achieved by adding an extra subscript to each object, which counts down to zero and only then starts the division process.

It is often the case that confluent nondeterminism can be avoided in a similar way, although this is usually proved by exhibiting a deterministic algorithm, rather than showing how to remove the nondeterminism from existing algorithms. It is then natural to ask whether this is indeed always the case, or if there exists a variant of P systems where confluent nondeterminism is strictly stronger than determinism.

For powerful enough P systems (e.g., able to efficiently simulate deterministic Turing machines, or stronger than that) we feel that the existence of such a variant would be very surprising, although there do exist confluent nondeterministic algorithms with no known deterministic version. For instance, the currently known proof of efficient universality (i.e., the ability to simulate any Turing machine with a polynomial slowdown) of P systems with active membranes using elementary membrane division [1] relies on a massive amount of nondeterministic choices performed at each simulated step; these are due to the fact that send-in communication rules cannot differentiate among membranes having the same label and electrical charges.

2.1 Simulation of priorities

To better examine the problem given by nondeterminism, even in the case of confluence, we will use a simple example. Consider the following two rules:

$$[a]_h^+ \to [\ ]_h^0\ b \tag{1}$$

$$[a \to w']_h^+ \tag{2}$$

When an instance of an object a is present inside a membrane with label h and positive charge, both rule (1) and rule (2) are applicable and, in a nondeterministic system, a nondeterministic choice is performed. If the P system is confluent, then the actual outcome of this choice is immaterial. However, if we know that the P system is a *deterministic* one then we can also infer a stronger condition: there will never be any instance of objects of type a inside any membrane with label h

with charge + in the only computation starting from the initial configuration. otherwise, there would be two different computations generated by the presence of a, contradicting the assumption that the P system is deterministic.

If we want to simulate a confluent system by means of a deterministic one in a somewhat direct way, then we must take care of situations like the one above. How can we simulate a situation that a deterministic P system cannot even reach? One such approach is the introduction of rule priorities. As shown in [10], for P systems with active membranes with charges the introduction of rule priorities *does not* change the computational power of confluent systems when the bound on computation time is at least polynomial.

Why it might work Rule priorities provide a way of addressing the problem of the example above and the more general problem of conflicts among rules. Suppose that in the previous example rule (1) has higher priority than rule (2). Then, there is no conflict between the two rules: the first one will always be applied if no other blocking rule with higher priority has already been applied, and the second one will be applied to all remaining copies of a. In fact, once a total ordering has been provided among all rules then no further conflict can happen: among two rules one will always have higher priority than the other. Therefore, now in a deterministic system we can have objects of type a inside a membrane with label h and positive charge, since their presence will not generate two distinct computations anymore.

Why it does not work If with rule priorities it is never possible to obtain two distinct computations due to a conflict between rules, since there are no conflicts anymore, then have we found a way to have one single computation starting from the initial configuration? Have we found a way to obtain determinism from confluence? The answer is, unfortunately, negative. The main problem is given by *send-in* communication rules. Consider, for example, the following rule:

$$a\ [\ ]_h^+ \to [b]_h^- \tag{3}$$

Furthermore, suppose that there are two membranes with label h and positive charge inside the membrane where a is located. If there is a single instance of a then rule (3) can send in a in either of the two membranes in a nondeterministic way. If the contents of the two membranes differ, then two computations are actually generated. In a confluent system this is not a problem, but for a deterministic system the application of a send-in rule is valid in only two situations:

- There are enough instances of object a to be sent-in inside all the membranes where rule (3) was applicable;
- The contents of all the membranes where rule (3) is applicable were actually the same.

This problem cannot be solved by rule priorities; in fact, there is no rule conflict in the example that we just presented. This is the main difference between the nondeterminism introduced by send-in rules and the one introduced by rule conflicts. This remains the main obstacle in showing that confluence and determinism give, for powerful enough systems, the same computational power.

3 Semi-uniformity vs uniformity

Recogniser P systems usually appear in families $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$, where each member of the family is associated to a string x and accepts if and only if x belongs to a given language.

A family of P systems is usually required to be at least *semi-uniform*, that is, to have an associated Turing machine M with some suitable resource bound (usually, polynomial time) such that M on input x outputs a suitable encoding of Π_x [16, 11].

A more restrictive condition on families of P systems is full-fledged *uniformity* [16, 11]: there exist *two* Turing machines F and E (again, usually with polynomial runtime) such that F on input $n = |x|$ constructs a P system "skeleton" Π_n, valid for all strings of length n, and E on input x produces a multiset w encoding x, which is then placed inside the input region of Π_n, giving the P system Π_x that computes the answer.

It is known [12] that, for restrictive enough resource bounds, uniformity is weaker than semi-uniformity. However, when polynomial-time semi-uniform solutions to problems sometimes appear in the literature first, polynomial-time uniform solutions usually follow.

We conjecture that polynomial-time uniformity and semi-uniformity do indeed coincide for powerful enough P systems, such as standard P systems with active membranes [14]. The idea here is that a semi-uniform family could be made uniform by simulating the "semi-uniform portion" of the construction, depending on the actual symbols of $x \in \Sigma^n$, with the P system constructed for all strings of length n.

3.1 Building and filling the membrane structure at runtime

Given a semi-uniform family of P systems $\boldsymbol{\Pi}$ constructed by a machine M we want to build two machines E and F that define a uniform family of P systems that solves the same problems that are solved by the systems in $\boldsymbol{\Pi}$.

One of the possible ideas to prove the equivalence of uniformity and semi-uniformity, at least for powerful enough kinds of P systems, is to capture the power of machine E of the uniformity condition, i.e., the one that has access to the entire input and not only to its length. While that machine does not have the ability construct the membrane structure of the P system or even to put objects inside membranes different from the input ones, it can perform the same operations of machine M of the semi-uniformity condition, therefore obtaining a "copy" of the initial membrane structure of the P system. Can we make use of such knowledge to overcome the limitation of uniformity and show that we can, in fact, simulate semi-uniformity?

Why it might work With the knowledge of the initial membrane structure and the content it is quite easy to build, for each object type a new types of objects of the form a_p where p is a path inside the membrane structure. It is usually not hard to write rules "consuming" the path p while moving the objects around following

the directions stored in p. Therefore, in polynomial time it is possible to move objects around the membrane structure. Since the objects in the initial system built by M on input x might depend on the input, we can delegate the creation of all the initial objects to machine E. The multiset produced as output of E on input x will contain all the objects present in the initial membrane structure built by M. However, that objects will be subscripted by a path as shown before. In this way they will be able to reach the correct position before starting to act like the objects in the system built by M.

Why it does not work A larger problem than the one tackled above is the fact that there is no assurance that the membrane structure generated by machine M will be the same on two different inputs x and y even when they are of the same length. We might think that machine F of the uniformity condition might be able to build a membrane structure that is the "sum" of all possible membrane structures that machine M can generate for inputs of a certain length. By looking at the literature it is possible to observe that this is an effective method to convert semi-uniform families to uniform ones. This method, however, will not work in general. A combinatorial analysis shows that the membrane structures that can be generated by machine M are too many to obtain a polynomially-sized membrane structure that contains all of them as substructures. The sharp contrast between the efficacy of this method in practice and its viability as a formal tool to prove the equivalence of uniformity and semi-uniformity leaves us with the following question: is a super-polynomial number of different membrane structures for inputs of the same length actually useful? Can we prove that this is never the case?

4 Membrane division vs internal evolution

The computing power of a *single* membrane (for cell-like P systems) or cell (for tissue-like P systems) working in polynomial time usually has a **P** upper bound, as already proved by the "Milano theorem" [22]; the only way to exceed this bound would be to include *really overly powerful* rules (e.g., rules able to perform an **NP**-complete task in a single step). The **P** upper bound can actually be achieved by having cooperative rewriting rules (even minimal cooperation [21, 20] suffices) or rules able to simulate them indirectly (e.g., active membrane rules with membrane charges [10]). Several techniques for simulating polynomial-time Turing machines using a single membrane are known [9].

Any additional power beyond **P** of models presented in the literature is due to membrane division, first exploited in order to solve **NP**-complete problems in polynomial time [14]. Membrane division enables us to create exponentially many processing units working in parallel; by using communication rules, these can synchronise and exchange information (this is the famous space-for-time trade-off in membrane computing).

It is reasonable to expect that P system variants where the power of a single membrane working in polynomial time coincides with **P** can be standardised in

a "Turing machine normal form": each membrane performs a Turing machine simulation[1], and the communication and division rules implement a network, whose shape can be exploited to simulate nondeterminism, alternation, or oracle queries [9].

Notice that what previously described does not necessarily carry over to variants of P systems with weaker rules internal to the membranes, such as "P conjecture systems" [15, Problem F] (active membranes without charges), which do not seem able to simulate cooperation [2], or with communication restricted to a single direction, either send-out [6, 7, 19] or send-in only [18].

4.1 Putting a Turing machine inside a membrane

One approach that is at first glance promising to solve the problem of characterising the computational power of a single membrane is to simply substitute the content of a membrane with a Turing machine simulated by that membrane, that makes its behaviour indistinguishable from the one of the original membrane. For an example, we will use P systems with more than three charges, like the ones in [10].

Let M be a Turing machine with set of states Q, alphabet Σ, transition function δ, and working in space n. Then it can be simulated by a single membrane M_h with the following kinds of rules:

$$[a_i \to b_i\ r_d]^{q_i}_{M_h} \qquad \text{for } q, r \in Q,\ a, b \in \Sigma,\ 1 \le i \le n, \text{ and } \delta(q, a) = (r, b, d)$$

$$[r_d]^{q_i}_{M_h} \to [\ ]^{r_{i+d}}_{M_h} \# \qquad \text{for } q, r \in Q,\ r \in \{-1, 1\}, \text{ and } 1 \le i \le n$$

The main idea is that in the objects inside the membrane it is possible to store the tape of the machine and the state and position of the tape head are stored in the charge of M_h. By alternating evolution rules (to rewrite the tape) and send-outs (to update the state and position of the tape head) it is possible to simulate an entire Turing machine inside a single membrane. How can we use this to replace the entire inner working of a membrane?

Why it might work If we consider an isolated membrane and we investigate its behaviour from the outside we are only interested in what is sent out and what is sent in. That is, if we are unable to distinguish a membrane h and a membrane simulating machine M that simulates h then the computational power of a single membrane cannot be greater than the one of machine M and, if M is a deterministic machine working in polynomial time then all the power of a P system comprised of such membranes must reside in the ability of the membranes to divide. Since most variants of P systems when limited to a single membrane are subjected to the Milano theorem, we can think that, with a careful enough "interfacing" with the objects sent in from the environment (which, when they enter, are not part of the machine tape and must be "incorporated" into it

[1] This can be trivially implemented by having each membrane simulate a Turing machine which, in turn, simulates the original membrane via the Milano theorem.

by other rules) we might be able to replicate (with a polynomial slowdown) the entire behaviour of a membrane h with the machine M simulated inside another membrane. And, in fact we can. So, why this does not prove that we can replace all the "inner working" of a membrane with a Turing machine?

Why it does not work When considering a membrane in isolation we can easily replace it with a Turing machine - and we can replace the Turing machine with a membrane simulating it. There is, however, one major problem. Consider the following membrane structure:

$$[\underbrace{[\,]_k \cdots [\,]_k}_{2^n}]_h$$

If we replace the "inner working" of membrane h with the simulation performed by a Turing machine, then at the moment when the membranes with label k send out – all at the same time – 2^n instances of the same object a then the simulation inside h is not sufficient anymore. It is not possible to write all the objects (either one at time or all together) on the Turing machine tape for reasons either of space (the tape is not long enough) or time (writing them one at a time requires exponentially many time steps). We would like to write, instead, the number of objects of type a that have entered membrane h by send-out. This, however, requires the power to count or, more precisely, the power to convert from unary to binary the number of objects. While this is possible with cooperative rewriting rules, it is unclear if for P systems with charges this is a task feasible for a single membrane.

5 Deep vs shallow membrane structures

Let us now consider cell-like P systems with membrane division, for instance P systems with active membranes [14]. It has already been shown that the nesting depth of membranes (more specifically, the nesting depth of membranes with associated division rules, which we might call *division depth*) is one of the most influential variables when establishing the efficiency of these P systems.

Indeed, P systems without membrane division (i.e., with division depth 0) are known to characterise the complexity class **P** in polynomial time [22]. At the other end of the spectrum, we have P systems with active membranes with elementary and non-elementary division rules (i.e., with polynomial division depth), which characterise **PSPACE** in polynomial time.

When only elementary membrane division is allowed (i.e., division depth 1), then the intermediate complexity class $\mathbf{P}^{\#\mathbf{P}}$ is characterised in polynomial time [3, 7]. This class contains all decision problems solved by deterministic polynomial-time Turing machines with oracles for counting problems in the class $\#\mathbf{P}$ [13].

It has been proved that moving from any constant division depth d to division depth $d+1$ allows the P systems to simulate Turing machines with more powerful oracles [4]. We conjecture that this is in fact a proper hierarchy. This result would require proving the upper bounds corresponding to the known lower bounds.

It also remains open to characterise the computing power of polynomial-time P systems with other division depths, such as $O(\log n)$.

More profound reasons for the split The reason for this split between shallow and deep membrane structures seems to be related to the way confluent systems perform a "simulation" of non-determinism by means of parallel computations inside different membranes. Shallow membrane structures can perform exponentially many computations (and each computation can perform the same work of a polynomial-time Turing machine) inside exponentially many elementary membranes; the results of the different computations are then collected inside the outermost membrane. P systems with a deep membrane structure can also perform the same number of computations, but the main difference is that they can combine the results in a more structured way: the results are, in some sense, organized hierarchically. When non-confluence is allowed, the importance of the membrane structure seems to fade away: it is known that shallow membrane structures are able to achieve the full power of **PSPACE** [8]. It is therefore natural to ask what is the relation between the power granted by non-confluence and the one obtained by having a deep membrane structure.

It seems that the problem of the computational power that depth gives to P systems is a more profound question related to the computational power that automata (or limited-resources Turing machines) can gain with different communication topologies. For example, with a polynomial time limit, automata on a grid are usually limited to the power of a deterministic Turing machine working in polynomial time. Similar results also hold for tissue P systems embedded in the Euclidean space [5], showing another point of contact with the more general problem. This approach linking complexity and communication topologies is quite new [17] and we think that it might be able to give us a better understanding of the link between membrane depth and computational power in P systems.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. Theoretical Computer Science 529, 69–81 (2014)
2. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 6th International Workshop, WMC 2005. Lecture Notes in Computer Science, vol. 3850, pp. 224–240. Springer (2006)
3. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014. Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2014)
4. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae 138(1–2), 97–111 (2015)

5. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Tissue P systems in the Euclidean space. In: Gheorghe, M., Petre, I., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Multidisciplinary Creativity: Homage to Gheorghe Păun on His 65th Birthday, pp. 118–128. Editura Spandugino (2015)
6. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)
7. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theoretical Computer Science 701, 161–173 (2017)
8. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Shallow non-confluent P systems. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 307–316. Springer (2017)
9. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Subroutines in P systems and closure properties of their complexity classes. In: Graciani, C., Păun, G., Riscos-Núñez, A., Valencia-Cabrera, L. (eds.) 15th Brainstorming Week on Membrane Computing, pp. 115–128. No. 1/2017 in RGNC Reports, Fénix Editora (2017)
10. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: A toolbox for simpler active membrane algorithms. Theoretical Computer Science 673, 42–57 (2017)
11. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
12. Murphy, N., Woods, D.: Uniformity is weaker than semi-uniformity for some membrane systems. Foundamenta Informaticae 134(1–2), 129–152 (2014)
13. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
14. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
15. Păun, G.: Further twenty six open problems in membrane computing. In: Gutíerrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 249–262. Fénix Editora (2005)
16. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
17. Porreca, A.E.: Communication topologies in natural computing (2018), seminar of the CANA research group, Laboratoire d'Informatique et Systèmes (LIS), UMR 7020, Aix-Marseille Université, CNRS, Université de Toulon, France https://aeporreca.org/talks/
18. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Restricted polarizationless P systems with active membranes: Minimal cooperation only inwards. In: Graciani, C., Păun, G., Riscos-Núñez, A., Valencia-Cabrera, L. (eds.) 15th Brainstorming Week on Membrane Computing, pp. 215–252. No. 1/2017 in RGNC Reports, Fénix Editora (2017)
19. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Restricted polarizationless P systems with active membranes: Minimal cooperation only outwards. In: Graciani, C., Păun, G., Riscos-Núñez, A., Valencia-Cabrera, L. (eds.) 15th Brainstorming Week on Membrane Computing, pp. 253–290. No. 1/2017 in RGNC Reports, Fénix Editora (2017)

20. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. Fundamenta Informaticae 153(1–2), 147–172 (2017)
21. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation. Theoretical Computer Science 701, 226–234 (2017)
22. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)