

# Conservative Computations in Energy-based P Systems

Alberto LEPORATI, Claudio ZANDRON, Giancarlo MAURI

Dipartimento di Informatica, Sistemistica e Comunicazione  
Università degli Studi di Milano – Bicocca  
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy  
E-mail: {leporati, zandron, mauri}@disco.unimib.it

## Abstract

Starting from conservative energy-based P systems, we define conservative computations as computations in which the amount of energy entering the system is the same as the amount of energy leaving it.

We show that conservative computations naturally induce an NP-hard optimization problem, that we have named MIN STORAGE, and a corresponding NP-complete decision problem, CONSCOMP. Finally, we propose a polynomial time 2-approximation algorithm for MIN STORAGE.

## 1 Preliminaries

P systems (also called *membrane systems*) have been introduced in [14] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. The rules are applied in a nondeterministic and maximally parallel way: all the objects that may evolve are forced to evolve. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane* or emitted from the skin of the system.

In what follows we assume that the reader is already familiar with the basic notions and the terminology underlying P systems. For details, and a systematic introduction on the subject, see the monograph [16]. The latest information about P systems can be found in [20].

Energy-based P systems have been introduced in [12] as P systems in which the amount of energy manipulated and/or consumed during computations is taken into account. A given amount of energy is associated to each object of the system. Moreover, instances of a special symbol  $e$  are used to denote free energy units occurring into the regions of the system. These energy units can be used to transform objects, using appropriate rules. The rules are defined according to conservativeness

considerations. An object can always be transformed into another object having the same energy. On the other hand, if the transformed object has a different energy then the required (resp., exceeding) free energy units are taken from (resp., released to) the region where the rule is applied. For each rule of the system it is required that the amount of energy occurring on the left side be the same as the amount of energy occurring on the right side. We assume that the application of rules consumes no energy. This means, in particular, that objects can be moved (without altering them) between the regions of the system without energy consumption. A special case of energy-based P systems are *conservative* P systems, where the amount of energy entering the system with the input values is completely returned with the output values at the end of the computation, and no free energy units enter or leave the system during the computation.

Formally, an energy-based P system (of degree  $m \geq 1$ ) is a construct

$$\Pi = (A, \varepsilon, \mu, e, w_1, \dots, w_m, R_1, \dots, R_m, i_{\text{in}}, i_{\text{out}}),$$

where:

- $A$  is an alphabet; its elements are called *objects*;
- $\varepsilon : A \rightarrow \mathbb{R}^+$  is a linear mapping that associates to each object  $a \in A$  the real value  $\varepsilon(a)$  (also denoted by  $\varepsilon_a$ ), which can be thought of as the “energy value of  $a$ ”. Precisely, if  $A = \{a_1, a_2, \dots, a_d\}$  then for all  $i \in \{1, 2, \dots, d\}$  it holds  $\varepsilon(a_i) = \varepsilon(a_1) + (i - 1)\delta$  for an appropriate real value  $\delta > 0$ . Hence, the energy values considered in the system are equispaced by the quantity  $\delta$ . Through an appropriate rescaling, we can always assume that all energy values are positive integer values, and that  $\delta = 1$ ;
- $\mu$  is a hierarchical membrane structure consisting of  $m$  membranes. For the sake of clarity, we will label membranes with mnemonic identifiers which recall their function;
- $e \notin A$  is a special symbol that denotes one *free energy* unit, that is, one unit of energy which is not embedded into any object;
- $w_i$ , for all  $i \in \{1, \dots, m\}$ , specify the multisets (over  $A \cup \{e\}$ ) of objects initially present in region  $i$ ;
- $R_i$ , for all  $i \in \{1, \dots, m\}$ , is a finite set of evolution rules over  $A$  associated with region  $i$ . Only rules of the following types are allowed:

$$ae^k \rightarrow (b, p) \quad , \quad a \rightarrow (b, p)e^k \quad , \quad e \rightarrow (e, p)$$

where  $a, b \in A$ ,  $p \in \{\text{here}, \text{in}(\text{name}), \text{out}\}$  and  $k$  is a non negative integer;

- $i_{\text{in}}$  is an integer between 1 and  $m$  and specifies the input membrane of  $\Pi$ ;
- $i_{\text{out}}$  is an integer between 0 and  $m$  and specifies the output membrane of  $\Pi$ . If  $i_{\text{out}} = 0$ , then the environment is used for the output, that is, the output value is the multiset of objects (over  $A$ ) emitted from the skin.

A special attention is due to the definition of rules. The meaning of rule  $ae^k \rightarrow (b, p)$ , with  $a, b \in A$ ,  $p \in \{\text{here}, \text{in}(\text{name}), \text{out}\}$ , and  $k$  a positive integer number, is the following: the object  $a$ , in presence of  $k$  free energy units, is allowed to be transformed into object  $b$ . If  $p = \text{here}$ , then the new object  $b$  remains in the same region; if  $p = \text{out}$ , then  $b$  leaves the current membrane. Finally, if  $p = \text{in}(\text{name})$ , then  $b$  enters into the membrane labelled with  $\text{name}$ , which must be a child of the current membrane in the membrane hierarchy.

The meaning of rule  $a \rightarrow (b, p)e^k$ , when  $k$  is a positive integer number, is analogous. The object  $a$  is allowed to be transformed into object  $b$  by releasing  $k$  units of free energy. As above, the new object  $b$  may optionally move one level up or down into the membrane hierarchy. The  $k$  free energy units can now be used by another rule to produce “more energetic” objects from “less energetic” ones.

When  $k = 0$  the rule  $ae^k \rightarrow (b, p)$  is written as  $a \rightarrow (a, p)$ , and simply moves (if  $p \neq \text{here}$ ) the object  $a$  upward or downward into the membrane hierarchy, without acquiring nor releasing any free energy unit. Analogously, rules  $e \rightarrow (e, p)$  simply move (if  $p \neq \text{here}$ ) one unit of free energy upward or downward into the membrane hierarchy.

A further constraint for the definition of rules is that each rule must be “conservative”, in the sense that the amount of energy occurring on the left side of the rule must be the same as the amount of energy which occurs on the right side.

With a little abuse of notation, when the pair  $(x, p)$ , with  $x \in A \cup \{e\}$  and  $p \in \{\text{here}, \text{in}(\text{name}), \text{out}\}$ , appears into a rule we will write  $x_p$ . Also, if  $p = \text{in}(\text{name})$  and no confusion arises we will usually write just the name of the membrane. Moreover, instead of writing  $e^k$  we will sometimes explicitly write  $k$  instances of  $e$ . It is also understood that the position of  $e^k$  (that is, on the left or on the right of the symbol of  $A$ ) either into the left or into the right side of a rule is uninfluent. Finally, when the position  $p$  of an object which occurs in the right side of a rule is “here” we will omit to write it.

**Example 1.1.** *Let us assume  $A = \{a, b, c, d\}$ , where the objects have energy values  $\varepsilon_a = 1$ ,  $\varepsilon_b = 2$ ,  $\varepsilon_c = 3$  and  $\varepsilon_d = 4$ . Then the rule  $be^2 \rightarrow (d, \text{out})$  (also written as  $bee \rightarrow d_{\text{out}}$ ) transforms an instance of the object  $b$  into an instance of the object  $d$ , provided that two free energy units are available, and makes the new object  $d$  leave the current membrane.*

*On the other hand, the rule  $c \rightarrow (a, \text{here})e^2$  (also written as  $c \rightarrow aee$ ) transforms an instance of the object  $c$  into an instance of the object  $a$  and releases two free energy units into the region in which the rule is defined.*

Let us note also that in case of necessity (for example, during proofs) we can safely assume that for each rule at most one instance of  $e$  cooperates with a symbol of the alphabet. In fact, any rule of the kind  $a \rightarrow (b, p)e^k$ , with  $a, b \in A$  and

$p \in \{\text{here}, \text{in}(\text{name}), \text{out}\}$ , involving  $k$  instances of  $e$ , can be decomposed as follows:

$$\begin{aligned} a &\rightarrow (b_1, \text{here})e \\ b_1 &\rightarrow (b_2, \text{here})e \\ &\vdots \\ b_{k-2} &\rightarrow (b_{k-1}, \text{here})e \\ b_{k-1} &\rightarrow (b, p)e \end{aligned}$$

by introducing into the alphabet the *new* symbols  $b_1, \dots, b_{k-1}$ . An analogous observation holds for rules of the kind  $ae^k \rightarrow b$ .

A possible extension of the model, which is nevertheless unimportant with respect to the problems considered in this paper, is to allow the use of constructor and destructor rules. A *constructor rule* is a rule of the kind  $e^k \rightarrow (a, p)$ , where  $a \in A$ ,  $\varepsilon_a = k$ ,  $p \in \{\text{here}, \text{in}(\text{name}), \text{out}\}$  and  $k$  is a positive integer. Informally, a constructor rule for an object  $a \in A$  is a rule which uses  $\varepsilon_a$  free energy units to build the object  $a$ . In other words, we allow transformations from “pure” energy to system objects. Analogously, a *destructor rule* is a rule of the kind  $a \rightarrow e^k$ , where  $a \in A$  and  $\varepsilon_a = k$  (a positive integer). Hence, a destructor rule for an object  $a \in A$  is a rule which transforms the object  $a$  into  $\varepsilon_a$  units of free energy.

A *configuration* of  $\Pi$  is an  $n$ -tuple  $(M_1, \dots, M_m)$  of multisets (over  $A \cup \{e\}$ ) of objects contained in each region of the system.  $(w_1, \dots, w_m)$  is called the *initial configuration*. For two configurations  $(M_1, \dots, M_m)$ ,  $(M'_1, \dots, M'_m)$  of  $\Pi$  we write  $(M_1, \dots, M_m) \Rightarrow (M'_1, \dots, M'_m)$  to denote a *transition* from  $(M_1, \dots, M_m)$  to  $(M'_1, \dots, M'_m)$ , that is, the parallel application of one or more rules of the system. The reflexive and transitive closure of  $\Rightarrow$  is denoted by  $\Rightarrow^*$ . A *final configuration* is a configuration where no rule can be applied.

A *computation* is a sequence of transitions between configurations of  $\Pi$ , starting from the initial configuration. A computation is *successful* if and only if it reaches a final configuration or, in other words, it *halts*. It is understood that the multiset (over  $A$ , that is, not considering free energy units) of objects which occur in  $w_{i_{\text{in}}}$  are the *input values* for the computation. Analogously, the multiset (over  $A$ ) of objects occurring in the output membrane (or emitted from the skin if  $i_{\text{out}} = 0$ ) in the final configuration is the *output* of the computation. A non-halting computation produces no output.

If  $\mathcal{M}$  denotes the set of all possible multisets over  $A$ , then we can define the *function*  $G : \mathcal{M} \rightarrow 2^{\mathcal{M}} \cup \{\perp\}$  *computed by*  $\Pi$  as the (partial) function that to each multiset  $M \in \mathcal{M}$  associates the set  $G(M)$  of possible multisets which can be produced in output by  $\Pi$  when given  $M$  in input. If the computation does not halt, then  $G(M) = \perp$ . Here we stress that, for an halting computation, only one of the multisets in  $G(M)$  is nondeterministically produced in output. With a little abuse of notation, for any fixed computation we denote also this multiset by  $G(M)$ .

Since energy is an additive quantity, it is natural to define the *energy of a multiset*  $M$ , denoted by  $E(M)$ , as the sum of the amounts of energy associated to each instance of the objects which occur into  $M$ . Analogously, the *energy of a configuration*  $C = \{M_1, \dots, M_m\}$ , denoted by  $E(C)$ , is the sum of the amounts of energy associated

to each multiset which occurs into the configuration:  $E(\mathcal{C}) = \sum_{i=1}^m E(M_i)$ . A *conservative energy-based P system* can thus be defined as an energy-based P system for which in every possible computation all configurations have the same amount of energy. Moreover, in a conservative P system it is required that the amount of energy entering the system with the input multiset  $M$  is entirely returned with the output multiset  $G(M)$  at the end of the computation.

In [12] we have shown that energy-based P systems are able to simulate any reversible circuit made of Fredkin gates. Since families  $\{FC_n\}_{n \in \mathbb{N}}$  of reversible Fredkin circuits are able to compute any family  $\{f_n\}_{n \in \mathbb{N}}$  of boolean functions, (families of) energy-based P systems constitute a *universal* model of computation. The simulating P systems considered in [12] are *self-reversible*, meaning that the same system is able to perform both “forward” and “backward” computations, that is, to compute the output values corresponding to any given input values, and vice versa. An interesting aspect of the simulations presented in [12] is that the simulating P systems are also *conservative*: the amount of energy present into the system during computations is constant. Hence it is possible to perform universal computations using only self-reversible and conservative P systems.

This is by no means the first time that energy is considered when dealing with P systems. Considering the energy balancing of processes in a cell was first investigated in [19] and then in [5]. There the energies of all rules to be used in a given step in a membrane are summed up; if the total amount of energies is positive [19] or within a given range [5], then this multiset of rules can be applied if it is maximal with this property. Energy and associations between energy and information have also been considered in [1, 6, 7, 8].

The paper is organized as follows. In section 2 we define conservative computations for energy-based P systems. Moreover, we introduce the NP-hard optimization problem MIN STORAGE and its corresponding NP-complete decision version, CONSCOMP. In section 3 a 2-approximation algorithm is proposed for MIN STORAGE. Section 4 proposes conservative languages with some directions for future research. Finally, section 5 concludes the paper.

## 2 Conservative computations

As we have said above, in a conservative energy-based P system the amount of energy entering the system with the input values is completely returned with the output values at the end of the computation. This means, in particular, that if some free energy units are present in the initial configuration  $\{w_1, \dots, w_m\}$  of a computation then these energy units will occur also in the final configuration. Once the output values have been removed from the output membrane (or, alternatively, once they have been expelled from the skin), this amount of free energy units can be used to perform another computation.

This situation suggests the following scenario. Assume that we have a sequence  $S_{in} = \langle M_1, M_2, \dots, M_k \rangle$  of multisets (over  $A$ ) to be used as input values for an energy-based P system  $\Pi$ . Moreover, assume that we already know that  $\Pi$  will produce the multisets  $G(M_1), \dots, G(M_k)$  when given in input

$M_1, \dots, M_k$ , where  $G(M_i) \neq \perp$  for all  $i \in \{1, 2, \dots, k\}$ . Let  $E(M_1), \dots, E(M_k)$  and  $E(G(M_1)), \dots, E(G(M_k))$  be the energies associated with the input and output multisets, respectively, and let us consider the quantities  $e_i = E(M_i) - E(G(M_i))$ , for all  $i \in \{1, 2, \dots, k\}$ . As we have said, we may assume without loss of generality that all  $e_i$ 's are integer values. We say that the computation of the output sequence  $S_{out} = \langle G(M_1), \dots, G(M_k) \rangle$ , obtained starting from  $S_{in}$ , is *conservative* if the following condition holds:

$$\sum_{i=1}^k e_i = \sum_{i=1}^k E(M_i) - \sum_{i=1}^k E(G(M_i)) = 0$$

This condition formalizes the requirement that the total energy provided by *all* input multisets of  $S_{in}$  is used to build all the output multisets of  $S_{out}$ . If no additional energy, in the form of free energy units, is supplied during the computation of  $S_{out}$  then this condition formalizes also the requirement that the energy entering the system during the computation is equal to the energy leaving it. Of course it may happen that  $e_i > 0$  or  $e_i < 0$  for some  $i \in \{1, 2, \dots, k\}$ . In the former case some free energy units remain into the system after producing  $G(M_i)$ . These energy units can be used during the computation of subsequent output multisets  $G(M_{i+1}), \dots, G(M_k)$ . Hence the P system  $\Pi$  acts as an *accumulator* of energy. Notice that in every conceivable physical realization of a P system there is a bound on the maximum amount  $C$  of energy units (both free and embedded into objects) which can be stored into the system. We call  $C$  the *capacity* of the system.

If the output multisets  $G(M_1), G(M_2), \dots, G(M_k)$  of  $S_{out}$  are computed exactly in this order then, assuming that the system  $\Pi$  starts with zero internal energy, it is easily seen that  $st_1 := e_1$ ,  $st_2 := e_1 + e_2$ ,  $\dots$ ,  $st_k := e_1 + e_2 + \dots + e_k$  is the *sequence of the amounts of energy stored* into the system during the computation of  $S_{out}$ . Notice that  $st_k = 0$  for conservative computations, so that the amount of energy stored into the system at the end of the computation is zero.

In some cases the order with which the output multisets of  $S_{out}$  are computed does not matter. We can thus introduce the following problem: Given an input sequence  $\langle M_1, \dots, M_k \rangle$  and the corresponding output sequence  $\langle G(M_1), \dots, G(M_k) \rangle$ , is there a permutation  $\pi \in S_k$  (the symmetrical group of order  $k$ ) such that the computation of  $G(M_{\pi(1)}), \dots, G(M_{\pi(k)})$  can be performed by an energy-based P system having a predefined capacity  $C$ ? This is a decision problem, whose formal statement follows. (Note that we do not actually need to know the multisets  $M_1, \dots, M_k$  and  $G(M_1), \dots, G(M_k)$ : all we need are the values  $e_i = E(M_i) - E(G(M_i))$ , for  $i \in \{1, 2, \dots, k\}$ .)

Let  $\mathcal{E} = \langle e_1, e_2, \dots, e_k \rangle$  be a finite sequence of integer numbers. For a fixed  $i \in \{1, 2, \dots, k\}$ , the  *$i$ -th prefix sum* of  $\mathcal{E}$  is the value  $\sum_{j=1}^i e_j$ . Let  $C$  be a positive integer; we say that  $\mathcal{E}$  is  *$C$ -feasible* if for each  $i \in \{1, 2, \dots, k\}$  the  $i$ -th prefix sum of  $\mathcal{E}$  is in the closed interval  $[0, C]$ .

**Problem 2.1.** NAME: CONSCOMP.

- **INSTANCE:** a set  $\mathcal{E} = \{e_1, e_2, \dots, e_k\}$  of integer numbers such that  $e_1 + e_2 + \dots + e_k = 0$ , and an integer number  $C > 0$ .

- **QUESTION:** *is there a permutation  $\pi \in S_k$  (the symmetric group of order  $k$ ) such that the sequence  $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(k)}$  is  $C$ -feasible?*  $\square$

The fact that the resulting sequence  $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(k)}$  is  $C$ -feasible can be explicitly written as:

$$0 \leq \sum_{j=1}^i e_{\pi(j)} \leq C \quad \forall i \in \{1, 2, \dots, k\} \quad (1)$$

The CONSCOMP problem can be obviously solved by trying every possible permutation  $\pi$  from  $S_k$ . However, this procedure requires an exponential time with respect to  $k$ , the length of the input sequence. A natural question is whether it is possible to give the correct answer in polynomial time. With the following theorem we show that the CONSCOMP problem is NP-complete, and hence it is very unlikely that a polynomial time algorithm exists that solves it. The proof of this theorem was originally published in [4].

**Theorem 2.1.** *CONSCOMP is NP-complete.*

*Proof.* CONSCOMP is clearly in NP, since a permutation  $\pi \in S_k$  has linear length and verifying whether  $\pi$  is a solution can be done in polynomial time. In order to conclude that CONSCOMP is NP-complete, let us show a polynomial reduction from PARTITION, which is a well known NP-complete problem [9, page 47].

Let  $A = \{a_1, a_2, \dots, a_k\}$  be a set of positive integer numbers, and let  $m = \sum_{i=1}^k a_i$ . The set  $A$  is a positive instance of PARTITION if and only if there exists a set  $A' \subseteq A$  such that  $\sum_{a \in A'} a = \frac{m}{2}$ . If  $m$  is odd then  $A$  is certainly a negative instance, and we can associate it to any negative instance of CONSCOMP. On the other hand, if  $m$  is even we build the corresponding instance  $(\mathcal{E}, C)$  of CONSCOMP by putting  $C = \frac{m}{2}$  and  $\mathcal{E} = \{e_1, e_2, \dots, e_k, e_{k+1}, e_{k+2}\}$ , where  $e_i = -a_i$  for all  $i \in \{1, 2, \dots, k\}$  and  $e_{k+1} = e_{k+2} = \frac{m}{2}$ . It is immediately seen that this construction can be performed in polynomial time.

We claim that  $A$  is a positive instance of PARTITION if and only if  $(\mathcal{E}, C)$  is a positive instance of CONSCOMP. In fact, let us assume that  $A$  is a positive instance of PARTITION. Then there exists a set  $A' \subseteq A$  such that  $\sum_{a \in A'} a = \frac{m}{2}$ , and the corresponding negative elements of  $\mathcal{E}$  constitute a subset  $\mathcal{E}'$  such that  $\sum_{e \in \mathcal{E}'} e = -\frac{m}{2}$ . We build a permutation  $\pi \in S_k$  by selecting first the element  $e_{k+1}$  followed by the elements of  $\mathcal{E}'$  (chosen with any order), and then  $e_{k+2}$  followed by the remaining elements of  $\mathcal{E}$ . It is immediately seen that  $\pi$  satisfies the inequalities stated in (1), and hence  $(\mathcal{E}, C)$  is a positive instance of CONSCOMP. Conversely, let us assume that  $(\mathcal{E}, C)$  is a positive instance of CONSCOMP. Then there exists a permutation  $\pi \in S_k$  that satisfies the inequalities stated in (1). Since the first chosen element cannot be negative, it must necessarily be  $\frac{m}{2}$ . Moreover, since  $C = \frac{m}{2}$ , the second  $\frac{m}{2}$  can be chosen if and only if the energy stored into the system is zero, that is, if and only if there exists a set  $\mathcal{E}' \subseteq \mathcal{E}$  of negative elements whose sum is equal to  $-\frac{m}{2}$ . The opposites of these elements constitute a set  $A' \subseteq A$  such that  $\sum_{a \in A'} a = \frac{m}{2}$ , and thus we can conclude that  $A$  is a positive instance of PARTITION.  $\square$

The CONSCOMP problem naturally leads to the formulation of the following optimization problem.

**Problem 2.2.** NAME: MIN STORAGE.

- INSTANCE: a set  $\mathcal{E} = \{e_1, e_2, \dots, e_k\}$  of integer numbers such that  $e_1 + e_2 + \dots + e_k = 0$ .
- SOLUTION: a permutation  $\pi \in S_k$  such that  $\sum_{j=1}^i e_{\pi(j)} \geq 0$  for each  $i \in \{1, 2, \dots, k\}$ .
- MEASURE:  $\max_{1 \leq i \leq k} \sum_{j=1}^i e_{\pi(j)}$ . □

Informally, the output of MIN STORAGE is the minimum value of  $C$  for which there exists a permutation  $\pi \in S_k$  such that the sequence  $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(k)}$  is  $C$ -feasible. Notice that a trivial upper bound for the value of  $C$  is:

$$\sum_{i \in \{1, 2, \dots, k\}: e_i > 0} e_i = \frac{1}{2} \sum_{i=1}^k |e_i|$$

while a trivial lower bound is  $\max_{1 \leq i \leq k} |e_i|$ .

It is immediate to see that MIN STORAGE is in the class NPO [2, page 27]. In fact, checking whether some given integers  $e_1, e_2, \dots, e_k$  sum up to zero can be trivially done in polynomial time; each feasible solution has linear length and besides it can be verified in polynomial time whether a given permutation  $\pi \in S_k$  is a feasible solution; finally, the measure function can be computed in polynomial time. Since the underlying decision problem CONSCOMP is NP-complete, we can immediately conclude that MIN STORAGE is NP-hard [2, page 30]. As with the CONSCOMP decision problem, this means that it is very unlikely that a polynomial time algorithm exists that gives the correct solution to every instance of MIN STORAGE.

If we drop the requirement  $e_1 + e_2 + \dots + e_k = 0$  in the instances of CONSCOMP and MIN STORAGE we obtain the following problems.

**Problem 2.3.** NAME: CONSCOMP II.

- INSTANCE: a set  $\mathcal{E} = \{e_1, e_2, \dots, e_k\}$  of integer numbers, and an integer number  $C > 0$ .
- QUESTION: is there a permutation  $\pi \in S_k$  such that the sequence  $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(k)}$  is  $C$ -feasible? □

**Problem 2.4.** NAME: MIN STORAGE II.

- INSTANCE: a set  $\mathcal{E} = \{e_1, e_2, \dots, e_k\}$  of integer numbers.
- SOLUTION: a permutation  $\pi \in S_k$  such that  $\sum_{j=1}^i e_{\pi(j)} \geq 0$  for each  $i \in \{1, 2, \dots, k\}$ .
- MEASURE:  $\max_{1 \leq i \leq k} \sum_{j=1}^i e_{\pi(j)}$ . □

Notice that it may happen that, for some instance  $\mathcal{E}$ , the set of feasible solutions of MIN STORAGE II is empty. In such a case, we put the solution equal to 0 by definition.

CONSCOMP II is obviously NP–complete, by the restriction property [9, page 63], since it contains CONSCOMP as a particular case. Notice that interesting instances of CONSCOMP II are obtained only for values  $e_1, e_2, \dots, e_k$  taken from the interval  $[-C, C]$  of integers. In fact, if  $e_i \notin [-C, C] \cap \mathbb{Z}$  for some  $i \in \{1, 2, \dots, k\}$  (a situation which can be verified in linear time) then the instance does not admit a solution. Since CONSCOMP II is the decision version of MIN STORAGE II, and it is NP–complete, MIN STORAGE II is NP–hard. Also for this problem, if the set of feasible solutions is not empty then a trivial upper bound for the value of the optimal solution is  $\sum_{i \in \{1, 2, \dots, k\} : e_i > 0} e_i$ , while a trivial lower bound is  $\max_{1 \leq i \leq k} |e_i|$ .

We conclude this section by observing that a different interpretation of CONSCOMP II and MIN STORAGE II can be given without reference to conservativeness and conservative computations. Let us consider a merchant whose business involves  $k$  cities. When the merchant arrives to the  $i$ -th city he either buys or sells some good. If he sells, he earns an amount  $e_i$  of money; if he buys, he spends an amount  $e_i$  of money. Hence we can associate to each city a positive integer earning  $e_i > 0$  or a “negative earning” (that is, an expense)  $e_i < 0$ . The CONSCOMP II problem can thus be seen as the formalization of the following problem: Given a wallet that may contain a maximum amount  $C$  of money, is the merchant able to make a tour of all cities (as in the TSP problem) without going out of money or earning too much? We call this interpretation of CONSCOMP II the TRAVELING MERCHANT problem. Analogously, MIN STORAGE II can be seen as the formalization of the problem which asks what is the minimum capacity of the wallet that allows the merchant to perform a tour of all cities. An appropriate name for this interpretation of MIN STORAGE II seems to be MIN WALLET.

### 3 Approximating MIN STORAGE

Since the MIN STORAGE problem is NP–hard, a natural question is how well its optimal solutions can be approximated in polynomial time. Precisely, we ask ourselves whether there exists a PTAS (Polynomial Time Approximation Scheme) or even an FPTAS (Fully Polynomial Time Approximation Scheme) for MIN STORAGE.

The fact that PARTITION can be thought of as a particular case of the SUBSET SUM problem (indeed, a direct polynomial reduction from SUBSET SUM to CONSCOMP can be trivially derived from the proof of Theorem 2.1) could suggest that a modification to the standard FPTAS for SUBSET SUM [11] could lead to an FPTAS for MIN STORAGE. However, differently from SUBSET SUM, CONSCOMP is NP–complete in the strong sense (and hence MIN STORAGE is NP–hard in the strong sense), as it is easily proved in the following. Let us consider the 3–PARTITION problem [9, page 224].

**Problem 3.1.** NAME: 3–PARTITION.

- **INSTANCE:** Set  $A$  of  $3m$  elements, a bound  $B \in \mathbb{Z}^+$ , and a size  $s(a) \in \mathbb{Z}^+$  for each  $a \in A$  such that  $B/4 < s(a) < B/2$  and such that  $\sum_{a \in A} s(a) = mB$ .
- **QUESTION:** Can  $A$  be partitioned into  $m$  disjoint sets  $A_1, A_2, \dots, A_m$  such that, for  $1 \leq i \leq m$ ,  $\sum_{a \in A_i} s(a) = B$  (note that each  $A_i$  must therefore contain exactly three elements from  $A$ )?  $\square$

The 3-PARTITION problem is NP-complete in the strong sense [9, page 224]. A simple modification to the proof of Theorem 2.1 allows to build an explicit polynomial reduction from 3-PARTITION to CONSCOMP, thus proving that also CONSCOMP is strongly NP-complete. As it is well known [2, page 116] this fact prevents the existence of an FPTAS for MIN STORAGE. Hence the next natural question is whether there exists a PTAS for MIN STORAGE. This possibility is still under investigation.

Here we show that MIN STORAGE is in the class APX of problems which admit a constant factor polynomial time approximation algorithm. Let us consider the following algorithm.

```

APPROX MIN STORAGE ( $\{e_1, e_2, \dots, e_k\}$ )
 $M \leftarrow \max_{1 \leq i \leq k} |e_i|$ 
 $E_p = E_n = \emptyset$ 
for  $i \leftarrow 1$  to  $k$ 
  do if  $e_i \geq 0$ 
    then  $E_p = E_p \cup \{e_i\}$ 
    else  $E_n = E_n \cup \{e_i\}$ 
 $max \leftarrow st \leftarrow 0$ 
while  $E_p \neq \emptyset$ 
  do if  $st < M$ 
    then  $x \leftarrow$  an element of  $E_p$ 
     $st \leftarrow st + x$ 
    if  $st > max$  then  $max \leftarrow st$ 
     $E_p = E_p \setminus \{x\}$ 
  else  $x \leftarrow$  an element of  $E_n$ 
     $st \leftarrow st + x$ 
     $E_n = E_n \setminus \{x\}$ 
return  $max$ 

```

The algorithm works as follows. The variable  $M$  is set to  $\max_{1 \leq i \leq k} |e_i|$ , which is a theoretical lower bound for the optimal solution. While scanning the elements of the instance in order to compute  $M$ , we can divide them into negative ( $E_n$ ) and non negative ( $E_p$ ) elements. The elements which are equal to 0 are uninfluent to the problem and can be put both in  $E_n$  and/or in  $E_p$ ; in the pseudo-code above we have put them all into  $E_p$ . The variable  $st$  records the energy which is currently stored into the system. The idea is to make this variable assume values only into the interval  $[0, 2M]$  (actually, into the interval  $[0, 2M - 1]$ ). The variable  $max$ , which contains the value returned at the end of the computation, records the maximum of

the values assumed by  $st$  into the subinterval  $[M, 2M]$ . Since the optimal solution cannot be less than  $M$ , this strategy allows the algorithm to return a value which is by a factor at most 2 greater than the optimal solution.

Notice that at the end of the execution only some elements of  $E_n$  will not be chosen. Since  $\sum_{i=1}^k e_i = 0$ , these elements will lead  $st$  to 0 and they will not affect the returned result. If we are required to build a permutation  $\pi \in S_k$  that corresponds to the solution found by the algorithm then it suffices to store the elements into an array as they are selected; the remaining elements from  $E_n$  can be chosen in any order to fill the final portion of the array.

A direct inspection of the pseudo-code reveals that the time complexity of the algorithm is linear with respect to  $k$ , the length of the input sequence.

**Proposition 3.1.** APPROX MIN STORAGE is a 2-approximation algorithm for MIN STORAGE.

*Proof.* We have to prove that, for any instance  $\mathcal{E}$  of MIN STORAGE, the algorithm 2-APPROX MIN STORAGE always returns a solution  $sol(\mathcal{E})$  which is at most the double of the optimal solution  $opt(\mathcal{E})$ :

$$sol(\mathcal{E}) \leq 2 \cdot opt(\mathcal{E})$$

First of all we note that the value of  $st$  is always non negative. In fact, when the execution starts the value of  $st$  is set equal to 0. In the subsequent steps the algorithm chooses a negative element of  $\mathcal{E}$  if and only if  $st \geq M$ . Since the absolute values of all negative elements are not greater than  $M$ , at the next iteration the value of  $st$  will remain non negative.

On the other hand, the value of  $st$  is always less than  $2M$ . In fact, the algorithm chooses a positive element of  $\mathcal{E}$  if and only if  $st < M$ . Since the chosen element cannot be greater than  $M$ , the resulting value of  $st$  remains less than  $2M$ .

The value returned by APPROX MIN STORAGE is the maximum value comprised between  $M$  and  $2M - 1$  assumed by the variable  $st$ . Since  $opt(\mathcal{E}) \geq M$  and  $sol(\mathcal{E}) < 2M$ , we can conclude that  $sol(\mathcal{E}) < 2M \leq 2 \cdot opt(\mathcal{E})$ .  $\square$

## 4 Conservative languages

As a direction for future work we propose to study the properties of *conservative languages*, which can be defined as follows. For a fixed integer  $C > 0$ , we first define the alphabet  $\Sigma_C = \mathbb{Z} \cap [-C, C]$  whose  $2C + 1$  elements are the integers from the interval  $[-C, C]$ . Moreover, let  $\Sigma_C^k$  be the set of strings of length  $k$  composed by symbols taken from  $\Sigma_C$ .

**Definition 4.1.** For any integer  $k \geq 1$ , the language  $CONS_C(k)$  is the following set of strings:

$$CONS_C(k) = \left\{ w = \sigma_1 \sigma_2 \cdots \sigma_k \in \Sigma_C^k : 0 \leq \sum_{j=1}^i \sigma_j \leq C \right. \\ \left. \text{for all } i \in \{1, 2, \dots, k\}, \text{ and } \sum_{i=1}^k \sigma_i = 0 \right\}$$

Moreover, we define the languages  $\text{CONS}_C = \bigcup_{k \geq 1} \text{CONS}_C(k)$  and  $\text{CONS} = \bigcup_{C \geq 1} \text{CONS}_C$ .

Depending upon the need it may be appropriate to include or not the empty string  $\lambda$  in  $\text{CONS}_C$ . Since the addition of zeroes in a given string  $w$  does not change the values of its prefix sums we can immediately conclude that for all  $k \geq 1$  the language  $\text{CONS}_C(k+1)$  contains an isomorphic image of  $\text{CONS}_C(k)$ . It is also immediate to see that the languages  $\text{CONS}_C$  form the following (infinite) hierarchy:

$$\text{CONS}_1 \subset \text{CONS}_2 \subset \dots \subset \text{CONS}_C \subset \dots$$

In fact, for any fixed positive integer  $C$  let  $w_C = C, -C$  be the string formed by the juxtaposition of the symbols  $C$  and  $-C$ . Then clearly  $w_C \in \text{CONS}_C \setminus \text{CONS}_{C-1}$  for all integers  $C > 1$ .

Let us now consider the following problems.

**Problem 4.1.** *Given a positive integer  $C$ , and  $\sigma_1, \sigma_2, \dots, \sigma_k \in \Sigma_C$  such that  $\sum_{i=1}^k \sigma_i = 0$ , can we form a word  $w \in \text{CONS}_C(k)$  by taking each  $\sigma_i$  exactly once?  $\square$*

The formalization of this problem is  $\text{CONSCOMP}$ , and hence it is NP-complete in the strong sense.

**Problem 4.2.** *Given  $\sigma_1, \sigma_2, \dots, \sigma_k \in \Sigma_C$  such that  $\sum_{i=1}^k \sigma_i = 0$ , what is the minimum value of  $C$  such that there exists  $w \in \text{CONS}_C(k)$ , obtained by taking each  $\sigma_i$  exactly once?  $\square$*

The formalization of this problem is  $\text{MIN STORAGE}$ , and hence it is NP-hard in the strong sense. A new interesting problem is the following.

**Problem 4.3.** *Given a positive integer  $C$ , and  $\sigma_1, \sigma_2, \dots, \sigma_k \in \Sigma_C$ , what is the longest word  $w \in \Sigma_C^\ell$ , with  $0 \leq \ell \leq k$ , that can be formed by picking each  $\sigma_i$  at most once, such that  $w \in \text{CONS}_C$ ?  $\square$*

Formally, this problem can be stated as follows.

**Problem 4.4.** NAME: MAX STRING LENGTH.

- INSTANCE: a set  $\{e_1, e_2, \dots, e_k\}$  of integer numbers, and an integer number  $C > 0$ .
- SOLUTION: a permutation  $\pi \in S_k$ .
- MEASURE:  $\max_{0 \leq i \leq k} \left\{ i \mid 0 \leq \sum_{j=1}^r e_{\pi(j)} \leq C \text{ for all } r \in \{1, \dots, i\}, \text{ and } \sum_{j=1}^i e_{\pi(j)} = 0 \right\}$ .  $\square$

It is immediate to see that MAX STRING LENGTH is in NPO. In fact, each feasible solution has linear length, and the measure function can be computed in polynomial time.

The decision version of this optimization problem asks whether, given a set  $\{e_1, e_2, \dots, e_k\}$  of integer numbers, a positive integer number  $C$ , and a non negative integer number  $L$ , there exists a permutation  $\pi \in S_k$  such that:

$$\max_{0 \leq i \leq k} \left\{ i \mid 0 \leq \sum_{j=1}^r e_{\pi(j)} \leq C \text{ for all } r \in \{1, \dots, i\}, \text{ and } \sum_{j=1}^i e_{\pi(j)} = 0 \right\} \geq L$$

This decision problem, that we name **STRING LENGTH**, is clearly NP-complete in the strong sense. In fact, let  $(\mathcal{E} = \{e_1, \dots, e_k\}, C)$  be an instance of **CONSCOMP**. We build the corresponding instance of **STRING LENGTH** by putting  $L = k$ . Then, a solution to this last problem immediately corresponds to a solution of **CONSCOMP**. As a consequence, we can conclude that the optimization problem **MAX STRING LENGTH** is NP-hard in the strong sense.

As for the computational power of energy-based P systems we propose the introduction of languages which can be generated using a bounded (fixed, logarithmic, polynomial, etc.) amount of energy or capacity, and the subsequent investigation of the properties of these languages. Another possibility is to define *families*  $\{P_n\}_{n \in \mathbb{N}}$  of energy-based P systems, where  $P_n$  uses  $n$  units of energy. Then, we can define the language generated by  $\{P_n\}_{n \in \mathbb{N}}$  as  $\bigcup_{n \in \mathbb{N}} L_n$ , where  $L_n$  is the language generated by  $P_n$ . This approach is reminiscent of circuit complexity [21]. Moreover, having defined both an input and an output membrane, we can view energy-based P systems as devices which map multisets into multisets, as we have done in this paper. With respect to this point of view, instead of asking what multisets can be generated by the system we can ask what mappings can be realized by imposing different bounds on the amount of resources that the system is allowed to use.

## 5 Conclusions

Starting from conservative energy-based P systems we have introduced the notion of conservative computations as computations in which the initial energy of the system is the same as the energy at the end of the computation.

We have shown that conservative computations induce **MIN STORAGE**, a new NP-hard optimization problem, and **CONSCOMP**, its naturally associated decision problem. Being **CONSCOMP** NP-complete in the strong sense, the existence of an FPTAS for **MIN STORAGE** is prevented. The existence of a PTAS is still under investigation. In this paper we have presented a 2-approximation algorithm for **MIN STORAGE**, thus proving that the problem is in the complexity class APX.

Finally, we have introduced conservative languages.

## References

- [1] G. Alford. Membrane systems with heat control. In *Pre-Proceedings of the Workshop on Membrane Computing*, Curtea de Arges, Romania, August 2002. Available at: <http://psystems.disco.unimib.it/>

- [2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti–Spaccamela, M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties*. Springer–Verlag, 1999.
- [3] J. Castellanos, G. Păun, A. Rodríguez–Paton. P systems with work objects. In *IEEE 7<sup>th</sup> International Conference on String Processing and Information Retrieval*, SPIRE 2000, La Coruna, Spain, 2000, pp. 64–74. See also CDMTCS Technical Report 123, University of Auckland, 2000. Available at: <http://www.cs.auckland.ac.nz/CDMTCS>
- [4] G. Cattaneo, G. Della Vedova, A. Leporati, R. Leporini. Towards a Theory of Conservative Computing. Accepted on *International Journal of Theoretical Physics*. Preprint available at <http://arxiv.org/abs/quant-ph/0211085>, November 2002.
- [5] R. Freund. Energy–Controlled P Systems. In *Membrane Computing*, Proceedings of the International Workshop WMC–CdeA 2002, Curtea de Arges, Romania, August 2002, Lecture Notes in Computer Science 2597, Springer, 2002, pp. 247–260.
- [6] P. Frisco. The conformon–P system: a molecular and cell biology–inspired computability model. *Theoretical Computer Science*, 312:295–319, 2004.
- [7] P. Frisco, S. Ji. Info–energy P systems. In *Proceedings of DNA 8, Eighth International Meeting on DNA Based Computers*, Hokkaido University, Japan, June 2002.
- [8] P. Frisco, S. Ji. Towards a Hierarchy of Conformons–P Systems. In *Membrane Computing*, Proceedings of the International Workshop WMC–CdeA 2002, Curtea de Arges, Romania, August 2002, Lecture Notes in Computer Science 2597, Springer, 2002, pp. 302–318.
- [9] M. R. Garey, D. S. Johnson. *Computers and Intractability. A Guide to the Theory on NP–Completeness*. W. H. Freeman and Company, 1979.
- [10] G. V. Gens, E. V. Levner. Computational complexity of approximation algorithms for combinatorial problems. *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 74, Springer–Verlag, Berlin, 1979, pp. 292–300.
- [11] O. H. Ibarra, C. E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of the ACM*, **22**, 1975, pp. 463–468.
- [12] A. Leporati, C. Zandron, G. Mauri. Simulating the Fredkin Gate with Energy–based P Systems. In [17], 292–308.
- [13] G. Mauri, A. Leporati. On the Computational Complexity of Conservative Computing. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, Lecture Notes in Computer Science 2747, Springer–Verlag Heidelberg, 2003, pp. 92–112.

- [14] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998. Available at: <http://www.tucs.fi/Publications/techreports/TR208.php>
- [15] G. Păun. Computing with Membranes. An Introduction. *Bulletin of the EATCS*, 67:139–152, February 1999.
- [16] G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.
- [17] G. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini (Eds.). Second Brainstorming Week on Membrane Computing, Sevilla, Spain, February 2–7, 2004. Department of Computer Sciences and Artificial Intelligence, University of Sevilla TR 01/2004.
- [18] G. Păun, G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287(1):73–100, 2002.
- [19] G. Păun, Y. Suzuki, H. Tanaka. P Systems with energy accounting. *International Journal Computer Math.*, 78(3):343–364, 2001.
- [20] The P systems Web page: <http://psystems.disco.unimib.it/>
- [21] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer–Verlag, 1999.