ACCEPTING NETWORKS OF EVOLUTIONARY PROCESSORS: COMPLEXITY ASPECTS

Recent Results and New Challenges

Florin Manea and Victor Mitrana
Faculty of Mathematics and Computer Science, University of Bucharest, Academiei 14, 010014, Bucharest, Romania

Keywords: Theory of computation, Computational Complexity, Complexity classes, Evolutionary processor.

Abstract: In this paper we survey some results reported so far, for the new computational model of Accepting Networks of Evolutionary Processors (ANEPs), in the area of computational and descriptional complexity. First we give the definitions of the computational model, and its variants, then we define several ANEP complexity classes, and, further, we show how some classical complexity classes, defined for Turing Machines, can be characterized in this framework. After this, we briefly show how ANEPs can be used to solve efficiently NP-complete problems. Finally, we discuss a list of open problems and further directions of research which appear interesting to us.

1 INTRODUCTION

The origin of the networks of evolutionary processors (NEPs for short) is twofold. A basic architecture for parallel and distributed symbolic processing, related to the Connection Machine (Hillis, 1985) as well as the Logic Flow paradigm (Errico and Jesshope, 1994), consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then local data becomes a mobile agent 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 to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see (Fahlman et al., 1983; Hillis, 1985).

On the other hand, in (Csuhaj-Varjú and Mitrana, 2000) we consider a computing model inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactical level. Cells are represented by strings which describe their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of strings, where each string represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on strings. Only those cells are accepted as surviving (correct) ones which are represented by a string in a given set of strings, called the genotype space of the species. This feature parallels with the natural process of evolution. Similar ideas may be met in other bio-inspired models like *membrane systems* (Păun, 2000), *evolutionary systems* (Csuhaj-Varjú and Mitrana, 2000), or models from Distributed Computing area like *parallel communicating grammar systems* (Păun and Sântean, 1989), *networks of parallel language processors* (Csuhaj-Varjú and Salomaa, 1997).

In (Castellanos et al., 2001) we modify this concept (considered from a formal language theory point of view in (Csuhaj-Varjú and Salomaa, 1997)) in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a processor which is able to 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 having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of strings (each string

Manea F. and Mitrana V. (2009).
ACCEPTING NETWORKS OF EVOLUTIONARY PROCESSORS: COMPLEXITY ASPECTS - Recent Results and New Challenges.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 597-604
DOI: 10.5220/0001796305970604

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. The work (Martín-Vide and Mitrana, 2005) is an early survey.

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 *string (word)* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by ε. The length of a string x is denoted by $|x|$ while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$. For the basic details regarding Turing machines and complexity classes we refer to (Garey and Johnson, 1979).

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 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 string $w \in V^*$, 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 as 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 of applying a deletion or insertion rule to a string, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the string, 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 string w and the language L by:

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

respectively.

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

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

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 requires that no forbidding symbol is 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:

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

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 one can see, 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 $PO \cap FO = \emptyset$).

We denote the set of evolutionary processors over V by EP_V. Clearly, the evolutionary processor described here is a mathematical concept similar to that of an evolutionary algorithm, both being inspired from the Darwinian evolution. As we mentioned above, the rewriting operations we have considered might be interpreted as mutations and the filtering process described above might be viewed as a selection process. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration (Sankoff et al., 1992).

However, another type of processor based on recombination only, called splicing processor has been considered as well in a series of works (Manea et al., 2007a; Loos et al., 2008) and the references thereof.

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

- V and U are the input and network alphabet, 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 with each node $x \in X_G$ 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 strings 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 string w can pass the input (resp. output) filter of x. Moreover, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of strings 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.

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 like *rings*, *stars*, *grids*, etc. We focus here on *complete* ANEPs i.e., ANEPs having a complete underlying graph.

A model closely related to that of ANEPs, introduced in (Drăgoi et al., 2007) and further studied in (Drăgoi and Manea, 2008), is that of Accepting Networks of Evolutionary Processors with Filttering Connections (ANEPFCs for short). An ANEPFC may be viewed as an ANEP where the filters are shifted from the nodes on the edges. Therefore, instead of having a filter at both ends of an edge on each direction, there is only one filter disregarding the direction.

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 associate with them filters that do not allow any string to pass. Therefore, for the sake of simplicity, the ANEPFCs we discuss in this paper have underlying graphs with useful edges only (note that such a simplification is not always possible for ANEPs).

A *configuration* of an ANEP/ANEPFC Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of strings with every node of the graph. A configuration may be understood as the sets of strings which are present in any node at a given moment. Given a string $w \in V^*$, 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 changing by an evolutionary step, for both ANEPs and ANEPFCs, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_G$.

When changing by a communication step, in the case of ANEPs, each node processor $x \in X_G$ sends one copy of each word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x providing that they can pass its input filter. Formally, we say that the 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 eliminated from that node. If they cannot pass the input filter of any node, they are lost.

Differently, when changing by 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 string is lost.

Let Γ be an ANEP (ANEPFC), 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 the 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 strings 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 ANEP/ANEPFC Γ is $L_a(\Gamma) = \{w \in V^* \mid$ the computation of Γ on w is an accepting one$\}$. We say that an ANEP/ANEPFC 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.

The ANEP computing model was modified in (Manea, 2005) to obtain Timed Accepting Networks of Evolutionary Processors (TANEP for short). Such a TANEP is a triple $\mathcal{T} = (\Gamma, f, b)$, where $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$ is an ANEP, $f : V^* \to \mathbf{N}$ is a Turing computable function, called *clock*, and $b \in 0, 1$ is a bit called the *accepting-mode bit*.

In this setting, the computation of a TANEP $\mathcal{T} = (\Gamma, f, b)$ on the input word w is the (finite) sequence of configurations of the ANEP Γ: $C_0^{(w)}, C_1^{(w)}, \ldots, C_{f(w)}^{(w)}$. The language accepted by $\mathcal{T}$ is defined as:

- if $b = 1$ then: $L(\mathcal{T}) = \{w \in V^* \mid C_{f(w)}^{(w)}(x_O) \neq \emptyset\}$
- if $b = 0$ then: $L(\mathcal{T}) = \{w \in V^* \mid C_{f(w)}^{(w)}(x_O) = \emptyset\}$

Intuitively we may think that a TANEP $\mathcal{T} = (\Gamma, f, b)$ is a triple that consists in an ANEP, a Turing Machine and a bit. For an input string w we first compute $f(w)$ on the tape of the Turing Machine (by this we mean that on the tape there will exist $f(w)$ elements of 1, while the rest are blanks). Then we begin to use the ANEP Γ, and at each evolutionary or communication step of the network we delete an 1 from the tape of the Turing Machine. We stop when no 1 is found on the tape. Finally, we check the accepting-mode bit, and, according to its value and the emptyness of $C_{f(w)}^{(w)}(x_O)$, we decide whether w is accepted or not.

Further, we define some computational complexity measures by using ANEP/ANEPFC as the computing model. To this aim we consider a ANEP/ANEPFC Γ with the input alphabet V that halts on every input. The *time complexity* of the halting computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in V^*, |x| = n\}.$$

In other words, $Time_\Gamma(n)$ delivers the maximal number of computational steps done by Γ on an input word of length n.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ and $\mathcal{X} \in \{$ANEP, ANEPFC$\}$ we define:

$$\mathbf{Time}_\mathcal{X}(f(n)) = \{L \mid \text{there exists an network of type } \mathcal{X}, \Gamma \text{ which decides } L, \text{ and } n_0 \text{ such that } \forall n \geq n_0 (Time_\Gamma(n) \leq f(n))\}.$$

Moreover, we write $\mathbf{PTime}_\mathcal{X} = \bigcup_{k \geq 0} \mathbf{Time}_\mathcal{X}(n^k)$.

The *space complexity* of the halting computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Space_\Gamma(x)$ and is defined by the relation::

$$Space_\Gamma(x) = \max_{i \in \{1,\ldots,m\}} (\max_{z \in X_G} card(C_i^{(x)}(z))).$$

The space complexity of Γ is the function from $\mathbf{N}$ to $\mathbf{N}$,

$$Space_\Gamma(n) = \max\{Space_\Gamma(x) \mid x \in V^*, |x| = n\}.$$

Thus $Space_\Gamma(n)$ returns the maximal number of distinct words existing in a node of Γ during a computation on an input word of length n.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ and $\mathcal{X} \in \{$ANEP, ANEPFC$\}$we define

$$\mathbf{Space}_\mathcal{X}(f(n)) = \{L \mid \text{there exists a network of type } \mathcal{X}, \Gamma \text{ which decides } L, \text{ and } n_0 \text{ such that } \forall n \geq n_0 (Space_\Gamma(n) \leq f(n))\}.$$

Moreover, we write $\mathbf{PSpace}_\mathcal{X} = \bigcup_{k \geq 0} \mathbf{Space}_\mathcal{X}(n^k)$.

The *length complexity* of the halting computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Length_\Gamma(x)$ and is defined by the relation:

$$Length_\Gamma(x) = \max_{w \in C_i^{(x)}(z), i \in \{1,\ldots,m\}, z \in X_G} |w|.$$

The length complexity of Γ is the function from $\mathbf{N}$ to $\mathbf{N}$,

$$Length_\Gamma(n) = \max\{Length_\Gamma(x) \mid x \in V^*, |x| = n\}.$$

Unlike the *Space* measure, $Length_\Gamma(n)$ computes the length of the longest word existing in a node of Γ during a computation on an input word of length n.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ and $\mathcal{X} \in \{$ANEP, ANEPFC$\}$ we define $\mathbf{Length}_\mathcal{X}(f(n)) = \{L \mid$ there exists an network of type $\mathcal{X}, \Gamma$ which decides L and n_0 such that $\forall n \geq n_0 (Length_\Gamma(n) \leq f(n))\}$ Moreover, we write $\mathbf{PLength}_\mathcal{X} = \bigcup_{k \geq 0} \mathbf{Length}_\mathcal{X}(n^k)$.

In the case of a TANEP $\mathcal{T} = (\Gamma, f, b)$ the time complexity definitions are the following: for the word $x \in V^*$ we define the time complexity of the computation on x as the number of steps that the TANEP makes having the word x as input, $Time_{\mathcal{T}}(x) = f(x)$. Consequently, we define the time complexity of $\mathcal{T}$ as a partial function from **N** to **N**, that verifies: $Time_{\mathcal{T}}(n) = \max\{f(x) \mid x \in L(\mathcal{T}), |x| = n\}$. For a function $g : \mathbf{N} \longrightarrow \mathbf{N}$ we define:

$$\mathbf{Time}_{TANEP}(g(n)) = \{L \mid L = L(\mathcal{T}) \text{for a TANEP } \mathcal{T} = (\Gamma, f, 1) \text{ with } Time_{\mathcal{T}}(n) \leq g(n) \text{ for some } n \geq n_0\}.$$

Moreover, we write $\mathbf{PTime}_{TANEP} = \bigcup_{k \geq 0} \mathbf{Time}_{TANEP}(n^k)$.

Note that the above definitions were given for TANEPs with the accepting-mode bit set to 1. Similar definitions are given for the case when the accepting-mode bit set to 0. For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define, as in the former case:

$$\mathbf{CoTime}_{TANEP}(g(n)) = \{L \mid L = L(\mathcal{T}) \text{for a TANEP } \mathcal{T} = (\Gamma, f, 0) \text{ with } Time_{\mathcal{T}}(n) \leq g(n) \text{ for some } n \geq n_0\}.$$

We define $\mathbf{CoPTime}_{TANEP} = \bigcup_{k \geq 0} \mathbf{CoTime}_{TANEP}(n^k)$.

3 COMPLEXITY RESULTS

The main result obtained so far states the fact that non-deterministic Turing machines can be simulated efficiently by ANEPs:

Theorem 1.
1. *(Manea et al., 2008; Manea et al., 2007b) For every nondeterministic single-tape Turing machine M, with working alphabet U, deciding a language L, there exists an ANEP Γ, of size $5|U|+8$, deciding the same language L. Moreover, if M works within $f(n)$ time, then $Time_\Gamma(n) \in O(f(n))$, and if M works within $f(n)$ space, then $Space_\Gamma(n) \in O(\max\{n, f(n)\})$..*
2. *(Drăgoi and Manea, 2008) For every nondeterministic single-tape Turing machine M, with working alphabet U, deciding a language L, there exists an ANEPFC Γ, of size $2|U|+12$, deciding the same language L. Moreover, if M works within $f(n)$ time, then $Time_\Gamma(n) \in O(f(n))$, and if M works within $f(n)$ space, then $Length_\Gamma(n) \in O(\max\{n, f(n)\})$.*

Basically the both results stated in this Theorem are based on the following approach: we construct an ANEP/ANEPFC Γ that simulates the computation of the Turing machine M on an input word w such that each move made by the Turing machine M is simulated by Γ in a constant number of steps of the ANEP/ANEPFC; moreover, Γ halts and accepts w if and only if M does this. More precisely, Γ obtains in parallel all the IDs that M may reach in one step from its previous ID in a constant number evolutionary and communication steps. Once M reaches a final ID, a word enters the output node of Γ. In the case when all computations of M on w stop but M does not accept, Γ passes through two identical consecutive configurations, hence it halts without accepting. Otherwise, both M and Γ continue their computations forever. Thus, if $L \in NTIME(f(n))$, then $Time_\Gamma(n) \in O(f(n))$. Since all the strings processed by the network have their length bounded by the length of an ID of M plus a constant number of symbols, it also results that if $L \in NSPACE(f(n))$, then $Length_\Gamma(n) \in O(f(n))$. Note that in the case of Turing machines, the complexity classe are those defined for single tapes machines.

The reversal of Theorem 1 holds as well:

Theorem 2. *(Manea et al., 2008; Drăgoi et al., 2007) For any ANEP/ANEPFC Γ accepting the language L, there exists a single-tape Turing machine M accepting L. Moreover, M can be constructed such that either it accepts in $O((Time_\Gamma(n))^2)$ computational time or in $O(Length_\Gamma(n))$ space.*

The proof of this Theorem is quite straightforward: the Turing Machine chooses and simulates (non-deterministically) a possible succession of processing and communication steps of Γ on the input word. If this succession of steps leads to a string that enters in the output node, then the input word is accepted.

A consequence of Theorems 1 and 2 is the following:

Theorem 3. *(Manea et al., 2008; Drăgoi et al., 2007)*
1. $\mathbf{NP} = \mathbf{PTime}_{ANEP} = \mathbf{PTime}_{ANEPFC}$.
2. $\mathbf{PSPACE} = \mathbf{PLength}_{ANEP} = \mathbf{PLength}_{ANEPFC}$.

These results were improved from the size complexity point of view: **NP** equals the class of languages accepted in polynomial time by ANEPs with 24 nodes and with the class of languages accepted in polynomial time by ANEPFCs with 26 nodes (see (Manea and Mitrana, 2007; Drăgoi and Manea, 2008)).

Finally one can obtain a characterization of **P**, also based on the result of Theorem 1:

Theorem 4. *(Manea et al., 2008) A language $L \in \mathbf{P}$ iff L is decided by an ANEP/ANEPFC Γ such that there exist two polynomials P, Q with $Space_\Gamma(n) \leq P(n)$ and $Time_\Gamma(n) \leq Q(n)$.*

It is worth mentioning that the last theorem does not say that the inclusion $\mathbf{PSpace}_{\mathcal{X}} \cap \mathbf{PTime}_{\mathcal{X}} \subseteq \mathbf{P}$

holds, for some $\mathcal{X} \in \{ANEP, ANEPFC\}$. The following facts are not hard to follow: we proved in Theorem 3 that every NP language, hence the NP-complete language 3-CNF-SAT, is in $\mathbf{PTime}_{\mathcal{X}}$; but, it is easy to see that 3-CNF-SAT can be decided also by a deterministic Turing Machine, working in exponential time and polynomial space. By Proposition 1, such a machine can be simulated by an ANEP/ANEPFC that uses polynomial space (but exponential time as well). This shows that 3-CNF-SAT is in $\mathbf{PTime}_{\mathcal{X}} \cap \mathbf{PSpace}_{\mathcal{X}}$, but it is not in **P**, unless $\mathbf{P} = \mathbf{NP}$.

TANEPs offer us the possibility to characterize uniformly both **NP** and **CoNP**:

Theorem 5. *(Manea, 2005)* $\mathbf{PTime}_{TANEP} = \mathbf{NP}$ *and* $\mathbf{CoPTime}_{TANEP} = \mathbf{CoNP}$.

As explained already, we can choose and simulate non-deterministically with a Turing Machine M each one of the possible succesion of processing and communication steps applied on the input string by the ANEP component of a TANEP $\mathcal{T} = (\Gamma, f, 1)$. Just that in this case we are interested only in the first $f(x)$ steps of the ANEP, and there exist a polynomial g such that $f(x) \leq g(|x|)$, for every possible input string x. From these follows that M works in polynomial time, and $\mathbf{PTime}_{TAHNEP} \subseteq \mathbf{NP}$. To prove that $\mathbf{NP} \subseteq \mathbf{PTime}_{TAHNEP}$ we also make use of Theorem 1: for a language $L \in \mathbf{NP}$ there exists an ANEP Γ and a polynomial g such that $x \in L$ if and only if $x \in L(\Gamma)$ and $Time_{\Gamma}(x) \leq g(|x|)$. From this it follows that the TANEP $\mathcal{T} = (\Gamma, f, 1)$, where $f(x) = g(|x|)$, accepts L. A similar proves the second part of the theorem, for TANEPs with accepting bit 0.

Theorems 5 provides a common framework for solving both problems from **NP** and from **CoNP**. For example, suppose that we want to solve the membership problem for a language L.

- If $L \in \mathbf{NP}$, using the proof of Theorems 1, we can construct a polynomial TANEP $\mathcal{T} = (\Gamma, f, 1)$ that accepts L.
- If $L \in \mathbf{CoNP}$, it results that $\mathbf{Co}L \in \mathbf{NP}$, and using the proofs of Theorems 1, we can construct a polynomial TANEP $\mathcal{T} = (\Gamma, f, 1)$ that accepts $\mathbf{Co}L$. We obtain that $(\Gamma, f, 0)$ accepts L.

Thus, Theorem 5 proves that the languages (the decision problems) that are efficiently recognized (solved) by the TANEPs (with both 0 and 1 as possible values for the accepting-mode bit) are those from $\mathbf{NP} \cup \mathbf{CoNP}$.

4 PROBLEM SOLVING

Recall that a possible correspondence between decision problems and languages can be done via an encoding function which transforms an instance of a given decision problem into a word, see, e.g., (Garey and Johnson, 1979). We say that a decision problem P is solved in time $O(f(n))$ by ANEPs/ANEPFCs if there exists a family $\mathcal{G}$ of ANEPs/ANEPFCs such that the following conditions are satisfied:

1. The encoding function of any instance p of P having size n can be computed by a deterministic Turing machine in time $O(f(n))$.
2. For each instance p of size n of the problem one can effectively construct, in time $O(f(n))$, an ANEP/ANEPFC $\Gamma(p) \in \mathcal{G}$ which decides, again in time $O(f(n))$, 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 $O(f(n))$ time solution to the considered problem.

If an ANEP/ANEPFC $\Gamma \in \mathcal{G}$ 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 ANEP/ANEPFC solving all instances of size n taking the (reasonable) encoding of instance as "input".

In (Manea et al., 2005) we propose a linear time solution for the 3-CNF-SAT and Hamiltonian Path problems, using ANEPs; also, in (Manea et al., 2007b) we propose a linear solution for the Vertex-Cover problem. In (Drăgoi et al., 2007) we propose another linear time solution for the Vertex-Cover problem, solved this time by ANEPFCs.

5 CHALLENGES

We presented new characterizations of some well-known complexity classes like **P, NP, co-NP, PSPACE** based on ANEPs and ANEPFCs. We also got upper bounds for the size of these networks. However, we do not know how close to the optimal size these bounds are. In our view, a comparison with other computational models might lead to better bounds.

Although we presented a characterization of **PSPACE** in terms of a complexity measure, namely *Length*, defined for ANEPs and ANEPFCs, this measure is rather artificial as it can never be smaller than the length of the input word. We consider that another

measure able to capture in a better way the similarity to the space measure defined for Turing machines is needed. Such a measure might shed a new light on the characterizations reported here.

On the other hand, the measure *Space* counts the maximum number of words existing in a node at a given step of a computation. This measure might also be useful though it seems to be less important from a biological point of view as an exponential number of DNA molecules can be produced by a linear number of Polymerase Chain Reaction (PCR) steps. One may remark that a limitation on the *Space* complexity of a computation may be translated as a limitation of the intrinsic power of this computing model to simulate by massive parallelism the nondeterminism of sequential machines. Another direction of research that appears to be of interest is the exact role filters, evolutionary operations, and underlaying structures play with respect to the computational power of ANEPs as well as their complexity. A first step was done in (Dassow and Mitrana, 2008), where ANEPs without insertion nodes were considered. An exhaustive study in this direction is under way.

A very preliminary work regarding the role of filters is (Dassow et al., 2006), where *generating* NEPs without filters are investigated. However, this work which reports only partial results is devoted to an extreme case for the generating model. Several variants in between might also be considered.

All the results presented here are essentially based on simulations of Turing machines. This is actually valid for almost all bio-inspired computational models. Even the universal ANEPs are obtained via simulations of Turing machines. In some sense, these simulations are not quite natural as all the bio-inspired models are mainly based on a possible huge parallelism while Turing machine is a sequential model. Therefore, direct simulations of parallel models as well as universal ANEPs derived directly from ANEPs are of a definite interest.

Last but not least, our presentation was not concern of practical matters regarding the possible biological or electronic implementation of these networks. There were reported some simulations on different computers under different softwares, see, e.g., (Gómez, 2008). Also some preliminary works on designing electronic components that could implement some aspects of ANEPs are under way.

REFERENCES

Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J. (2001) Solving NP-complete Problems with Networks of Evolutionary Processors, In *International Work-Conference on Artificial and Natural Neural Networks (IWANN 2001), LNCS 2084*, 621–628. Springer.

Csuhaj-Varjú, E. and Salomaa, A. (1997) Networks of Parallel Language Processors. In *New Trends in Formal Languages, LNCS 1218*, 299 - 318. Springer.

Csuhaj-Varjú, E., Mitrana, V. (2000). Evolutionary Systems: A Language Generating Device Inspired by Evolving Communities of Cells. *Acta Informatica, 36*, 913 – 926. Springer.

Dassow, J. and Mitrana, V. (2008). Accepting Networks of Non-Inserting Evolutionary Processors, In *Proceedings of NCGT 2008: Workshop on Natural Computing and Graph Transformations*, 29–42.

Dassow, J., Martin-Vide, C., Mitrana, V. (2006). Free Generating Hybrid Networks of Evolutionary Processors, In *Formal Models, Languages and Applications Series in Machine Perception and Artificial Intelligence 66*, 65–78. World Scientific.

Drăgoi, C. and 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. World Scientific.

Drăgoi, C., Manea, F., Mitrana, V. (2007). Accepting Networks of Evolutionary Processors With Filtered Connections. *Journal of Universal Computer Science, 13:11*, 1598 – 1614. Springer.

Errico, L., and Jesshope, C. (1994). Towards a New Architecture for Symbolic Processing, In *Artificial Intelligence and Information-Control Systems of Robots '94*, 31–40. World Scientific.

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. AAAI Press.

Garey, M., and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*, San Francisco, CA: W. H. Freeman.

Gómez Blas, N. (2008) *Redes de Procesadores Evolutivos: Autoaprendizaje de Filtros en las Conexiones*, *PhD Thesis*, Politechnical University of Madrid. (in Spanish).

Hillis, W.D. (1979). *The Connection Machine*. MIT Press, Cambridge.

Loos, R., Manea, F., Mitrana, V. (2008) On Small, Reduced, and Fast Universal Accepting Networks of Splicing Processors, in press *Theoretical Computer Science*, doi:10.1016/j.tcs.2008.09.048. Elsevier.

Manea, F. (2005). Timed Accepting Hybrid Networks of Evolutionary Processors, In *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, LNCS 3562*, 122 – 132. Springer.

Manea, F., Martin-Vide, C., Mitrana, V. (2005). Solving 3CNF-SAT and HPP in Linear Time Using WWW, In *Machines, Computations and Universality, LNCS 3354*, 269 – 280. Springer.

Manea, F., Martin-Vide, C., Mitrana, V. (2007) Accepting Networks of Splicing Processors: Complexity Results, *Theoretical Computer Science, 371:1-2*, 72–82. Elsevier.

Manea, F., Martin-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. Cambridge University Press.

Manea, F., and 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. Elsevier.

Manea, F., Margenstern, M., Mitrana, V., Perez-Jimenez, 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. Springer.

Martín-Vide, C. and Mitrana, V. (2005) Networks of Evolutionary Processors: Results and Perspectives, In *Molecular Computational Models: Unconventional Approaches*, 78-114. Idea Group Publishing.

Păun, G. and Sântean, L. (1989) Parallel Communicating Grammar Systems: The Regular Case, *Annals of University of Bucharest, Ser. Matematica-Informatica 38*, 55 - 63.

Păun, G. (2000) Computing with Membranes, *Journal of Computer and System Sciences 61*, 108 - 143. ACM Press.

D. Sankoff 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.