An Approach to Computational Complexity in Membrane Computing

Mario J. Pérez-Jiménez

Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
Mario.Perez@cs.us.es

Abstract. In this paper we present a theory of computational complexity in the framework of membrane computing. Polynomial complexity classes in recognizer membrane systems and capturing the *classical* deterministic and non-deterministic modes of computation, are introduced. In this context, a characterization of the relation $\mathbf{P} = \mathbf{NP}$ is described.

1 Introduction

The necessity to define in a satisfactory way what means a *definite method* for solving mathematical problems was studied by A. Turing who investigated how such methods should be applied *mechanically*, and, moreover, he formalized the task of performing such methods in terms of the operations of a *machine* able to read and write symbols on a tape divided into parts called cells (simulating how a person can solve a problem with paper and pencil manipulating symbols).

The *theory of computation* deals with the *mechanical solvability* of problems, that is, searching solutions that can be described by a finite sequence of elementary processes or instructions. The first goal in the theory of computation is general problem solving; that is, to develop principles and special methods that are able to solve any problem from a certain class of questions.

A *computational model* tries to capture those aspects of mechanical solutions of problems that are relevant to these solutions, including their inherent limitations. In some sense, we can think that computational models design machines according to certain necessity.

From a practical point of view, the goal of computation theory is to take real–life problems and try to solve them through a method capable of being simulated by a machine when we use a suitable language to communicate that problem to the machine (a language is a system of signs used to communicate information between different parties).

Abstract machines are formal computing devices that we use to investigate properties of real computing devices. Computable languages are a special type of formal languages that can be processed by abstract machines that represent computers.

If we have a mechanically solvable problem and we have a specific algorithm solving it that can be implemented in a real machine, then it is very important to know how much computational resources (time or memory) are required for a given instance, in order to recognize the limitations of the real device.

One of the main goals of the *theory of computational complexity* is the study of the efficiency of algorithms and their data structures through the analysis of the resources required for solving problems (that is, according to their intrinsic computational difficulty). This theory provides a classification of the *abstract problems* that allows us to detect their inherent complexity from the computational solutions point of view.

Of course, such a classification demands a precise and formal definition of the concept of *abstract problem* and the model to be considered.

The following parameters are used to specify a *complexity class* within a general computational framework:

- First: the *model* of computation, D (in our case, recognizer P systems).
- Second: the *mode* of computation, M (in our case, non-deterministic and parallel).
- Third: the resource, r, that we wish to bound (usually time and space).
- Finally, we must specify an upper *bound* of the resources, f (a total recursive function from natural numbers to natural numbers).

Then, a complexity class is defined as the set of all languages decided by the device D operating in mode M and such that for any input string, u, D expends at most $f(|u|)$ units of the resource r, to accept or reject u.

Many interesting problems of the real world are presumably intractable and hence it is not possible to execute algorithmic solutions in an electronic computer when we deal with instances of those problems whose size is large. The theoretical limitations of the Turing machines in terms of computational power are also practical limitations to the digital computers.

Natural Computing is a new computing area inspired by nature, using concepts, principles and mechanisms underlying natural systems. *Evolutionary Algorithms* use different concepts from biology. *Neural Networks* are inspired in the structures of the brain and nervous system. *DNA Computing* is based on the computational properties of DNA molecules and on the capacity to handle them. *Membrane Computing* is inspired by the structure and functioning of living cells.

These two last models of computation provide unconventional devices with an attractive property (*computational efficiency*), they are able to create an exponential workspace in polynomial time (and, in some sense, trading space for time).

Can some unconventional devices be used to solve presumably intractable problems in a feasible time? The answer is affirmative at least from a theoretical point of view.

In this paper we provide a systematic and formal framework for the design of polynomial solutions to hard problems, and to classify them according to their polynomial solvability by cell–like membrane systems. Complexity classes in the framework of membrane computing and their relationships with the problems they contain, are the main subjects of this paper.

The paper is structured as follows. The next section is devoted to describe in an informal way the deterministic and non-deterministic mode of computation in a computing model. In Sections 3, 4, and 5 combinatorial optimization problems and decision problems are introduced, and a relationship between them from a complexity point of view is showed. The **P** versus **NP** problem is presented in Section 6, and in Section 12 a characterization of that problem is obtained. In Section 7 weakly and strongly **NP**–complete problems are studied. Sections 8 and 9 are devoted to present the general framework (recognizer membrane systems) within a theory of computational complexity developed here. Deterministic and non-deterministic polynomial complexity classes in membrane systems are introduced in Sections 10 and 11. Finally, we study the P systems with the capability to construct an exponential workspace in polynomial time, and the polynomial complexity classes associated with them.

2 Determinism Versus Non-determinism

A model of computation is properly given when we formally define the concept of mechanical procedure (*algorithm*). For that, it is necessary to syntactically define it, and determine precisely how such procedures can be executed (the semantic of the model).

The devices (systems or machines) modelling mechanical procedures can be represented through *configurations* (containing a complete description of the current state of the device). These configurations can evolve according to the semantic of the model. Formally, the semantic defines the concept of *transitions* from a configuration of the system to a next configuration; that is, the semantic of the model specifies what means *next configuration* of a given configuration. A configuration which has no next configuration is called a *halting configuration*.

A *computation* or execution of a device of a model is a sequence (finite or infinite) of configurations such that each configuration (except the first) is obtained from the previous one by a (step of) transition of the system. That is, a computation starts with an *initial configuration* of the system, and then it proceeds step by step, and halts when reaches a halting configuration (and then the result is encoded in this configuration).

When we use the devices of a model of computation to solve certain kind of problems on strings (in particular to *recognize* a language), it is necessary to define what means to *accept* or *reject* a string. In this case it is possible to consider two *modes of computation* in a computing model.

– The *deterministic mode* is characterized by the following fact: each configuration has *at most* one next configuration. In a deterministic device, given a current configuration, the next configuration of the system is uniquely determined, if any.

– The *non-deterministic mode* verifies the following property: each non halting configuration hast *at least* a next configuration. In a non-deterministic device several next configurations can be reached from a current configuration.

The computation of deterministic devices can be viewed as a tree with only one branch, whereas the computation of a non-deterministic device can be viewed as a tree having many possible branches. The root of the tree corresponds to the beginning of the computation, and every node in the tree corresponds to a point of the computation at which the machine has eventually multiple choices. Each branch of this tree determines one computation of the system.

Next we define what means to accept or reject a string by a deterministic or non-deterministic device (whose answers are only *yes* or *no*).

– A deterministic device M *accepts* (respectively, *rejects*) a string a if the answer of M on input a is *yes* (respectively, *no*).
– A non-deterministic device M *accepts* a string a if there exists a computation of M with input a such that the answer is *yes*.

Let us note the difference between the definition of acceptance by deterministic and non-deterministic devices. An input string a is accepted by a deterministic machine M, if *the* computation of M on input a halts and answers *yes*. A non-deterministic machine M accepts a string a if there exists *some* computation of M on input a answering *yes*; that is, there exists a sequence of non-deterministic choices that answers *yes*. In this case, it is possible that we accept a string but that there exists another computation with the same input that either halts and answers *no*, or does not halt.

Thus, a deterministic device can (mechanically) reject a string, but this is not the case in non-deterministic machines. How can we decide (in a mechanical way) whether there exists a non halting computation?

Non-deterministic Turing machines are like existential quantifiers: they accept an input string if there exists an accepting path in the corresponding computation tree. In some sense, we can affirm that non-deterministic devices do not properly capture the intuitive idea underlying the concept of algorithm, because the result of such a machine on an input (that is, the output of a computation) is not reliable, since the answer of the device is not always the same.

non-determinism can be considered as a generalization of determinism (the computation may branch at each configuration), and it can be viewed as a kind of parallel computation where several "processes" can be run simultaneously.

3 Combinatorial Optimization Problems

Roughly speaking, when we deal with *combinatorial optimization problems* we wish to find the *best* solution (according to a given criterion) among a class of possible (candidate or feasible) solutions. That is, in this kind of problems there can be many possible solutions, each one has associated a value (a positive rational number), and we aim to find a solution with the optimal (minimum or maximum) value.

For example, a *vertex cover* of an undirected graph is a set of vertices such that any edge of the graph has, at least, an endpoint in that set. Then, we may want to find one of the smallest vertex covers among all possible vertex covers in the input graph. This is the combinatorial optimization problem called *Minimum Vertex Cover Problem*. The main ingredients in this problem are the following: (a) the collection of all undirected graphs, (b) the finite set of all vertex covers associated with a given undirected graph, and (c) the cardinality of each vertex cover of a given undirected graph.

We can formalize these ideas in the following definition.

Definition 1. *A combinatorial optimization problem, X, is a tuple (I_X, s_X, f_X) where:*

- *I_X is a language over a finite alphabet.*
- *s_X is a function whose domain is I_X and, for each $a \in I_X$, the set $s_X(a)$ is finite.*
- *f_X is a function (the objective function) that assigns to each instance $a \in I_X$ and each $c_a \in s_X(a)$ a positive rational number $f_X(a, c_a)$.*

The elements of I_X are called *instances* of the problem X. For each instance $a \in I_X$, the elements of the finite set $s_X(a)$ are called *candidate* (or *feasible*) *solutions* associated with the instance a of the problem. For each instance $a \in I_X$ and each $c_a \in s_X(a)$, the positive rational number $f_X(a, c_a)$ is called *solution value* for c_a. The function f_X provides the criterion to determine the *best* solution.

For example, the *Minimum Vertex Cover* problem is a combinatorial optimization problem such that I_X is the set of all undirected graphs, and for each undirected graph G, $s_X(G)$ is the set of all vertex covers of G; that is, each vertex cover of the graph is a candidate solution for the problem. The objective function f_X is defined as follows: for each undirected graph G and each vertex cover C of G, $f_X(G, C)$ is the cardinality of C.

Definition 2. *Let $X = (I_X, s_X, f_X)$ be a combinatorial optimization problem. An optimal solution for an instance $a \in I_X$ is a candidate solution $c \in s_X(a)$ associated with this instance such that,*

- *either for all $c' \in s_X(a)$ we have $f_X(a, c) \leq f_X(a, c')$ (and then we say that c is a minimal solution for a),*
- *either for all $c' \in s_X(a)$ we have $f_X(a, c) \geq f_X(a, c')$ (and then we say that c is a maximal solution for a).*

A *minimization* (respectively, *maximization*) *problem* is a combinatorial optimization problem such that each optimal solution is a minimal (respectively, maximal) solution.

That is, an optimization problem seeks the best of all possible candidate solutions, according to a simple cost criterion given by the objective function. For example, the *Minimum Vertex Cover* problem is a minimization problem because a minimal solution associated with an undirected graph G, provides one of the smallest vertex covers of G.

An *approximation computational device*, $\mathcal{D}$, for a combinatorial optimization problem, X, provides a candidate solution $c \in s_X(a)$ for each instance $a \in I_X$. If the provided solution is always optimal, then $\mathcal{D}$ is called an *optimization computational device* for X.

For instance, an approximation machine for the *Minimum Vertex Cover* problem needs only find some vertex cover associated with each undirected graph, whereas an optimization machine must always find a vertex cover with the least cardinality associated with each undirected graph.

Having in mind that until now polynomial time optimization algorithms have not be found for many presumably intractable problems (it is believed that this kind of solutions can never be found), it is convenient to find an approximation algorithm running in polynomial time and such that, for all problem instances the candidate solution given by the algorithm is *close*, in a sense, to an optimal solution.

4 Decision Problems

An important class of combinatorial optimization problems is the class of decision problems, that is, problems that require either an *yes* or a *no* answer.

Definition 3. *A decision problem, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called instances) and θ_X is a total boolean function (that is, a predicate) over I_X.*

Therefore, a decision problem $X = (I_X, \theta_X)$ can be viewed as a combinatorial optimization problem $X = (I_X, s_X, f_X)$ where for each instance $a \in I_X$ we have the following:

– $s_X(a) = \{\theta_X(a)\}$ (the only possible candidate solution associated with instance a is 0 or 1, depending on the answer of the problem to a).
– $f_X(a, \theta_X(a)) = 1$.

Thus, each decision problem can be considered as a minimization (or maximization) problem.

There exists a natural correspondence between languages and decision problems in the following way. Each language L, over an alphabet Σ, has a decision problem, X_L, associated with it as follows: $I_{X_L} = \Sigma^*$, and $\theta_{X_L} = \{(x, 1) \mid x \in L\} \cup \{(x, 0) \mid x \notin L\}$; reciprocally, given a decision problem $X = (I_X, \theta_X)$, the language L_X over the alphabet of I_X corresponding to it is defined as follows: $L_X = \{a \in I_X \mid \theta_X(a) = 1\}$.

Usually, NP-completeness has been studied in the framework of decision problems. Many abstract problems are not decision problems, but combinatorial optimization problems, in which some value must be optimized (minimized or maximized). In order to apply the theory of NP-completeness to combinatorial optimization problems, we must consider them as decision problems.

We can transform any combinatorial optimization problem into a roughly equivalent decision problem by supplying a target/threshold value for the quantity to be optimized, and asking the question whether this value can be attained. Next we give two examples.

1. The *Minimum Vertex Cover Problem.*
   Optimization version: Given an undirected graph G, find the cardinality of a *minimal* set of a vertex cover of G.
   Decision version: Given an undirected graph G, and *a positive integer* k, determine whether or not G has a vertex cover of size *at most* k.
2. The *Common Algorithmic Problem* [10].
   Optimization version: given a finite set S and a family F of subsets of S, find the cardinality of a *maximal* subset of S which does not include any set belonging to F.
   Decision version: given a finite set S, a family F of subsets of S, and a positive integer k, we are asked whether there is a subset A of S such that the cardinality of A is *at least* k, and which does not include any set belonging to F.

If a combinatorial optimization problem can be *quickly* solved, then its decision version can be quickly solved as well (because we only need to compare the solution value with a threshold value). Similarly, if we can make clear that a decision problem is hard, we also make clear that its associated combinatorial optimization problem is hard.

For example, let A be a polynomial time algorithm for the optimization version of the Minimum Vertex Cover problem. Then we consider the following polynomial time algorithm for the decision version of the Minimum Vertex Cover problem: given an undirected graph G, and a positive integer k, if $k < A(G)$ (here $A(G)$ is the cardinality of a smallest vertex cover of G), then answer *no*; otherwise, the answer is *yes*.

Reciprocally, let B be a polynomial time algorithm for the decision version of the Minimum Vertex Cover problem. Then we consider the following polynomial time algorithm for the optimization version of the Minimum Vertex Cover problem: given an undirected graph G, repeatedly while $k \leq$ number of vertices of G (starting from $k = 0$, and in the next step considering $k + 1$) we execute the algorithm A on input (G, k), until we reach a first *yes* answer, and then the result is k.

5 Solving Decision Problems

Recall that, in a natural way, each decision problems has associated a language over a finite alphabet. Next, we define the solvability of decision problems through the recognition of languages associated with them.

In order to specify the concept of solvability we work with an universal computing model: Turing machines.

Let M be a Turing machine such that the result of any halting computation is *yes* or *no*. Let L be a language over an alphabet Σ.

If M is a *deterministic* device (with Σ as working alphabet), then we say that M *recognizes* or *decides* L whenever, for any string a over Σ, if $a \in L$, then the answer of M on input a is *yes* (that is, M accepts a), and the answer is *no* otherwise (that is, M reject a).

If M is a *non-deterministic* Turing machine, then we say that M *recognizes* or *decides* L if the following is true: for any string a over Σ, $a \in L$ if and only if there exists a computation of M with input a such that the answer is *yes*. That is, an input string a is accepted by M if there is *an* accepting computation of M on input a. But now we do not have a mechanical criterion to reject an input string.

Recall that any deterministic Turing machine with multiple tapes can be simulated by a deterministic Turing machine with one tape with a polynomial loss of efficiency, whereas the simulation of non-determinism through determinism involves an exponential loss of efficiency.

In the context of computation theory, we consider a problem X to be solved when we have a *general* (definite) *method* (described in a model of computation) that works for any instance of the problem. From a practical point of view, such methods only run over a finite set of instances whose sizes depend on the available resources.

We say that a Turing machine M solves a decision problem X if M *recognizes* the language associated with X; that is, for any instance a of the problem: (1) in the deterministic case, the machine (with input a) output *yes* if the answer of the problem is *yes*, and the output is *no* otherwise; (2) in the non-deterministic case, some computation of the machine (with input a) output *yes* if the answer of the problem is *yes*.

Due to the fact that we represent the instances of abstract problems as strings we can consider their size in a natural manner: the size of an instance is the length of the string. Then, how do the resources required to execute a method increase according to the size of the instance? This is a relevant question in computational complexity theory.

6 The P Versus NP Problem

In order to solve an abstract problem by a computational device, problem instances must be represented (encoded) in an adequate way that the device understands.

Given a problem it is possible to use different *reasonable* encoding schemes to represent the instances (we do not attempt to define *reasonable*, however informally we say that *reasonable* means [8] to codify instances in a concise manner, without irrelevant information, and the numbers occurring in them should be represented in binary, or any fixed base other than 1). It is easy to prove that the input sizes that different reasonable encoding schemes determine differ, at most, polynomially from one another.

Recall that complexity classes provide a way to group decision problems of similar computational complexity.

P is the class of all decision problems solvable (or languages recognized) by some deterministic Turing machine in a time bounded by a polynomial on the size of the input. Having in mind that all *reasonable* deterministic computational models are polynomially equivalent (that is, any one of them can simulate another with only a polynomial loss of efficiency), this class is the same for all models of computation that are polynomially equivalent to the deterministic Turing machine with one tape. Moreover, informally speaking, **P** corresponds to the class of problems having a *feasible* algorithm that gives an answer in a *reasonable* time; that is, problems that are realistically solvable on a machine (even for large instances of the problem).

NP is the class of all decision problems solvable in a polynomial time by non-deterministic Turing machines; that is, for every accepted input there exists at least one accepting computation taking an amount of steps bounded by a polynomial on the length of the input. This class is invariant for all reasonable non-deterministic computational models because all of them are polynomially equivalent.

Every deterministic Turing machine can be considered as a non-deterministic one, so we have $\mathbf{P} \subseteq \mathbf{NP}$. In terms of the previously defined classes, the **P** *versus* **NP** problem can be expressed as follows: is it verified the relation $\mathbf{NP} \subseteq \mathbf{P}$? That is, the **P** versus **NP** problem is the problem of determining whether every language recognized by some non-deterministic Turing machine in polynomial time is also can be recognized by some deterministic Turing machine in polynomial time.

The $\mathbf{P} \overset{?}{=} \mathbf{NP}$ question is one of the outstanding open problems in theoretical computer science. The relevance of this question is not only the inherent pleasure of solving a mathematical problem, but in this case an answer to it would provide information of high economical interest. On the one hand, a negative answer to this question would confirm that the majority of current cryptographic systems are secure from a practical point of view. On the other hand, a positive answer would not only show the uncertainty about the secureness of these systems, but also this kind of answer is expected to come together with a general procedure provides a deterministic algorithm solving most of **NP**-complete problem in polynomial time (furthermore, mathematics would be *transformed* because real computers will be able to find a formal proof of any theorem which has a proof of reasonable length).

In the last years several computing models using powerful tools from nature have been developed (because of this, they are known as *bio-inspired* models) and several solutions in polynomial time to problems from the class **NP** have been presented, making use of non-determinism and/or of an exponential amount of space. This is the reason why a practical implementation of such models (in biological, electronic, or other mediums) could provide a significant advance in the resolution of **NP**-complete problems.

7 Strongly NP–Complete Problems

The *Subset Sum* problem is the following: given a finite set A, a weight function, $w : A \to \mathbf{N}$, and a constant $k \in \mathbf{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$.

It is well known that *Subset Sum* can be solved in time $O(n \cdot k)$, using a dynamic programming algorithm. Hence, that algorithm is polynomial in the number of input items n and the magnitude of the items k. But such a algorithm is not a polynomial algorithm because its time bound is not a polynomial function on the size of the input (that is, of the order $\Omega(n \cdot \log k)$). Then we say that such a algorithm is *pseudo-polynomial*, and that the problem can be solved in *pseudo-polynomial time*. Nevertheless if we represent the input in *unary* form then that algorithm becomes a polynomial algorithm.

Definition 4. *An algorithm that solves a problem X will be called a pseudo-polynomial time algorithm for X if its running time would be polynomial if all input numbers associated with each instance were expressed in unary notation.*

The *Knapsack* and *Partition* problems are also **NP**–complete problems that can be solved by a pseudo-polynomial time algorithm.

Often, problems which can be solved in pseudo-polynomial time are also called *weakly* **NP**–*complete* problems. The existence of a pseudo-polynomial time algorithm for a given **NP**–complete problem illustrate that the problem is not so *intractable* after all.

Thus it is important to determine whether a problem is weakly **NP**–complete, or whether it has the following stronger property.

Definition 5. *A problem is said to be* **NP**–*complete in the strong sense if the variant of it in which any instance of size n is restricted to contain integers of size at most p(n), where p is a polynomial, remains* **NP**–*complete.*

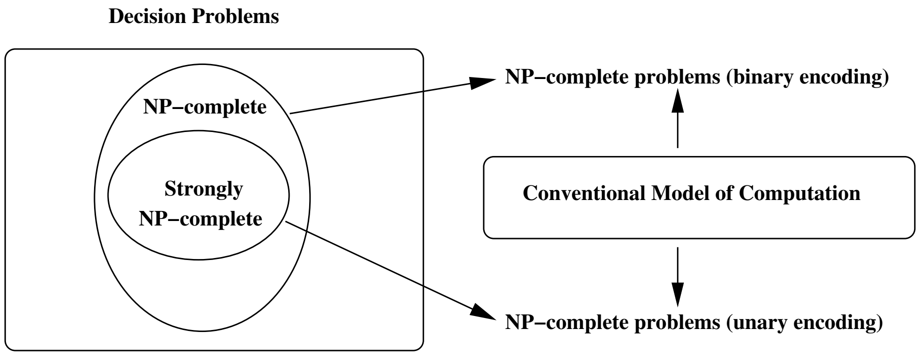
Decision Problems
NP–complete
Strongly NP–complete
NP–complete problems (binary encoding)
Conventional Model of Computation
NP–complete problems (unary encoding)

Fig. 1. NP–Completeness and codification of instances

That is, the strongly **NP**–complete problems remains **NP**–complete even if all numbers in the input are written in unary notation.

For example, the decision version of the *Minimum Vertex Cover* problem is a strongly **NP**–complete problem since the numbers in the input (an undirected graph) are bounded by a polynomial in the number of vertices (input size).

Other strongly **NP**–complete problems are the following: *3–Partition*, *Sat*, *Clique*, *HPP* (Hamiltonian Path Problem), *TSP* (Travelling Salesman Problem), and *Bin Packing.*

What happens if a strongly **NP**–complete problem can be solved by a pseudo-polynomial time algorithm? Let X be such a problem. Then the variant Y of it in which all input numbers of X are written in unary notation is also **NP**–complete. Moreover, if A is a pseudo-polynomial time algorithm solving X, then it is also a polynomial time algorithm that solves Y. Hence, **P**=**NP**.

Theorem 1. *The following propositions are equivalent:*

1. $\mathbf{P} = \mathbf{NP}$.
2. *Every strongly* **NP***–complete problem can be solved by a pseudo-polynomial time algorithm.*
3. *There exists a strongly* **NP***–complete problem that can be solved by a pseudo-polynomial time algorithm.*

Thus, to prove **P**=**NP** suffices to find a strongly **NP**–complete problem solvable in pseudo-polynomial time. Recall that the concept of solvability above mentioned is formally associated with deterministic Turing machines.

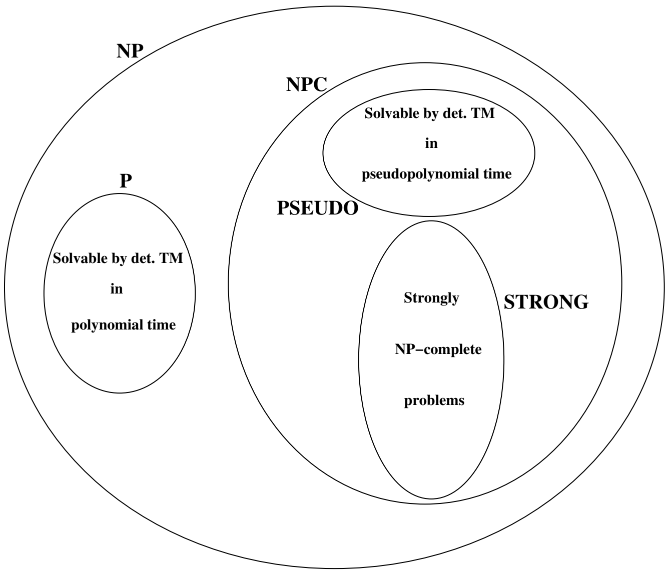

Fig. 2. Kinds of NP–complete problems

However, P systems take multisets as input and handle them through computations. Hence the inputs in P systems are provided in *unary*, so it is necessary to analyze with more details when it is said that a problem is polynomial-time solvable in the framework of membrane computing (particularly, concerning the size of the problem instances).

In this context we can say that polynomial solutions to **NP**–complete problems in the framework of membrane computing, can be considered, in a sense, as *pseudo-polynomial* solutions in the classical sense.

8 Recognizer Membrane Systems

Membrane computing is a recent branch of natural computing initiated in [23]. It has been developed basically from a theoretical point of view.

Membrane systems – usually called P systems – are distributed parallel computing models inspired by the structure and functioning of living cells.

Membrane systems have several *syntactic* ingredients: a *membrane structure* consisting of a hierarchical arrangements of membranes embedded in a *skin* membrane, and delimiting *regions* or compartments where multisets of *objects* and sets (eventually empty) of (evolution) *rules* are placed.

Also, P systems have two main *semantic* ingredients: their inherent *parallelism* and *non-determinism.* The objects inside the membranes can evolve according to given rules in a synchronous (in the sense that a global clock is assumed), parallel, and non-deterministic manner.

Can this parallelism and non-determinism be used to solve hard problems in a feasible time? The answer is affirmative, but we must point out two considerations. On the one hand, we have to deal with the non-determinism in such a way that the solutions obtained from these devices are algorithmic solutions in the classic sense; that is, the answers of the computations of the system must be reliable. On the other hand, the drastic decrease of the execution time from an exponential to a polynomial one is not achieved for free, but by the use of an exponential workspace (in the form of membranes or string–objects), although this space is created in polynomial (often linear) time.

In this paper we use membrane computing as a framework to attack the resolution of decision problems. In order to solve this kind of problems and having in mind the relationship between the solvability of a problem and the recognition of the language associated with it, we consider P systems as *recognizer languages* devices.

Moreover, for technical reasons we only work with devices such that all computations halt, and such that the result (*yes* or *no* answer, because we deal with recognition of strings) is collected in the environment (and in the last step of the computation).

All these restrictions make more difficult the process of designing families of recognizer P systems to solve decision problems.

Definition 6. *A* recognizer P system *is a P system with external output such that:*

1. *The working alphabet contains two distinguished elements* yes *and* no*.*
2. *All computations halt.*
3. *If $\mathcal{C}$ is a computation of the system, then either object* yes *or object* no *(but not both) must have been released into the environment, and only in the last step of the computation.*

In recognizer P systems, we say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if the object *yes* (respectively, *no*) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$. Hence, these devices send to the environment an accepting or rejecting answer, in the end of their computations.

If we want these kind of systems to properly solve decision problems and capture the true algorithmic concept, it is necessary to require a condition of *confluence*; that is, the system must always give the same answer. In order to accept or reject a string it should be enough to read the answer of *any* computation of the system. In this manner, an observer outside the system can identify the exact moment when the system halts, and know the answer.

Since P systems work in a non-deterministic manner, we need to adapt the usual definition of acceptance in non-deterministic Turing machines.

9 Soundness and Completeness

In order to assure that a family of recognizer P systems solves a decision problem, two main properties must to be proved: for each instance of the problem,

(a) if *there exists an* accepting computation of the membrane system processing it, answering *yes*, then the problem also answer *yes* for that instance (*soundness*);
(b) if the problem answers *yes*, then *any* computation of the system processing that instance, answer *yes* (*completeness*).

If we demand that the family of membrane systems is sound and complete, then it satisfies a condition of *confluence*: every computation of a system from the family has the same output.

Next, we formalize these ideas in the following definition.

Definition 7. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$ be a family of recognizer membrane systems* without input*.*

- *We say that the family $\mathbf{\Pi}$ is sound with regard to X if the following is true: for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(w)$, then $\theta_X(w) = 1$.*
- *We say that the family $\mathbf{\Pi}$ is complete with regard to X if the following is true: for each instance of the problem $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(w)$ is an accepting computation.*

The soundness property means that if we obtain an *acceptance response* of the system (associated with an instance) through some computation, then the answer of the problem (for that instance) is *yes*. The completeness property means that if we obtain an *affirmative* response to the problem, then any computation of the system must be an accepting one.

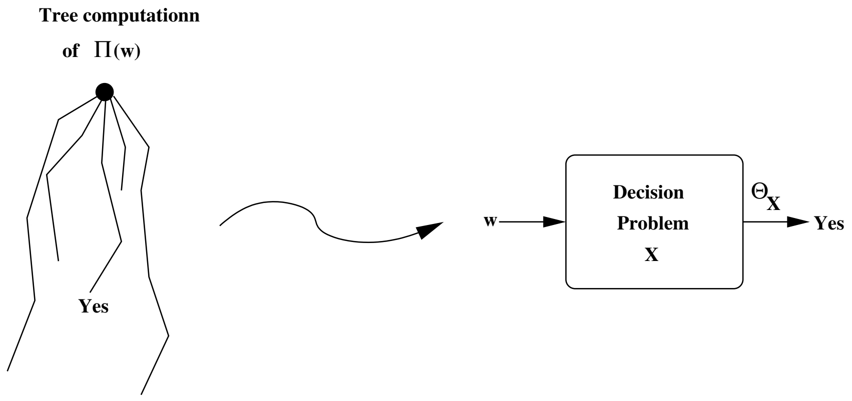

Fig. 3. Soundness of a family of P systems without input

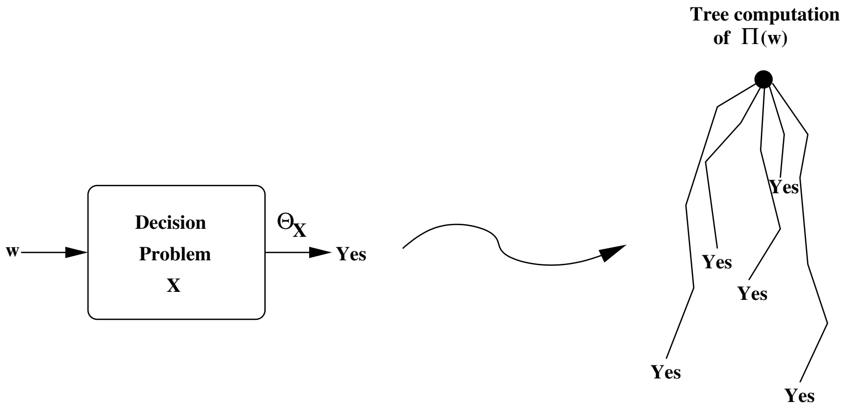

Fig. 4. Completeness of a family of P systems without input

These concepts can be extended to families of recognizer P systems *with input membrane* in a natural way, but in this case a P system belonging to the family can process several instances of the problem provided that an appropriate input, depending on the instance, is supplied to the system.

Definition 8. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer membrane systems* with input. *A* polynomial encoding *of X in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial time computable functions over I_X such that for each instance $w \in I_X$, $s(w)$ is a natural number and $cod(w)$ is an input multiset of the system $\Pi(s(w))$.*

Definition 9. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer membrane systems* with input. *Let (cod, s) be a polynomial encoding of X in $\mathbf{\Pi}$.*

– *We say that the family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s) *if the following is true: for each instance of the problem* $w \in I_X$*, if there exists an accepting computation of* $\Pi(s(w))$ *with input* $cod(w)$*, then* $\theta_X(w) = 1$*.*
– *We say that the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s) *if the following is true: for each instance of the problem* $u \in I_X$*, if* $\theta_X(u) = 1$ *then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting computation.*

The soundness property means that if given an instance we obtain an *acceptance response* of the system associated with it (and individualized by the appropriate input multiset) through some computation, then the answer of the problem (for that instance) is *yes*. The completeness property means that if we obtain an *affirmative* response to the problem, then any computation of the system associated with it (and individualized by the appropriate input multiset) must be an accepting one.

10 Polynomial Complexity Classes in Membrane Systems

Next, we consider different complexity classes in the framework of recognizer membrane systems.

10.1 Recognizer Membrane Systems Without Input

The first results about *solvability* of **NP**–complete problems in polynomial time (even linear) by membrane systems were given by Gh. Păun [25], C. Zandron et al. [43], S.N. Krishna et al. [12], and A. Obtulowicz [16] in the framework of P systems that lack an input membrane. Thus, the constructive proofs of such results design *one* system for *each* instance of the problem.

In this context, next we define polynomial complexity classes in recognizer membrane systems without input. In order to solve a decision problem we need, then, to associate with each instance of the problem a system which decides the instance. We impose these systems to be *confluent* in the following sense: an instance of the problem will have a positive answer if and only if *every* (or, equivalently, there exists a) computation of the corresponding system is an accepting computation.

We also demand that *every* computation is bounded, in execution time, by a polynomial function. This is because we do not only want to obtain the same answer, independently of the chosen computation, but that all the computations consume, at most, the same amount of resources (in time).

Definition 10. *Let* $\mathcal{R}$ *be a class of recognizer P systems without input membrane. A decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$*, of P systems of type* $\mathcal{R}$*, and we denote it by* $X \in \mathbf{PMC}^*_{\mathcal{R}}$*, if the following is true:*

– *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *from the instance* $w \in I_X$*.*

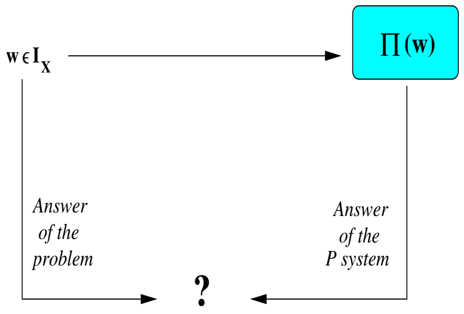

Fig. 5. Complexity class for membrane systems without input

– *The family* **Π** *is polynomially bounded; that is, there exists a polynomial function* $p(n)$ *such that for each* $w \in I_X$*, all computations of* $\Pi(w)$ *halt in at most* $p(|w|)$ *steps.*
– *The family* **Π** *is sound and complete with regard to* X*.*

Note that in this complexity class we consider two different tasks: the first one is the construction of the family, which we require to be done in polynomial time (sequential time by deterministic Turing machines). The second one is the execution of the systems of the family, in which we imposed that the total number of steps performed by their computations are bounded by the function g (parallel time by non-deterministic membrane systems).

As a direct consequence of working with recognizer membrane systems is the fact that these complexity classes are closed under complement.

Moreover, the complexity classes are closed under polynomial time reduction, in the classical sense. Recall that if $X = (I_X, \theta_X)$ and $Y = (I_Y, \theta_Y)$ are decision problems, then we say that X is reducible to Y in polynomial time if there exists a polynomial time function f from I_X to I_Y verifying the following condition: for each $w \in I_X$ we have $\theta_X(w) = 1$ if and only if $\theta_Y(f(w) = 1$.

Proposition 1. *Let* $\mathcal{R}$ *be a class of recognizer P systems without input membrane. Let* X *and* Y *be two decision problems such that* X *is reducible to* Y *in polynomial time. If* $Y \in \mathbf{PMC}^*_{\mathcal{R}}$*, then* $X \in \mathbf{PMC}^*_{\mathcal{R}}$*.*

The *Hamiltonian Path Problem* can be solved in *quadratic time* by a family $\mathcal{R}$ of recognizer P systems without input in an uniform way (see [26]). Then $\mathbf{NP} \subseteq \mathbf{PMC}^*_{\mathcal{R}}$.

10.2 Recognizer Membrane Systems with Input

A computation of a Turing machine starts when the machine is in the initial state and we "write" a string in the input tape of the machine. Then, the machine starts to compute according to the transition function. In the definitions of basic P systems that have been initially considered, there is no membrane in which we can "introduce" input objects before allowing the system to begin to work. However, it is easy to consider input membranes in this kind of devices.

In this section we deal with recognizer membrane systems *with an input membrane* and we propose to solve hard problems in an *uniform* way in the following sense: all instances of a decision problem that have the same *size* (according to a prefixed polynomial time computable criterion) are processed by the same system, to which an appropriate input, that depends on the specific instance, is supplied.

Now, we formalize these ideas in the following definition.

Definition 11. *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is solvable in polynomial time by a family of recognizer membrane systems* with input $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$*, and we denote it by $X \in \mathbf{PMC}_{\mathcal{R}}$, if the following is true:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine that constructs in polynomial time the system $\Pi(n)$ from $n \in \mathbf{N}$.*
- *There exists a polynomial encoding (cod, s) of X in $\mathbf{\Pi}$ such that:*
  - *The family $\mathbf{\Pi}$ is polynomially bounded with regard (X, cod, s); that is, there exists a polynomial function $p(n)$ such that for each $w \in I_X$ every computation of the system $\Pi(s(w))$ with input $cod(w)$ is halting and, moreover, it performs at most $p(|w|)$ steps.*
  - *The family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s).*

Note that in the above definition and in order to decide about an instance, w, of a decision problem, first of all we need to compute the natural number $s(w)$, obtain the input multiset $cod(w)$, and construct the system $\Pi(s(w))$. This is properly a *pre-computation stage*, running in polynomial time expressed by a number of *sequential steps* in the framework of the Turing machines. After that, we execute the system $\Pi(s(w))$ with input $cod(w)$. This is properly the *computation stage*, also running in polynomial time, but now it is described by a number of *parallel steps*, in the framework of membrane computing.

As mentioned above, these complexity classes are closed under complement.

Moreover, these complexity classes are closed under polynomial time reduction, in the classical sense.

Proposition 2. *Let $\mathcal{R}$ be a class of recognizer P systems with input membrane. Let X and Y be two decision problems such that X is reducible to Y in polynomial time. If $Y \in \mathbf{PMC}_{\mathcal{R}}$, then $X \in \mathbf{PMC}_{\mathcal{R}}$.*

The *Satisfiability Problem* can be solved in linear time by a family $\mathcal{R}$ of recognizer P systems with input (see [36]). Then $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{R}}$.

11 (Non-deterministic) Polynomial Complexity Classes in Membrane Systems

According to the usual manner of considering acceptance by non-deterministic Turing machines, we can consider non-deterministic complexity classes in P systems without requiring them to be confluent, that is, *characterizing* the acceptance of an input string by the existence of an accepting computation.

Definition 12. *Let $\mathcal{R}$ be a class of recognizer P systems without input membrane. A decision problem $X = (I_X, \theta_X)$ is non-deterministically solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$, of P systems of type $\mathcal{R}$, and we denote it by $X \in \mathbf{NPMC}^*_{\mathcal{R}}$, if the following is true:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *The family $\mathbf{\Pi}$ is polynomially bounded.*
- *The family $\mathbf{\Pi}$ is sound and complete with regard to X, in the following sense: for each instance $w \in I_X$ of the problem, $\theta_X(w) = 1$ if and only if there exists an accepting computation of $\Pi(w)$.*

Note that in this definition, in contrast to the corresponding definition for deterministic complexity classes, we only demand that for each instance w with affirmative answer there exists at least *one* accepting computation of the system $\Pi(w)$, instead of demanding *every* computation of the system to be an accepting one.

Again, this class is closed under polynomial time reduction, but notice that it does not have to be closed under complement.

Let us denote by $\mathcal{T}$ the class of recognizer *transition* P systems (see [23]). In [36] we construct a family of recognizer transition P systems solving *HPP* (in the directed version with two distinguished nodes) in *linear time*, in a non-deterministic manner. That is, we have the following:

Proposition 3. *$HPP \in \mathbf{NPMC}^*_{\mathcal{T}}$, and $\mathbf{NP} \subseteq \mathbf{NPMC}^*_{\mathcal{T}}$.*

In a similar way we can define non-deterministic complexity classes for recognizer membrane systems with input.

Definition 13. *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is non-deterministically solvable in polynomial time by a family of recognizer membrane systems* with input *$\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$, and we denote it by $X \in \mathbf{NPMC}_{\mathcal{R}}$, if the following is true:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *There exists a polynomial encoding (cod, s) of X in $\mathbf{\Pi}$ such that:*
  - *The family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s).*
  - *The family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s), but now in the following sense: for each instance $w \in I_X$ of the problem, $\theta_X(w) = 1$ if and only if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$.*

This class is closed under polynomial time reduction, but it does not have to be closed under complement.

In [36] we construct a family of recognizer transition P systems solving *SAT* in *constant time*, in a non-deterministic manner. That is, we have the following:

Proposition 4. *$SAT \in \mathbf{NPMC}_{\mathcal{T}}$, and $\mathbf{NP} \subseteq \mathbf{NPMC}_{\mathcal{T}}$.*

12 Characterizing the $\mathbf{P} \neq \mathbf{NP}$ Relation Through P Systems

In this section we show how it is possible to attack the **P** versus **NP** problem within the framework of membrane computing.

We consider deterministic Turing machines as language decision devices. That is, the machines halt over any string on the input alphabet, with the halting state being equal to the accepting state, in the case that the string belongs to the decided language, and with that state equal to the rejecting state, in the case that the string does not belong to that language.

It is possible to associate with a Turing machine a decision problem, which will permit us to say when such a machine is simulated by a family of P systems.

Definition 14. *Let TM be a Turing machine with input alphabet Σ_{TM}. The* decision problem associated with TM *is the problem $X_{TM} = (I, \theta)$, where $I = \Sigma_{TM}^*$, and for every $w \in \Sigma_{TM}^*$, $\theta(w) = 1$ if and only if TM accepts w.*

Obviously, the decision problem X_{TM} is solvable by the Turing machine TM.

Definition 15. *We say that a Turing machine TM is simulated in polynomial time by a family of recognizer P systems if $X_{TM} \in \mathbf{PMC}_{\mathcal{R}}$.*

In P systems, evolution rules, communication rules and rules involving dissolution are called *basic rules.* That is, by applying this kind of rules the size of the structure of membranes does not increase. Hence, it is not possible to construct an exponential working space in polynomial time using only basic rules in a P system.

In Chapter 9 of [40], and following the ideas from [41], we state that every deterministic Turing machine can be simulated in polynomial time by a family of systems of the class $\mathcal{R}$.

Proposition 5. *Let TM be a deterministic Turing machine working in polynomial time. Then TM can be simulated in polynomial time by a family of recognizer P systems using only basic rules.*

In [38], we proved the following result that can be considered as a reciprocal of the above proposition.

Proposition 6. *If a decision problem is solvable in polynomial time by a family of recognizer P systems (using only basic rules), then there exists a Turing machine solving it in polynomial time.*

From the above propositions, we establish characterizations of the $\mathbf{P} \neq \mathbf{NP}$ relation by means of the polynomial time unsolvability of **NP**–complete problems by families of recognizer P systems.

Theorem 2. *The following conditions are equivalent:*

(1) $\mathbf{P} \neq \mathbf{NP}$.
(2) There exists an **NP***–complete decision problem unsolvable in polynomial time by a family of of recognizer P systems using only basic rules.*
(3) Each **NP***–complete decision problem is unsolvable in polynomial time by a family of of recognizer P systems using only basic rules.*

From the constructive proof given in [38], we can deduce the following nice result characterizing the class **P**.

Proposition 7. *Let* $\mathcal{T}$ *the class of recognizer transition P systems. Then* $\mathbf{P} = \mathbf{PMC}_{\mathcal{T}}$.

13 P Systems with Active Membranes

P systems with membrane division were introduced in [25], and in this variant the number of membranes can increase exponentially in polynomial time.

Next, we define P systems with active membranes using 2-division for elementary membranes, with polarizations, but without cooperation and without priorities (and without permitting the change of membrane labels by means of any rule).

Definition 16. *A P system with active membranes using 2-division for elementary membranes is a tuple* $\Pi = (\Sigma, H, \mu, \mathcal{M}_1, \dots, \mathcal{M}_m, R)$, *where:*

1. $m \geq 1$, *is the initial degree of the system;*
2. Σ *is an alphabet of symbol-objects;*
3. H *is a finite set of labels for membranes;*
4. μ *is a membrane structure, of* m *membranes, labelled (not necessarily in a one-to-one manner) with elements of* H*;*
5. $\mathcal{M}_1, \dots, \mathcal{M}_m$ *are strings over* Σ, *describing the initial multisets of objects placed in the* m *regions of* μ*;*
6. *R is a finite set of evolution rules, of the following forms:*
   (a) $[\, a \to \omega \,]_h^{\alpha}$ *for* $h \in H$,$\alpha \in \{+, -, 0\}$, $a \in \Sigma$, $\omega \in \Sigma^*$*: This is an object evolution rule, associated with a membrane labelled with h and depending on the polarity of that membrane, but not directly involving the membrane.*
   (b) $a\, [\ \ ]_h^{\alpha_1} \to [\, b \,]_h^{\alpha_2}$ *for* $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$*: An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object, and, simultaneously, the polarity of the membrane can be changed.*
   (c) $[\, a \,]_h^{\alpha_1} \to b\, [\ \ ]_h^{\alpha_2}$ *for* $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$*: An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object, and, simultaneously, the polarity of the membrane can be changed.*
   (d) $[\, a \,]_h^{\alpha} \to b$ *for* $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in \Sigma$*: A membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.*

(e) $[\,a\,]_h^{\alpha_1} \rightarrow [\,b\,]_h^{\alpha_2}\ [\,c\,]_h^{\alpha_3}$ *for* $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in \Sigma$*: An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.*

These rules are applied according to the following principles:

– All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must evolve.
– If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.
– If at the same time a membrane labelled by 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. Of course, this process takes only one step.
– The rules associated with membranes labelled by h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)-(e).

Note that these P systems have some important properties:

– They use three electrical charges.
– The polarization of a membrane can be modified by the application of a rule.
– The label of a membrane cannot be modified by the application of a rule.
– They do not use cooperation neither priorities.

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division for elementary membranes.
In this class of recognizer membrane systems:

– Some weakly **NP**–complete problems are solvable in polynomial time: for example, *Knapsack* ([31]), *Subset Sum* ([30]), *Partition* ([9]) $\in \mathbf{PMC}_{\mathcal{AM}}$.
– Some strongly **NP**–complete problems are solvable in polynomial time: for example, the following problems *SAT* ([36]), *Clique* ([3]), *Bin Packing* ([33]), *CAP* ([34]) belong to the complexity classes $\mathbf{PMC}_{\mathcal{AM}}$.

Recall that polynomial time solutions to strongly **NP**–complete problems by recognizer membrane systems, can be considered as pseudo-polynomial solutions in the classical sense.

Having in mind that the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under complement and polynomial time reductions we have the following result.

Proposition 8. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$*, and* $\mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

P. Sosik in [42] provides a semi–uniform efficient solution to *QBF* (satisfiability of quantified propositional formulas), a well known **PSPACE**–complete problem, in the framework of P systems with active membranes but using 2–division for non–elementary membranes. Hence we have the following result.

Proposition 9. *Let $\mathcal{AM}^*$ be the class of recognizer P systems with active membranes using 2-division for non–elementary membranes. Then,* $\mathbf{PSPACE} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^*}$.

This result shows that the complexity classes $\mathbf{PMC}_{\mathcal{AM}}$ and $\mathbf{PMC}^*_{\mathcal{AM}^*}$ are not precise enough to describe classical complexity classes below **NP**. Therefore, it is challenging to investigate weaker variants of P systems with active membranes able to characterize classical complexity classes (especially, the classes **NP** and **PSPACE**).

In [4], universality has been achieved removing the polarization of membranes from P systems with active membranes but allowing the change of membrane labels by means of communication rules and membrane division rules. Moreover, in this framework it is possible to solve **NP**–complete problems (e.g., the *SAT* problem) in linear time.

Several efficient solutions to **NP**–complete problems have been obtained within the following variants of membrane systems with active membranes:

– P systems using 2–division for elementary membranes, without cooperation, without priorities, without label changing, but using only two electrical charges ([1], [39]).
– P systems using 2–division for elementary membranes, without cooperation, without priorities, without changing of membrane labels, without polarizations, but using bi–stable catalysts ([32]).
– P systems without polarizations, without cooperation, without priorities, without label changing, without division, but using three types of membrane rules: separation, merging, and release ([19]).
– P systems with separation rules instead of division rules, in two different cases: in the first, using polarizations and separation rules; and in the second one, without polarizations, without change of membrane labels but using separation rules with change of membrane labels ([20]).

It is easy to obtain solutions to **NP**–complete problems through P systems with active membranes using 2-division for elementary membranes, without polarizations, without priorities, without label changing possibilities, but using cooperation (or trading cooperation by priority).

But, what happens if we consider only recognizer P systems with active membranes using 2-division for elementary membranes, without polarizations, without cooperation, without priority, and without changing of membrane labels? Let $\mathcal{AM}^0$ be the class of recognizer P systems of this kind.

What is exactly the class of decision problems solvable in polynomial time by families of systems belonging to $\mathcal{AM}^0$? Is the relation $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0}$ true?

Another interesting questions about the relationship between classical and cellular complexity classes are the following ones:

Question 1: Is there a classical complexity class $\mathcal{C}$, such that $\mathcal{C} = \mathbf{PMC}_{\mathcal{AM}}$?
Question 2: Given a classical complexity class $\mathcal{C}$, determine a (minimal in a descriptive sense) class of recognizer P systems $\mathcal{F}$ such that $\mathcal{C} = \mathbf{PMC}_{\mathcal{F}}$?

14 Conclusions

In this paper, some polynomial complexity classes in recognizer membrane systems, without or with input, and capturing the "classical" deterministic and non-deterministic modes of computation, have been introduced.

The complexity classes corresponding to membrane systems without input (respectively, with input) provide the general framework to design solutions to decision problems in a *semi–uniform* (respectively, *uniform*) way.

In this context we have proven that membrane computing offers a new way to attack the **P** versus **NP** problem.

The convenience of characterizing classical complexity classes through these new classes is an interesting topic in order to study the minimal ingredients required, from membrane systems point of view, to obtain certain *computational efficiency.*

Acknowledgement

The author wish to acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

1. A. Alhazov, R. Freund, Gh. Păun, P systems with active membranes and two polarizations. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, 2004, 20–35.
2. A. Alhazov, T.-O. Ishdorj, Membrane operations in P systems with active membranes. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, 2004, 37–52.
3. A. Alhazov, C. Martín–Vide, L. Pan, Solving graph problems by P systems with restricted elementary active membranes. *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), Lecture Notes in Computer Science, 2950 (2004), 1–22.
4. A. Alhazov, L. Pan, Gh. Păun, Trading polarizations for labels in P systems with active membranes. *Acta Informatica*, to appear.
5. J. Castellanos, Gh. Păun, A. Rodríguez–Patón, P systems with worm–objects. *IEEE 7th International Conference on String Processing and Information Retrieval, SPIRE 2000*, La Coruña, Spain, 64–74.
6. A. Cordón–Franco, M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, F. Sancho–Caparrini, Implementing in Prolog an effective cellular solution for the Knapsack problem. *Membrane Computing* (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Lecture Notes in Computer Science, 2933 (2004), 140-152.
7. E. Czeiler, Self–activating P systems. *Membrane Computing* (Gh. Păun, G. Rozenberg, A, Salomaa, C. Zandron, eds.), Lecture Notes in Computer Science, 2597 (2003), 234–246.
8. M.R. Garey, D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

9. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, A fast P system for finding a balanced 2-partition. *Soft Computing*, in press.
10. T. Head, M. Yamamura, S. Gal, Aqueous computing: writing on molecules. *Proceedings of the Congress on Evolutionary Computation 1999*, IEEE Service Center, Piscataway, NJ, 1999, 1006–1010.
11. M. Ito, C. Martín–Vide, Gh. Păun, Characterization of Parikh sets of ET0L languages in terms of P systems. In *Words, Semigroups, and Transducers* (M. Ito, Gh. Păun, S. Yu, eds.), World Scientific, Singapore, 2001, 239–254.
12. S.N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP–complete problems. *Romanian Journal of Information Science and Technology*, 2, 4 (1999), 357–367.
13. S.N. Krishna, R. Rama, P systems with replicated rewriting. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 345–350.
14. S.N. Krishna, R. Rama, Breaking DES using P systems. *Theoretical Computer Science*, 299, 1-3 (2003), 495–508.
15. M. Madhu, K. Kristhivasan, P systems with membrane creation: Universality and efficiency. *Proceedings Third International Conference on Universal, Machines and Computations*, Chisinau, Moldova, 2001 (M. Margenstern, Y. Rogozhin, eds.), Lecture Notes in Computer Science, 2055 (2001), 276–287.
16. A. Obtulowicz, Deterministic P systems for solving SAT problem. *Romanian Journal of Information Science and Technology*, 4, 1–2 (2001), 551–558.
17. A. Obtulowicz, On P systems with active membranes: Solving the Integer Factorization problem in a polynomial time. In *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Lecture Notes in Computer Science, 2235 (2001), 267–285.
18. A. Obtulowicz, Note on some recursively family of P systems with active membranes. Submitted, 2004.
19. L. Pan, A. Alhazov, T.-O. Ishdorj, Further remarks on P systems with active membranes, separation, merging, and release rules. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, 2004, 316–324.
20. L. Pan, T.-O. Ishdorj, P systems with active membranes and separation rules. *Journal of Universal Computer Science*, 10, 5 (2004), 630–649.
21. L. Pan, C. Martín–Vide, C. Solving multiset 0–1 knapsack problem by P systems with input and active membranes. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, 2004, 342–353.
22. A. Păun, On P systems with membrane division. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 187–201.
23. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
24. Gh. Păun, Computing with membranes: Attacking **NP**–complete problems. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), 2000, 94–115.
25. Gh. Păun, P systems with active membranes: Attacking **NP**–complete problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.
26. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.

27. Gh. Păun, M.J. Pérez–Jiménez, A. Riscos–Núñez, P systems with tables of rules. *Theory is Forever. Essays Dedicated to Arto Salomaa on the Ocassion of His 70th Birthday* (J. Karhumaki, H. Maurer, Gh. Păun, G. Rozenberg, eds.), Lecture Notes in Computer Science, 3113 (2004), 235-249.
28. Gh. Păun, G. Rozenberg, A guide to membrane computing. *Theoretical Computer Science*, 287 (2002), 73–100.
29. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori, On the power of membrane division in P systems. *Theoretical Computer Science*, 324, 1 (2004), 61–85.
30. M.J. Pérez–Jiménez, A. Riscos–Núñez, Solving the Subset-Sum problem by P systems with active membranes. *New Generation Computing*, in press.
31. M.J. Pérez–Jiménez, A. Riscos–Núñez, A linear time solution to the Knapsack problem using active membranes. *Membrane Computing* (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.). Lecture Notes in Computer Science, 2933 (2004), 250–268.
32. M.J. Pérez–Jiménez, F.J. Romero-Campero, Trading polarizations for bi-stable catalysts in P systems with active membranes. In this volume.
33. M.J. Pérez–Jiménez, F.J. Romero-Campero, An efficient family of P systems for packing items into bins. *Journal of Universal Computer Science*, 10, 5 (2004), 650–670.
34. M.J. Pérez–Jiménez, F.J. Romero-Campero, Attacking the Common Algorithmic problem by recognizer P systems. *Pre-proceedings of the Machines, Computations and Universality, MCU'2004 (abstracts)*, September 21-26, 2004, Sankt Petesburg, p. 27.
35. M.J. Pérez–Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, *Teoría de la Complejidad en Modelos de Computación con Membranas*, Ed. Kronos, Sevilla, 2002.
36. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini, Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2, 3 (2003), 265–285.
37. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini, Solving VALIDITY problem by active membranes with input. *Proceedings of the Brainstorming Week on Membrane Computing* (M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.), Report GRLMC 26/03, 2003, 279–290.
38. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini, The P versus NP problem through cellular computing with membranes. *Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Ocassion of His 70th Birthday* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), Lecture Notes in Computer Science, 2950 (2004), 338–352.
39. A. Riscos-Núñez, *Programación celular: Resolución eficiente de problemas numéricos* **NP**–*completos.* PhD. Thesis, University of Seville, Spain, 2004.
40. A. Romero-Jiménez, *Complexity and Universality in Cellular Computing Models*, PhD. Thesis, University of Seville, Spain, 2003.
41. A. Romero-Jiménez, M.J. Pérez–Jiménez, Simulating Turing machines by P systems with external output. *Fundamenta Informaticae*, 49, 1-3 (2002), 273–287.
42. P. Sosik, The computational power of cell division. *Natural Computing*, 2, 3 (2003), 287–298.
43. C. Zandron, C. Ferreti, G. Mauri, Solving NP-complete problems using P systems with active membranes. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M.J. Dinneen, eds.), Springer-Verlag, Berlin, 2000, 289–301.
44. C. Zandron, G. Mauri, C. Ferreti, Universality and normal forms on membrane systems. *Proceedings International Workshop on Grammar Systems, 2000* (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, July 2000, 61–74.