Triangle: Language, Literature, Computation, n. 7, 2012
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

Networks of Bio-inspired Processors ⋆

Fernando Arroyo Montoro[1], Juan Castellanos[2], Victor Mitrana[1], Eugenio Santos[1], José M. Sempere[3]

[1] Departamento Lenguajes, Proyectos y Sistemas Informáticos
Escuela Universitaria de Informática
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: {farroyo,esantos}@eui.upm.es, victor.mitrana@upm.es

[2] Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: jcastellanos@fi.upm.es

[3] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Valencia, Spain
E-mail: jsempere@dsic.upv.es

1 Introduction

The goal of this work is twofold. Firstly, we propose a uniform view of three types of accepting networks of bio-inspired processors: networks of evolutionary processors, networks of splicing processors and networks of genetic processors. And, secondly, we survey some features of these networks: computational

⋆ Work partially supported by the Spanish Ministry of Science and Innovation under coordinated research project TIN2011-28260-C03-00 and research projects TIN2011-28260-C03-01, TIN2011-28260-C03-02 and TIN2011-28260-C03-03.

power, computational and descriptional complexity, the existence of universal networks, efficiency as problem solvers and the relationships among them.

These networks are based on a rather common architecture for parallel and distributed symbolic processing, related to the Connection Machine [28] and the Logic Flow paradigm [24], and they consist of several processors, each of which is placed in a node in a virtual complete graph, which can handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then the local data become mobile agents which can navigate in the network following a given protocol. Only that data which is able to pass a filtering process can be communicated. This filtering process may require that some conditions imposed by the sending processor be satisfied by the receiving processor or by both processors. All the nodes simultaneously send their data and the receiving nodes use a variety of strategies to handle, also simultaneously, all the arriving messages (see [25, 28]).

The general idea briefly presented above is modified here using a method inspired by cell biology. Each processor in a node is very simple, either an evolutionary, a splicing or a genetic processor. The three types of processors differ from each other by the operation they carry out.

By an evolutionary processor we mean a processor which can perform very simple operations: namely, point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell that contains genetic information encoded in DNA sequences which may evolve by local evolutionary events: that is, point mutations. Each node is specialized for just one of these evolutionary operations.

By a splicing processor we mean a processor that can perform the splicing operation which is one of the basic mechanisms by which the DNA sequences are recombined under the effect of enzymatic activities.

By a genetic processor we mean a processor that can perform two different types of operations: either a pure mutation operation (i.e. the substitution operation in the evolutionary processors) or a full and massive crossover operation between strings which can be considered as a splicing operation with null contexts between strings.

Furthermore, the data in each node is organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place.

A series of papers was devoted to different variants of this model viewed as language generating devices (see [2, 3, 4, 5, 6, 12, 13, 17, 19]). The paper [42] is an early survey in this area. Similar ideas may be found in other bio-inspired models: for example, *tissue-like membrane systems* [51] or models from Distributed Computing area like *parallel communicating grammar systems* [47].

2 Basic Definitions

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called word over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. The length of a word x is denoted by $|x|$ while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$.

In the course of its evolution, the genome of an organism mutates by different processes. At the level of individual genes the evolution proceeds by local operations (point mutations) which substitute, insert and delete nucleotides of the DNA sequence. In what follows, we define some rewriting operations that will be referred to as *evolutionary operations* since they may be viewed as linguistic formulations of local gene mutations. We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise.} \end{cases}$

Note that a rule such as the one above is applied to all occurrences of the letter a in different copies of the word w. An implicit assumption is that arbitrarily many copies of w are available.

- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$
$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, \\ \{w\}, \text{ otherwise} \end{cases}$
$\sigma^l(w) = \begin{cases} \{v : \ w = av\}, \\ \{w\}, \text{ otherwise} \end{cases}$
- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then $\sigma^*(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}$,
$\sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}$.

$\alpha \in \{*, l, r\}$ expresses the way a deletion or insertion rule is applied to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. The note for the substitution operation mentioned above remains valid for insertion and deletion at any position. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

$$M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w) \quad \text{and} \quad M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),$$

respectively.

For two disjoint and nonempty subsets P and F of an alphabet V and a word z over V, we define the following two predicates

$$rc_s(z; P, F) \equiv P \subseteq alph(z) \ \wedge \ F \cap alph(z) = \emptyset$$
$$rc_w(z; P, F) \equiv alph(z) \cap P \neq \emptyset \ \wedge \ F \cap alph(z) = \emptyset.$$

The construction of these predicates is based on *context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, both conditions require that no forbidding symbol be present in w; furthermore the first condition requires all permitting symbols to appear in w, while the second one requires at least one permitting symbol to appear in w. It is plain that the first condition is stronger than the second one.

For every language $L \subseteq V^*$ and $\beta \in \{s, w\}$, we define:

$$rc_\beta(L, P, F) = \{z \in L \mid rc_\beta(z; P, F)\}.$$

An *evolutionary processor over V* is a 5-tuple (M, PI, FI, PO, FO), where:

– Either ($M \subseteq Sub_V$) or ($M \subseteq Del_V$) or ($M \subseteq Ins_V$). The set M represents the set of evolutionary rules of the processor. As can be seen, a processor is "specialized" in one evolutionary operation only.

– $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor (with $PI \cap FI = \emptyset$ and $PO \cap FO = \emptyset$).

An evolutionary processor such as the one above with $PI = PO = P$ and $FI = FO = F$ is called a *uniform evolutionary processor* and is defined as the triple (M, P, F). We denote the set of (uniform) evolutionary processors over V by $(U)EP_V$. Clearly, the (uniform) evolutionary processor described here is a mathematical concept similar to that of an evolutionary algorithm, both being inspired by Darwinian evolution. As we have mentioned above, the rewriting operations we have considered might be interpreted as mutations and the filtering process described might be viewed as a selection process. Recombination is missing but evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [57]. However, another type of processor based on recombination only, called a splicing processor, has been the focus of a series of studies which will be surveyed in the sections below.

3 Three Variants of Accepting Networks of Evolutionary Processors

An *accepting network of evolutionary processors* (ANEP for short) is an 8-tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

- V and U are the input and network alphabets, respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph without loops with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network.
- $N : X_G \longrightarrow EP_U$ is a mapping which associates each node $x \in X_G$ with the evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $\beta : X_G \longrightarrow \{s, w\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

$$\text{input filter: } \rho_x(\cdot) = rc_{\beta(x)}(\cdot; PI_x, FI_x),$$
$$\text{output filter: } \tau_x(\cdot) = rc_{\beta(x)}(\cdot; PO_x, FO_x).$$

That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. Moreover, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.

- $x_I, x_O \in X_G$ are the *input* and the *output* node of Γ, respectively.

An *Accepting Network of Uniform Evolutionary Processors* (UANEP for short) is an ANEP with uniform evolutionary processors only.

We say that $card(X_G)$ is the size of Γ. If α and β are constant functions, then the network is said to be *homogeneous*. In the theory of networks some types of underlying graphs are common (for example *rings, stars, grids*, etc.). In most of the cases considered here, we focus on *complete* networks (i.e., networks having a complete underlying graph). The last section is an exception, as we discuss an incomplete [U]ANEP that simulates a given ANEPFC (see the meaning of the abbreviation ANEPFC in the next subsection).

A *configuration* of an [U]ANEP Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node at a given moment. Given a word $w \in V^*$, the initial configuration of Γ on w is defined by $C_0^{(w)}(x_I) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G - \{x_I\}$.

When changed for an evolutionary step, each component $C(x)$ of configuration C is changed in accordance with the set of evolutionary rules M_x associated with node x and the way the rules $\alpha(x)$ are applied. Formally, we say that configuration C' is obtained in *one evolutionary step* from configuration C, written as $C \Longrightarrow C'$, iff

$$C'(x) = M_x^{\alpha(x)}(C(x)) \text{ for all } x \in X_G.$$

When changed for a communication step, each node processor $x \in X_G$ sends one copy of each word it has which can pass the output filter of x to all the node processors connected to x. And it receives all the words sent by any node processor connected with x providing that they can pass its input filter. Formally, we say that configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y)))$$

for all $x \in X_G$. Note that words which leave a node are removed from that node. If they cannot pass the input filter of any node, they are lost.

A model closely related to that of ANEPs, introduced in [23] and further studied in [22, 31], is that of *accepting networks of evolutionary processors with filtered connections* (ANEPFCs for short). An ANEPFC may be viewed as an ANEP in which the filters are shifted from the nodes on the edges. Therefore, instead of having a filter at both ends of an edge in each direction, there is only one filter independently of the direction.

An ANEPFC is a 9-tuple

$$\Gamma = (V, U, G, \mathcal{R}, \mathcal{N}, \alpha, \beta, x_I, x_O),$$

where:

- V, U, $G = (X_G, E_G)$, have the same meaning as for ANEP,
- $\mathcal{R} : X_G \longrightarrow 2^{Sub_U} \cup 2^{Del_U} \cup 2^{Ins_U}$ is a mapping which associates each node with *the set of evolutionary rules* that can be applied in that node. Note that each node is associated only with one type of evolutionary rules: namely, for every $x \in X_G$ either $\mathcal{R}(x) \subset Sub_U$ or $\mathcal{R}(x) \subset Del_U$ or $\mathcal{R}(x) \subset Ins_U$ holds.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives *the action mode of the rules* of node x on the words existing in that node.
- $\mathcal{N} : E_G \longrightarrow 2^U \times 2^U$ is a mapping which associates each edge $e \in E_G$ with *the permitting and forbidding filters of that edge*; formally, $\mathcal{N}(e) = (P_e, F_e)$, with $P_e \cap F_e = \emptyset$.
- $\beta : E_G \longrightarrow \{s, w\}$ defines *the filter type of an edge*.
- $x_I, x_O \in X_G$ are *the input and the output node* of Γ, respectively.

Note that every ANEPFC can be immediately transformed into an equivalent ANEPFC with a complete underlying graph by adding the edges that are missing and associating with them filters that do not allow any words to pass. Note that such a simplification is not always possible for ANEPs.

A configuration of an ANEPFC is defined in the same way as the configuration of an ANEP (see above). An evolutionary step is also defined in the same way as above.

Otherwise, when changed for a communication step, in an ANEPFC, each node-processor $x \in X_G$ sends one copy of each word it contains to every node-processor y connected to x, provided they can pass the filter of the edge between x and y. It keeps no copy of these words but receives all the words sent by any node processor z connected with x providing that they can pass the filter of the edge between x and z. In this case, no word is lost.

Let Γ be an [U]ANEP[FC], the computation of Γ on the input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is

the initial configuration of Γ defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G$, $x \neq x_I$, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Note that the configurations are changed by alternative evolutionary and communication steps. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by configuration $C_{i-1}^{(w)}$.

A computation *halts* (and it is said to be *halting*) if one of the following two conditions holds:

(i) There exists a configuration in which the set of words existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation*.

(ii) There exist two identical configurations obtained either in consecutive evolutionary steps or in consecutive communication steps.

The *language accepted* by the [U]ANEP[FC] Γ is $L_a(\Gamma) = \{w \in V^* \mid$ the computation of Γ on w is an accepting one$\}$. We denote by $\mathcal{L}([U]ANEP[FC])$ the class of languages accepted by [U]ANEP[FC]s.

We say that an [U]ANEP[FC] Γ decides the language $L \subseteq V^*$, and write $L(\Gamma) = L$ iff $L_a(\Gamma) = L$ and the computation of Γ on every $x \in V^*$ halts.

3.1 Computational power of [U]ANEP[FC]s

The results obtained so far ([40, 38, 22, 23]) state that non-deterministic Turing machines can be simulated by ANEPs and ANEPFCs.

Therefore we have:

Theorem 1. *Both $\mathcal{L}(ANEP)$ and $\mathcal{L}(ANEPFC)$ equal the class of recursively enumerable languages.*

It is clear that filters associated with each node of an ANEP allow the computation to be closely controlled. However, by moving the filters from the nodes to the edges, the possibility of controlling the computation seems to be diminished. For instance, data cannot be lost during the communication steps. In spite of this, we have seen that ANEPFCs are still computationally complete. This means that moving the filters from the nodes to the edges does not decrease the computational power of the model. Although the two variants are equivalent from the point of view of computational power, a direct proof would have been worthwhile. In [8] it was shown that the two models can efficiently simulate each other: namely, each computational step in one model is simulated in a constant number of computational steps in the other. This

is particularly useful when the solution of a problem needs to be translated from one model to the other. Note that a translation via a Turing machine, by the constructions shown in [40, 38, 22, 23] squares the time complexity of the new solution. A natural question arises: What is the computational power of UANEPs? The answer was given in [9] where the time complexity preserving simulation between ANEPs and ANEPFCs was extended to UANEPs. More precisely, it was shown that each pair of networks among the three variants efficiently simulates each other. Consequently, we can state the first main result of this section:

Theorem 2.
1. Each class $\mathcal{L}([U]ANEP[FC])$ *equals the class of recursively enumerable languages.*
2. Each pair of networks among the three variants efficiently simulates each other.

These results can be improved by showing that each recursively enumerable language can be accepted by an ANEP[FC] of constant size. More precisely:

Theorem 3. [32, 31, 6]
1. Every recursively enumerable language can be accepted by an ANEP of size 7.
2. Every recursively enumerable language can be accepted by an ANEPFC of size 16.

The second result can be extended to characterize the class **NP**. Although the first result cannot be extended to a similar succinct characterization of **NP**, as the proof in [6] is based on the simulation of a phrase-structure grammar, such a succinct characterization of **NP** is proposed in [32, 31].

Theorem 4.
1. A language is in **NP** *if and only if it is accepted by an ANEP of size 10 in polynomial time.*
2. A language is in **NP** *if and only if it is accepted by an ANEPFC of size 16 in polynomial time.*

We do not know whether similar results like those in Theorems 3 or 4 holds for UANEPs.

4 Accepting Networks of Splicing Processors

In the case of Accepting Networks of Splicing Processors (ANSP for short), the point mutations associated with each node are replaced by the missing operation (recombination), which is present here in the form of splicing. This computing model is to some extent similar to the test tube distributed systems based on splicing introduced in [16] and further explored in [48]. However, there are several differences: first, the model proposed in [16] is a language generating mechanism while ours is an accepting one; second, we use a single splicing step, while every splicing step in [16] is actually an infinite process consisting of iterated splicing steps; third, each splicing step in our model is reflexive; fourth, the filters of our model are based on random context conditions while those in [16] are based on membership conditions; fifth, at every splicing step a set of auxiliary words, always the same and particular to every node, is available for splicing. Along the same lines, we should stress the differences between this model and the time-varying distributed H systems, a generative model introduced in [50] and further studied in [41, 49, 46]. The computing strategy of such a system is that the passing of words from a set of rules to another one is specified by a cycle. Only those words that are obtained at one splicing step by using a set of rules are passed in a circular way to the next set of rules. This means that words which cannot be spliced at some step disappear from the computation while words produced at different splicing steps cannot be spliced together. Now, the differences between time-varying distributed H systems and ANSPs are evident: each node of an ANSP has a set of auxiliary words, words obtained at different splicing steps in different nodes can be spliced together, words are not communicated in a circular way, since identical copies of the same word are sent out to all the nodes, the communication is controlled by filters.

A *splicing rule* over a finite alphabet V is a word of the form $u_1\#u_2\$v_1\#v_2$ such that u_1, u_2, v_1, and v_2 are in V^* and such that \$ and # are two symbols not in V.

For a splicing rule $r = u_1\#u_2\$v_1\#v_2$ and for $x, y, w, z \in V^*$, we say that r produces (w, z) from (x, y) (denoted by $(x, y) \vdash_r (w, z)$) if there exist some $x_1, x_2, y_1, y_2 \in V^*$ such that $x = x_1u_1u_2x_2$, $y = y_1v_1v_2y_2$, $z = x_1u_1v_2y_2$, and $w = y_1v_1u_2x_2$.

For a language L over V and a set of splicing rules R we define

$$\sigma_R(L) = \{z, w \in V^* \mid (\exists u, v \in L, r \in R)[(u, v) \vdash_r (z, w)]\}.$$

A *splicing processor* over V is a 6-tuple (S, A, PI, FI, PO, FO), where S a finite set of splicing rules over V, A a finite set of auxiliary words over V, and all the other parameters have the same meaning as in the definition of evolutionary processors. Now an ANSP can be defined in the same way as an ANEP except that the processors associated with nodes are splicing processors.

A *configuration* of an ANSP Γ is a mapping $C : X_G \to 2^{U^*}$ which associates a set of words to every node of the graph. By convention, the auxiliary words do not appear in any configuration.

There are two ways to change a configuration: by a splicing step or by a communication step. When a splicing step is used, each component $C(x)$ of the configuration C is changed according to the set of splicing rules S_x, whereby the words in set A_x are available for splicing. Formally, configuration C' is obtained in one splicing step from configuration C, written as $C \Rightarrow C'$, iff for all $x \in X_G$

$$C'(x) = \sigma_{S_x}(C(x) \cup A_x).$$

Since each word present in a node, as well as each auxiliary word, appears in an arbitrarily large number of identical copies, all possible splicings are assumed to be done in one splicing step. If the splicing step is defined as $C \Longrightarrow C'$, iff

$$C'(x) = S_x(C(x), A_x) \text{ for all } x \in X_G,$$

then all processors of Γ are called *restricted* and Γ itself is said to be restricted.

A communication step and the language accepted/decided by an ANSP are defined in the same way as those for ANEP. The definitions of the complexity classes defined on ANEPs can be straightforwardly carried over ANSPs. On the other hand, accepting networks of splicing processors with filtered connections (ANSPFC) are defined similarly to ANEPFCs.

4.1 Computational power of ANSP[FC]s

The main result in [37, 36] is:

Theorem 5.
1. Each recursively enumerable language L is accepted by a restricted ANSP of size 7.
2. Each **NP** *language L is accepted by a restricted ANSP of size 7 in polynomial time.*

We should point out that only the rules in the node input node depend on the language L, and the encoding that we use for its symbols while the parameters of the other nodes do not depend in any way on language L. If we allow all the parameters of the networks to depend on the given language, we have

Theorem 6. [30]
1. All recursively enumerable languages are accepted by ANSPs of size 2.
2. All languages in **NP** *can be accepted by ANSPs of size 3 working in polynomial time.*

Note that the ANSPs in the last theorem are not necessarily restricted. Since, by definition, ANSPs need at least two nodes to accept any non-trivial language, these results go a long way to settling this issue, although they do leave one problem unsolved: the efficient simulation of non-deterministic Turing machines by ANSPs with two nodes.

As far as the computational power of ANSPFCs is concerned, a complete characterization is reported in [14]:

Theorem 7.
1. A language is recursively enumerable if and only if it is accepted by a restricted ANSPFC of size 4.
2. A language is in **NP** *if and only if it is accepted by a restricted ANSPFC of size 4 in polynomial time.*

5 Problem Solving with [U]ANEP[FC]s/ANSP[FC]s

Although the results in the previous sections state that every problem in **NP** can be solved in polynomial time using different variants of accepting networks, the results are obtained by simulating a nondeterministic Turing machine; thus we still have to obtain a classic solution to a problem, and then translate it in terms of [U]ANEP[FC]s/ANSP[FC]s. To overcome this drawback, a series of papers discussed how [U]ANEP[FC]s and ANSP[FC]s can be viewed as problem solvers.

Recall that a possible correspondence between decision problems and languages can be made via an encoding function which transforms an instance of a given decision problem into a word (see, e.g., [26]). We say that a decision problem P is solved in time $\mathcal{O}(f(n))$ by [U]ANEP[FC]s/ANSP[FC]s if there exists a family $\mathcal{G}$ of [U]ANEP[FC]s/ANSP[FC]s such that the following conditions are satisfied:

1. The encoding function of any instance p of P with size n can be computed by a deterministic Turing machine in time $\mathcal{O}(f(n))$.
2. For each instance p of size n of the problem one can effectively construct, in time $\mathcal{O}(f(n))$, an [U]ANEP[FC]/ANSP[FC] $\Gamma(p) \in \mathcal{G}$ which decides, again in time $\mathcal{O}(f(n))$, the word encoding the given instance. This means that the word is decided if and only if the solution to the given instance of the problem is "YES". This effective construction is called an $\mathcal{O}(f(n))$ time solution to the problem.

If the [U]ANEP[FC]/ANSP[FC] $\Gamma \in \mathcal{G}$ constructed above decides the language of words encoding all instances of the same size n, then the construction of Γ is called a uniform solution. Intuitively, a solution is uniform if for problem size n, we can construct a unique [U]ANEP[FC]/ANSP[FC] that solves all instances of size n taking the (reasonable) encoding of instance as "input".

The paper [34] proposes using ANEPs to provide uniform linear time solutions to the 3-CNF-SAT and Hamiltonian Path; in [38] a uniform linear solution to the Vertex-Cover problem is proposed. And [23] proposes another uniform linear time solution to the Vertex-Cover problem, solved this time by ANEPFCs. Uniform linear time solutions to the SAT and Hamiltonian Path problems with ANSPs and ANSPFCs are discussed in [33].

6 Accepting Networks of Genetic Processors

The third case that we refer to in this work is the Accepting Networks of Genetic Processors (ANGP). Here, there are two sources of inspiration: the classical paradigm of Genetic Algorithms and Evolutionary Computation [43], and the models of Evolutionary or Splicing processors mentioned above. A genetic processor can perform one of the following two operations: (1) Mutation between symbols (here, the substitution operation in the evolutionary processors can be considered), and (2) Pure and massive crossover (which can be considered as the splicing operation by taking empty contexts). Observe that both operations were considered in the past as the main ingredients of genetic algorithms. Despite this, ANGP differs from classical Genetic Algorithms in two aspects: first, ANGP consists of a finite number of processors that run in parallel independently, so they should be considered as a full parallel scheme for genetic algorithms [1]; and second, the model is an acceptation model not an optimization one (like genetic algorithms). Nevertheless, ANGP could be modified to tackle optimization problems instead of acceptation ones.

For any alphabet V, the *mutation* rules take the form $a \to b$, with $a, b \in V$, and they can be applied over the string xay to produce the new string xby. The *crossover operation* is defined as follows: Let x and y be two strings, then $x \bowtie y = \{x_1y_2, y_1x_2 : x = x_1x_2 \text{ and } y = y_1y_2\}$. Observe that $x, y \in x \bowtie y$ given that we can take ε to be a part of x or y. In addition, the crossover operation can be extended over languages in the usual form.

A *genetic processor* over V is a tuple $(M_R, A, PI, FI, PO, FO, \alpha, \beta)$, where M_R is a finite set of mutation rules over V, A is a multiset of strings over V with a finite support and an arbitrary large number of copies of every string, $PI, FI \subset V^*$ are the input permitting/forbidding contexts, $PO, FO \subset V^*$ are the output permitting/forbidding contexts, $\alpha \in \{1, 2\}$ defines the working mode with the following values

- If $\alpha = 1$ the processor applies mutation rules
- If $\alpha = 2$ the processor applies crossover rules and $M_R = \emptyset$

and $\beta \in \{(s), (w)\}$ defines the type of the input/output filters of the processor. Here, s means the strong predicate $rc_s(\cdot; P, F)$ as defined in the evolutionary case, and w the weak predicate denoted by $rc_w(\cdot; P, F)$. Nevertheless, given that $P, F \subset V$, the previous predicates will be defined over the segments of a given string instead of its symbols.

An Accepting Network of Genetic Processors is defined as in the previous models of ANEPs and ANSPs. The acceptance criterion, the configuration of the network and the alternation between communication steps and *genetic* steps are defined as in the previous models.

With respect to the completeness of the ANGP model, we have the following result.

Theorem 8. [10] *Every recursively enumerable language can be accepted by an ANGP.*

The proof of the previous result is approached in a non-uniform manner. Hence, one can construct in polynomial time an ANGP that simulates the computation of an arbitrary Turing machine with an arbitrary input string (no matter its length). Given that the previous simulation works in polynomial time depending on the length of the input string (provided that we take into account the number of genetic and communication steps), the following result comes easily by simulating a nondeterministic Turing machine.

Theorem 9. [10] *Every language in* **NP** *can be accepted/decided in polynomial time by an ANGP.*

Observe that no results have been obtained to define the description complexity of this model. Nevertheless, a formal proof that 16 genetic processors are sufficient to generate any recursively enumerable language is provided in [11]. So, it is expected that further results of the descriptive complexity of ANGPs will be provided shortly.

7 Towards an Unifying Model

We have presented three different models of Accepting Networks of Bio-inspired processors. They have common characteristics and features that point to a model which can be formally defined. They share the following aspects and, probably, new models will be formulated in the near future:

1. A finite set of processors that apply operations over strings which have been inspired by biomolecular functions and operations in nature. The processors work with a multiset of strings.
2. A connection topology between processors in the form of a network.
3. A set of (input/output) filters which can be attached to the processors or to the connections.

A biologically inspired processor with filters, over an alphabet V, can be defined as the tuple (op, PI, FI, PO, FO), where op is a biologically inspired operation over strings and the rest of the elements have been defined in the evolutionary processors.

The following table shows some of the operations that we have defined in this study and others which can be used instead of the operations that have been defined previously.

insertion	Insert a symbol into a string
deletion	Delete a symbol from a string
substitution (mutation)	Substitute a symbol into a string
splicing	Splicing rules
crossover	Full massive splicing with empty context
hairpin completion	Hairpin completion from folded strings [15, 52]
superposition	Complementarity completion from double stranded strings [7]
loop and double loop recombination	DNA recombination based on gene assembly [54]
inversion, duplication and transposition	DNA fragments modification as operations over substrings [29, 18]

Table 1: Some operations which can be inserted into biologically inspired processors

Once we have introduced a generalization of previously defined processors, an Accepting Network of Bio-inspired Processors (ANBP) can be defined as the tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where the difference with respect to ANEPs is that the function N associates a biologically inspired processor to every vertex in the connection graph.

Here, we describe a new framework that should be studied in depth. In particular the following questions should be addressed:

- Some of the operations shown in Table 1, do not have computational completeness (i.e. they do not characterize recursively enumerable languages). We can combine some of these operations by inserting them into different processors. It is natural to ask whether computational completeness could be achieved for some combinations of operations, and what the minimal combination is to achieve it.
- Filtered connections have been proposed for ANEPs while other models consider only filtered processors. The transformation of filtered processors into filtered connections should be explored in the different combinations of operations. Furthermore, we could provide a pure hybrid network where different types of filters (connections or processors) work together.

References

1. Alba, E., Troya, J.M. (1999). A survey of parallel distributed genetic algorithms. *Complexity, 4(4)*, 31–52.
2. Alhazov, A., Rogozhin, Y. (2008). About Precise Characterization of Languages Generated by Hybrid Networks of Evolutionary Processors with One Node. it Computer Science Journal of Moldova 16(3), 364-376.
3. Alhazov, A., Csuhaj-Varjú, E., Martín-Vide, C., Rogozhin, Y. (2008). About Universal Hybrid Networks of Evolutionary Processors of Small Size. In *Language and Automata Theory and Applications (LATA 2008), LNCS 5196*, 28-39.
4. Alhazov, A., Martín-Vide, C., Truthe, B., Dassow, J., Rogozhin, Y. (2009). On Networks of Evolutionary Processors with Nodes of Two Types. *Fundam. Inform. 91(1)*, 1-15.
5. Alhazov, A., Bel Enguix, G., Rogozhin, Y. (2009). Obligatory Hybrid Networks of Evolutionary Processors. In *International Conference on Agents and Artificial Intelligence (ICAART 2009)*, 613-618.
6. Alhazov, A., Csuhaj-Varjú, E., Martín-Vide, C., Rogozhin, Y. (2009). On the Size of Computationally Complete Hybrid Networks of Evolutionary Processors. *Theoretical Computer Science 410*, 3188–3197.
7. Bottoni, P., Labella, A., Manca, V., Mitrana, V. (2004). Superposition Based on WatsonÛCrick-Like Complementarity *Theory of Computing Systems 39*, 503-524.
8. Bottoni, P., Labella, A., Manea, F., Mitrana, V., Sempere, J.M. (2009). Filter Position in Networks of Evolutionary Processors Does Not Matter: A Direct Proof. In *International Meeting on DNA Computing and Molecular Programming (DNA 15), LNCS 5877*, 1–11.
9. Bottoni, P., Labella, A., Manea, F., Mitrana, V., Petre, I., Sempere, J.M. (2010). Complexity-Preserving Simulations Among Three Variants of Accepting Networks of Evolutionary Processors *Natural Computing*, in press.
10. Campos, M., Sempere, J.M. (2011). Accepting Networks of Genetic Processors are computationally complete. (*submitted*)
11. Campos, M., Sempere, J.M. (2011). Descriptive Complexity of Generating Networks of Genetic Processors. (*submitted*)
12. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M. (2003). Networks of Evolutionary Processors. *Acta Informatica 39*, 517–529.
13. Castellanos, J., Leupold, P., Mitrana, V. (2005). On the Size Complexity of Hybrid Networks of Evolutionary Processors. *Theoretical Computer Science 330 (2)*, 205–220.
14. Castellanos, J., Manea, M., Mingo López, L.F., Mitrana, V. (2007). Accepting Networks of Splicing Processors with Filtered Connections. In *Machines, Computations, and Universality (MCU 2007), LNCS 4664*, 218–229.
15. Cheptea, D., Martín-Vide, C., V. Mitrana, V. (2006) A new operation on words suggested by DNA biochemistry: Hairpin completion, in: Proc. Transgressive Computing, 216–228.

16. Csuhaj-Varjú, E., Kari, L., Păun, G. (1996). Test Tube Distributed Systems Based on Splicing. *Computers and AI, 15*, 211–232.
17. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V. (2005). Hybrid Networks of Evolutionary Processors are Computationally Complete. *Acta Informatica 41*, 257–272.
18. Dassow, J., Mitrana, V., Salomaa, A. (2002). Operations and language generating devices suggested by the genome evolution *Theoretical Computer Science 270*, 701–738.
19. Dassow, J., Truthe, B. (2007). On the Power of Networks of Evolutionary Processors. In *Machines, Computations, and Universality (MCU 2007), LNCS 4667*, 158–169.
20. Dassow, J., Mitrana, V. (2008). Accepting Networks of Non-Inserting Evolutionary Processors, In *Proceedings of NCGT 2008: Workshop on Natural Computing and Graph Transformations*, 29–42.
21. Dassow, J., Mitrana, V., Truthe, B. (2008). The Role of Evolutionary Operations in Accepting Networks of Evolutionary Processors. Submitted.
22. Drăgoi, C., Manea, F. (2008). On the Descriptional Complexity of Accepting Networks of Evolutionary Processors with Filtered Connections. *International Journal of Foundations of Computer Science, 19:5*, 1113–1132.
23. Drăgoi, C., Manea, F., Mitrana, V. (2007). Accepting Networks of Evolutionary Processors With Filtered Connections. *Journal of Universal Computer Science, 13:11*, 1598–1614.
24. Errico, L., Jesshope, C. (1994). Towards a New Architecture for Symbolic Processing, In *Artificial Intelligence and Information-Control Systems of Robots '94*, 31–40.
25. Fahlman, S. E., Hinton, G.E., Seijnowski, T.J. (1983). Massively Parallel Architectures for AI: NETL, THISTLE and Boltzmann Machines, In *Proc. of the National Conference on Artificial Intelligence*, 109–113.
26. Garey, M., Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*, San Francisco, CA: W. H. Freeman.
27. Hartmanis, J., Stearns, R.E. (1965). On the Computational Complexity of Algorithms, *Trans. Amer. Math. Soc.*, 117, 533–546.
28. Hillis, W.D. (1979). *The Connection Machine*. MIT Press, Cambridge.
29. Leupold, P., Mitrana, V., Sempere, J.M. (2004). Formal Languages Arising from Gene Repeated Duplication, In *Aspects of Molecular Computing LNCS 2950*, 297–308.
30. Loos, R., Manea, F., Mitrana, V. (2009). On Small, Reduced, and Fast Universal Accepting Networks of Splicing Processors, *Theoretical Computer Science 410:4-5*, 417–425.
31. Loos, R., Manea, F., Mitrana, V. (2009). Small Universal Accepting Networks of Evolutionary Processors with Filtered Connections, In *Proceedings of the Descriptional Complexity of Formal Systems Workshop, EPTCS 3*, 173–183.

32. Loos, R., Manea, F., Mitrana, V. (2009). Small Universal Accepting Networks of Evolutionary Processors, submitted.
33. Manea, F., Martín-Vide, C., Mitrana, V. (2005). Accepting Networks of Splicing Processors. In *Computability in Europe (CiE 2005), LNCS 3526*, 300–309.
34. Manea, F., Martín-Vide, C., Mitrana, V. (2005). Solving 3CNF-SAT and HPP in Linear Time Using WWW, In *Machines, Computations and Universality, LNCS 3354*, 269–280.
35. Manea, F., Martín-Vide, C., Mitrana, V. (2006). On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors, in *Proceedings of the First International Workshop on Developments in Computational Models, ENTCS 135*, 95–105.
36. Manea, F., Martín-Vide, C., Mitrana, M. (2006). All NP-Problems Can Be Solved in Polynomial Time by Accepting Networks of Splicing Processors of Constant Size. *International Meeting on DNA Computing, DNA12, LNCS 4287*, 47–57.
37. Manea, F., Martín-Vide, C., Mitrana, V. (2007). Accepting Networks of Splicing Processors: Complexity Results, *Theoretical Computer Science, 371:1-2*, 72–82.
38. Manea, F., Martín-Vide, C., Mitrana, V. (2007). On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors, *Mathematical Structures in Computer Science, 17:4*, 753–771.
39. Manea, F., Mitrana, V. (2007). All NP-problems Can Be Solved in Polynomial Time by Accepting Hybrid Networks of Evolutionary Processors of Constant Size, *Information Processing Letters, 103:3*, 112 – 118.
40. Manea, F., Margenstern, M., Mitrana, V., Pérez-Jiménez, M. J. (2008). A New Characterization of NP, P, and PSPACE With Accepting Hybrid Networks of Evolutionary Processors, in press *Theory of Computing Systems*, doi:10.1007/s00224-008-9124-z.
41. Margenstern, M., Rogozhin, Y. (2001). Time-varying Distributed H Systems of Degree 1 Generate All Recursively Enumerable Languages. In *Words, Semigroups, and Transductions* World Scientific Publishing, Singapore, 329–340.
42. Martín-Vide, C., Mitrana, V. (2005). Networks of Evolutionary Processors: Results and Perspectives, In *Molecular Computational Models: Unconventional Approaches*, 78–114.
43. Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs, Springer.
44. Minsky, M.L., Size and Structure of Universal Turing Machines Using Tag Systems. In: *Recursive Function Theory, Symp. in Pure Mathematics 5*, 229–238.
45. Papadimitriou C.H. (1994). *Computational Complexity*, Addison-Wesley.
46. Păun, A. (1999). On Time-varying H Systems. *Bulletin of the EATCS, 67*, 157–164.
47. Păun, G., Sântean, L. (1989). Parallel Communicating Grammar Systems: The Regular Case, *Annals of University of Bucharest, Ser. Matematica-Informatica 38*, 55–63.

48. Păun, G. (1998). Distributed Architectures in DNA Computing Based on Splicing: Limiting the Size of Components. In *Unconventional Models of Computation*, Springer-Verlag, Berlin, 323–335.
49. Păun, G. (1996). Regular Extended H Systems are Computationally Universal. *J. Automata, languages, Combinatorics, 1(1)*, 27–36.
50. Păun, G. (1997). DNA Computing; Distributed Splicing Systems. In *Structures in Logic and Computer Science, LNCS 1261*, Springer-Verlag, Berlin, 351–370.
51. Păun, G. (2000). Computing with Membranes, *Journal of Computer and System Sciences 61*, 108–143.
52. Păun G., Rozenberg, G., Yokomori, T. (2001). Hairpin languages. *International Journal of Foundations of Computer Science 12*, 837Ű-847.
53. Post, E.L. (1943). Formal Reductions of the General Combinatorial Decision Problem, *Amer. J. Math. 65*, 197–215.
54. Prescott, D.M., Ehrenfeucht, A., Rozenberg, G. (2001). Molecular operations for DNA processing in hypotrichous ciliates. *European Journal of Protistology 37*, 241–260.
55. Rogozhin, Y. (1996). Small Universal Turing Machines, *Theoretical Computer Science 168*, 215-240.
56. Rozenberg, G., Salomaa, A., eds. (1997). *Handbook of Formal Languages*, vol. I-III, Springer-Verlag, Berlin.
57. Sankoff, D., et al. (1992). Gene Order Comparisons for Phylogenetic Inference: Evolution of the Mitochondrial Genome, In *Proceedings of the National Academy of Sciences of the United States of America 89*, 6575–6579.