Journal of Universal Computer Science, vol. 13, no. 11 (2007), 1598-1614
submitted: 10/9/07, accepted: 19/11/07, appeared: 28/11/07 © *J.UCS*

Accepting Networks of Evolutionary Processors with Filtered Connections[1,2]

Cezara Drăgoi
(LIAFA, Universite Paris Diderot - Paris 7
Case 7014, 75205 Paris Cedex 13, France
and
Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 010014, Bucharest, Romania
E-mail: dragoi.cezara@gmail.com)

Florin Manea[3]
(Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 010014, Bucharest, Romania
E-mail: flmanea@gmail.com)

Victor Mitrana
(Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 010014, Bucharest, Romania
and
Research Group in Mathematical Linguistics, Rovira i Virgili University
Pça. Imperial Tarraco 1, 43005, Tarragona, Spain
E-mail: mitrana@fmi.unibuc.ro)

Abstract: In this paper we simplify a recent model of computation considered in [Margenstern et al. 2005], namely accepting network of evolutionary processors, by moving the filters from the nodes to the edges. Each edge is viewed as a two-way channel such that input and output filters, respectively, of the two nodes connected by the edge coincide. Thus, the possibility of controlling the computation in such networks seems to be diminished. In spite of this observation these simplified networks have the same computational power as accepting networks of evolutionary processors, that is they are computationally complete. As a consequence, we propose characterizations of two complexity classes, namely **NP** and **PSPACE**, in terms of accepting networks of evolutionary processors with filtered connections.

Key Words: Evolutionary processor, network of evolutionary processors, Turing machine, complexity class.

Category: F.1.1, F.1.3, F.4.3

[1] C. S. Calude, G. Stefanescu, and M. Zimand (eds.). *Combinatorics and Related Areas. A Collection of Papers in Honour of the 65th Birthday of Ioan Tomescu.*

[2] The authors acknowledge partial support from the Romanian Ministry of Education and Research (PN-II Program, Project *GlobalComp - Models, semantics, logics and technologies for global computing*).

[3] The work of this author was also partially supported by the Research Grant no. ET75/2005 of the Romanian National Authority for Scientific Research.

1 Introduction

The origin of networks of evolutionary processors (NEP for short) is a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [Hillis 1985] as well as the Logic Flow paradigm presented in [Errico and Jesshope 1994], which 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. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g., [Fahlman et.al. 1983, Hillis 1985].

In a series of papers (see [Martín-Vide and Mitrana 2005] for a survey) one considers that 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 words (each word appears in an arbitrarily large number of copies), and all the copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, the computational process just described is not exactly an evolutionary process in the Darwinian sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process 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].

We want to stress from the very beginning that the evolutionary processor we discuss here is a mathematical object only and the biological hints presented above are intended to explain in an informal way how some biological phenomena are *sources of inspiration* for our mathematical computing model.

In [Margenstern et al. 2005] one presents a characterization of the complexity class **NP** based on accepting networks of evolutionary processors (ANEP for short). In [Margenstern et al. 2005] (and a series of subsequent papers), these ANEPs are actually called accepting *hybrid* networks of evolutionary processors). The work [Manea et al. 2007] discusses how ANEPs can be considered as problem solvers. In [Manea and Mitrana 2007], one shows that every recursively enumerable language can be accepted by an ANEP with 24 nodes. More precisely, one proposes a method for constructing, for every **NP**-language, an ANEP of size 24 deciding that language in polynomial time. While the number of nodes of this ANEP does not depend on the language, the other parameters of the network (rules, symbols, filters) depend on it. Since each ANEP may be viewed as a problem solver as shown in [Margenstern et al. 2005], the later result may be interpreted as a method for solving every **NP**-problem in polynomial time by ANEPs of constant size.

It is clear that filters associated with each node allow a strong control of the computation. Indeed, every node has an input and output filter; two nodes can exchange data if it passes the output filter of the sender *and* the input filter of the receiver. Moreover, if some data is sent out by some node and not able to enter any node, then it is lost. In this paper we simplify the ANEP model considered in [Margenstern et al. 2005] by moving the filters from the nodes to the edges. Each edge is viewed as a two-way channel such that the input and output filters, respectively, of the two nodes connected by the edge coincide. Clearly, the possibility of controlling the computation in such networks seems to be diminished. For instance, there is no possibility to loose data during the communication steps. In spite of this fact we prove here that these new devices are still computationally complete. This means that moving the filters from the nodes to the edges do not decrease the computational power of the model. Although the two variants are equivalent from the computational power point of view, we do not know a direct proof for this equivalence. As a consequence, following ideas from [Margenstern et al. 2005] we propose characterizations of two complexity classes, namely **NP** and **PSPACE**, in terms of accepting networks of evolutionary processors with filtered connections.

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 finite 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 $x \in W^*$.

A nondeterministic Turing machine (see, e.g., [Hartmanis 1968]) is a construct $M = (Q, V, U, \delta, q_0, B, F)$, where Q is a finite set of states, V is the input alphabet, U is the tape alphabet, $V \subset U$, q_0 is the initial state, $B \in U \setminus V$ is the "blank" symbol, $F \subseteq Q$ is the set of final states, and δ is the transition mapping, $\delta : (Q \setminus F) \times U \to 2^{Q \times (U \setminus \{B\}) \times \{R,L\}}$. The variant of a Turing machine we use in this paper can be described intuitively as follows: it has a semi-infinite tape (bounded to the left) divided into cells (each cell may store exactly one symbol from U). The machine has a central unit storing a state from a finite set of states, and a reading/writing tape head which scans the tape cells; the head cannot write blank symbols. The input is a word over V stored on the tape starting with the leftmost cell while all the other tape cells initially contain the symbol B. When M starts a computation, the tape head scans the leftmost cell and the central unit is in the state q_0. The machine performs moves that depend on the content of the cell currently scanned by the tape head and the

current state stored in the central unit. A move consists of: change the state, write a symbol from U on the current cell and move the tape head one cell either to the left (provided that the cell scanned was not the leftmost one) or to the right. An input word is accepted iff after a finite number of moves the Turing machine enters a final state. An instantaneous description (ID for short) of a Turing machine M as above is a word over $(U \setminus \{B\})^*Q(U \setminus \{B\})^*$. Given an ID $\alpha q\beta$, this means that the tape contents is $\alpha\beta$ followed by an infinite number of cells containing the blank symbol B, the current state is q, and the symbol currently scanned by the tape head is the first symbol of β provided that $\beta \neq \varepsilon$, or B, otherwise.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}, a \neq b$, 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 as above σ 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

$$\sigma^*(w) = \sigma^r(w) = \sigma^l(w) = \begin{cases} \{ubv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$

– If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, \\ \{w\}, \ \text{otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : \ w = av\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$

– 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 word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. 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:

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

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistic formulations of local gene mutations.

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

$$\varphi^s(x; P, F) \equiv P \subseteq alph(x) \ \wedge \ F \cap alph(x) = \emptyset$$
$$\varphi^w(x; P, F) \equiv alph(x) \cap P \neq \emptyset \ \wedge \ F \cap alph(x) = \emptyset.$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, the former condition requires (s stands for strong) that all permitting symbols are and no forbidding symbol is present in x, while the latter (w stands for weak) is a weaker variant such that at least one permitting symbol appears in x but still no forbidding symbol is present in x.

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

$$\varphi^\beta(L, P, F) = \{x \in L \mid \varphi^\beta(x; P, F)\}.$$

An *accepting network of evolutionary processors with filtered connections* (ANEPFC for short) is a 8-tuple

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

where:

- V and U are the input and network alphabet, respectively, such that $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph without loops with the set of nodes X_G and the set of edges E_G. Each edge is given in the form of a binary set. G is called the *underlying graph* of the network.
- $\mathcal{R} : X_G \longrightarrow 2^{Sub_U} \cup 2^{Del_U} \cup 2^{Ins_U}$ is a mapping which associates with each node 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 with each edge $e \in E_G$ the disjoint sets $\mathcal{N}(e) = (P_e, F_e)$.
- $\beta : E_G \longrightarrow \{s, w\}$ defines the *filter* type of an edge.
- $x_I, x_O \in X_G$ are the *input* and the *output* node of Γ, respectively.

We say that $card(X_G)$ is the size of Γ. Generally, the ANEPs considered in the literature have complete underlying graphs, namely graphs without loops in

which every two nodes are connected. Starting from the observation that every ANEPFC can be immediately transformed into an equivalent ANEPFC with a complete underlying graph (the edges that are to be added are associated with filters which make them useless), for the sake of simplicity we discuss in what follows ANEPFCs with underlying graphs having useful edges only. Note that this is not always possible for ANEPs.

A *configuration* of an ANEPFC 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 $x \in X_G \setminus \{x_I\}$.

A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules $\mathcal{R}(x)$ 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

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

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each word it has 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.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$\begin{aligned} C'(x) = (C(x) \setminus (& \bigcup_{\{x,y\} \in E_G} \varphi^{\beta(\{x,y\})}(C(x), \mathcal{N}(\{x,y\})))) \\ \cup (& \bigcup_{\{x,y\} \in E_G} \varphi^{\beta(\{x,y\})}(C(y), \mathcal{N}(\{x,y\}))) \end{aligned}$$

for all $x \in X_G$.

Let Γ be an ANEPFC, the computation of Γ on the input word $z \in V^*$ is a sequence of configurations $C_0^{(z)}, C_1^{(z)}, C_2^{(z)}, \ldots$, where $C_0^{(z)}$ is the initial configuration of Γ on z, $C_{2i}^{(z)} \Longrightarrow C_{2i+1}^{(z)}$ and $C_{2i+1}^{(z)} \vdash C_{2i+2}^{(z)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(z)}$ is uniquely determined by the configuration $C_{i-1}^{(z)}$. In other words, each computation in an ANEPFC is *deterministic*. A computation *halts* (and it is said to be *finite*) 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 Γ is

$$L_a(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is an accepting one.}\}$$

We say that an 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 $z \in V^*$ halts.

In a similar way to Turing machine, we define two computational complexity measures using ANEPFC as the computing model. To this aim we consider an ANEPFC Γ with the input alphabet V that halts on every input. The *time complexity* of the finite 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 partial function from $\mathbf{N}$ to $\mathbf{N}$,

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

We say that Γ decides L in time $O(f(n))$ if $Time_\Gamma(n) \in O(f(n))$.

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define:

$$\mathbf{Time}_{ANEPFC_p}(f(n)) = \{L \mid \text{there exists an ANEPFC } \Gamma, \text{ of size } p, \text{ deciding } L, \text{ and } n_0 \text{ such that } Time_\Gamma(n) \leq f(n) \text{ for all } n \geq n_0\}$$

Moreover, we write $\mathbf{PTime}_{ANEPFC_p} = \bigcup_{k \geq 0} \mathbf{Time}_{ANEPFC_p}(n^k)$ for all $p \geq 1$ as well as $\mathbf{PTime}_{ANEPFC} = \bigcup_{p \geq 1} \mathbf{PTime}_{ANEPFC_p}$.

The *length complexity* of the finite 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 equals

$$max\{|w| \ : \ w \in C_i^{(x)}(z), i \in \{1, \ldots, m\}, z \in X_G\}.$$

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

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

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Length}_{ANEPFC_p}(f(n)) = \{L \mid \text{there exists an ANEPFC } \Gamma, \text{ of size } p, \text{ deciding } L \text{ and } n_0 \text{ such that } Length_\Gamma(n) \leq f(n) \text{ for all } n \geq n_0\}$$

We write $\mathbf{PLength}_{ANEPFC_p} = \bigcup_{k \geq 0} \mathbf{Length}_{ANEPFC_p}(n^k)$, for all $p \geq 1$, and $\mathbf{PLength}_{ANEPFC} = \bigcup_{p \geq 1} \mathbf{PLength}_{ANEPFC_p}$.

3 An example

We briefly present an example, in order to exhibit the way ANEPFCs are used to solve difficult problems. The reader interested in all technical definitions concerning ANEPFC as problem solvers following the same definitions for ANEPs is referred to [Manea et al. 2007].

We consider the Vertex-Cover problem ([Garey and Johnson 1979]). Let $G = (V, E)$ be an undirected graph with the set of vertices V and the set of edges E. A vertex-cover of G is a set $A \subseteq V$ such that each edge $(u, v) \in E$ has at least one of u and v in A. The Vertex-Cover problem is: given a graph G and an integer k, does G have a vertex-cover of cardinality k or less?

In the following we assume that we are given an instance (G, k) of the Vertex-Cover problem, where the graph $G = (V, E)$ has $V = \{1, 2, \ldots, n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ for some $n, m \geq 1$. This instance is encoded by the word $w = e_1 \ldots e_m \#^k$, where $\#$ is a new symbol. Note that an edge e_i of the graph is also considered as a symbol and the fact that the number of nodes is not explicitly encoded in w. We present an ANEPFC Γ that can be constructed in (Turing) polynomial time and space and decides w if and only if G has a vertex-cover of cardinality k or less. The computation takes (ANEPFC) polynomial time.

For sake of clarity, we use the notation $[i, j]$ instead of $\{i, j\}$ for the edges of the graph (remember that the graph is undirected). Clearly, the input alphabet of the network is $U = \{[i, j] | 1 \leq i, j \leq n\} \cup \{\#\}$. We use $W = \{i | 1 \leq i \leq n\} \cup U$ as the working alphabet for Γ. Formally,

$$\Gamma = (U, W, H, \mathcal{R}, \mathcal{N}, \alpha, \beta, In, Out),$$

where H is the graph with $n + 2$ vertices $\{In, Out, N_1, \ldots, N_n\}$ and edges that are listed below.

The parameters of these nodes are defined as follows:

- The input node In :
  $-R(In) = \{\# \to i, 1 \leq i \leq n\}$
  $-\alpha(In) = *$
- The nodes $N_i, 1 \leq i \leq n$:
  $-R(N_i) = \{[i, j] \to \varepsilon, [j, i] \to \varepsilon | 1 \leq j \leq n\}$
  $-\alpha(N_i) = *$
- The output node Out :
  $-R(Out) = \emptyset$
  $-\alpha(Out) = *$

The parameters of the edges are defines as follows:

- The edges $\{In, N_i\}$, $1 \leq i \leq n$:
  - $P_{\{In,N_i\}} = \{i\}$
  - $F_{\{In,N_i\}} = \{\#\}$
  - $\beta(\{In, N_i\}) = w$
- The edges $\{N_i, Out)\}$ $1 \leq i \leq n$:
  - $P_{\{N_i,Out\}} = \emptyset$,
  - $F_{\{N_i,Out\}} = \{[i,k],[k,i] | 1 \leq k \leq n\}$
  - $\beta(\{N_i, Out\}) = w$
- The edges $\{N_i,N_j\}$, $1 \leq i,j \leq n$:
  - $P_{(\{N_i,N_j\})} = \{i,j\}$
  - $F_{(\{N_i,N_j\})} = \emptyset$
  - $\beta(\{N_i, N_j\}) = s$

Note that H is almost a complete graph excepting the edge $\{In, Out\}$ that doesn't allow any communication. This ANEPFC implements a very simple strategy: it replaces in k evolutionary steps all symbols # by integers between 1 and n. Note that two different occurrences of # may be replaced by the same number. The set of all these numbers, considered now as vertices of G, is further checked to see whether or not it is a vertex-cover of G. Thus, all such sets will be checked in parallel. If $\{i_1, \ldots, i_t\}$, $1 \leq t \leq k$, is a vertex cover of G, then a word $e_1 \ldots e_m x$, where x contains at least one occurrence of each i_j, $1 \leq j \leq t$, is sent out by x_I and enters simultaneously all nodes N_{i_j}, $1 \leq j \leq t$. In every visit of each node N_{i_j}, a symbol encoding an edge that connects i_j in G is deleted. All these strings are exchanged among all pairs of nodes N_p and N_q, provided that $p, q \in \{i_1, \ldots, i_t\}$, until all symbols encoding edges that connect either p or q are deleted. Now strings that do not contain any symbol from $\{e_1, e_2, \ldots, e_m\}$ can enter Out and the computation halts successfully.

Note that all the parameters of the network are bounded by a polynomial in n, and that the computation of the network on every input word encoding a graph with m edges for which one searches a vertex cover of cardinality at most k halts in at most $2(m + k + 1)$ steps. On the other hand, if G has no vertex cover of cardinality k or less, the input word, encoding G, is rejected after $2(m + k + 1)$ steps. Note that the network remains unchanged for all instances encoding graphs of size n which means that the solution presented above is uniform (see [Manea et al. 2007]).

4 Completeness of ANEPFCs

We start this section showing that ANEPFCs are computationally complete.

Theorem 1. *For any language L, accepted (decided) by a Turing Machine M, there exists an ANEPFC Γ, accepting (deciding) L. Moreover, Γ can be constructed such that:*

1. *if $L \in NTIME(f(n))$ then $Time_\Gamma(n) \in \mathcal{O}(f(n))$.*
2. *if $L \in NSPACE(f(n))$ then $Length_\Gamma(n) \in \mathcal{O}(f(n))$.*

Proof. Let $M = (Q, V, U, R, q_0, B, Q_f)$ be a Turing machine, without loss of generality we may assume that M never halts in the initial state q_0. We construct an ANEPFC that simulates, in parallel, all the computations of M on an input word. Let $\Gamma = (V, U', G, \mathcal{R}, \mathcal{N}, \alpha, \beta, x_I, x_O)$, where

$$U' = U \cup \{a', a^{\diamond}, a^{\bullet}, a^{\dagger}, a^{\circ}, a^{\oplus}, \bar{a}^{\circ}, \bar{a}^{\diamond}, \bar{a}^{\oplus}, a^{\ddagger}, a^{\spadesuit}, a^{\clubsuit} \mid a \in U \setminus \{B\}\} \cup \{B'\} \cup \{[q] \mid q \in Q\} \cup \{[q_1, a, q_2, b, X] \mid q_1, q_2 \in Q, a, b \in U, X \in \{L, R\}, \text{ and } \delta(q_1, a) \ni (q_2, b, X)\}$$

and G is the graph illustrated in Figure 1 whose nodes are listed below together with their associated sets of evolutionary rules.

Nodes of the graph in Figure 1 and their associated sets of evolutionary rules:

– $\underline{x_I}$:
$M(x_I) = \{\lambda \to B\}$, $\alpha(x_I) = r$

– $\underline{x'_I}$:
$M(x'_I) = \{\lambda \to B\}$, $\alpha(x'_I) = l$

– $\underline{x_1}$:
$M(x_1) = \{\lambda \to [q_0]\}$, $\alpha(x_1) = r$

– $\underline{x_2}$:
$M(x_2) = \{[q_1] \to [q_1, a, q_2, b, X] \mid q_1, q_2 \in Q, a, b \in U, X \in \{L, R\},$
where $\delta(q_1, a) \ni (q_2, b, X)\}$, $\alpha(x_2) = *$

– $\underline{x_3}$:
$M(x_3) = \{a \to a' \mid a \in U\}$, $\alpha(x_3) = *$

– $\underline{x_{1,a}}$, with $a \in U \setminus \{B\}$:
$M(x_{1,a}) = \{a' \to \lambda\}$, $\alpha(x_{1,a}) = l$

– $\underline{x_{1,B}}$:
$M(x_{1,B}) = \{B' \to \lambda\}$, $\alpha(x_{1,B}) = l$

– $\underline{x_4}$:
$M(x_4) = \{\lambda \to a^{\circ} \mid a \in U \setminus \{B\}\}$, $\alpha(x_4) = r$

– $\underline{x'_4}$:
$M(x'_4) = \{a^{\circ} \longrightarrow \bar{a}^{\circ} \mid a \in U \setminus \{B\}\}$, $\alpha(x'_4) = *$

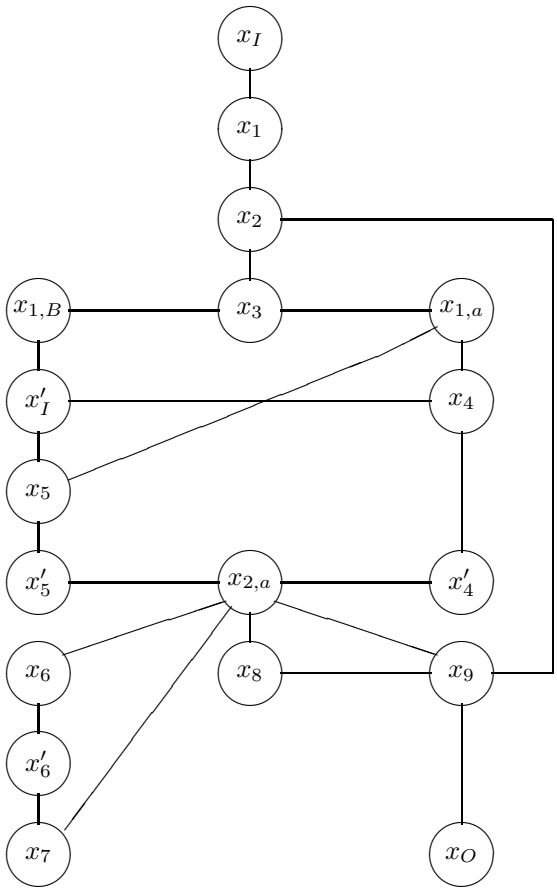
– $\underline{x_{2,a}}$, with $a \in U \setminus \{B\}$:
$M(x_{2,a}) = \{\bar{a}^{\circ} \to a^{\dagger}, \bar{a}^{\diamond} \to a^{\clubsuit}, \bar{a}^{\oplus} \to a^{\ddagger}, a^{\bullet} \to a^{\spadesuit}\}$, $\alpha(x_{2,a}) = *$

– $\underline{x_5}$:
$M(x_5) = \{\lambda \to a^{\oplus} \mid a \in U \setminus \{B\}\}$, $\alpha(x_5) = l$

– $\underline{x'_5}$:
$M(x'_5) = \{a^{\oplus} \to \bar{a}^{\oplus} \mid a \in U \setminus \{B\}\}$, $\alpha(x'_5) = *$

– $\underline{x_6}$:
$M(x_6) = \{\lambda \to a^{\diamond} \mid a \in U \setminus \{B\}\}$, $\alpha(x_6) = l$

– $\underline{x'_6}$:
$M(x'_6) = \{a^{\diamond} \to \bar{a}^{\diamond} \mid a \in U \setminus \{B\}\}$, $\alpha(x'_6) = *$



(Note that nodes $x_{1,a}$ and $x_{2,a}$ has to be understood as generic nodes for all $a \in U \setminus \{B\}$.)

Figure 1: The graph of the ANEPFC used in the proof of Theorem 1.

– $\underline{x_7}$:
$M(x_7) = \{a \to a^\bullet \mid a \in U \setminus \{B\}\}$, $\alpha(x_7) = *$
– $\underline{x_8}$:
$M(x_8) = \{a^\spadesuit \to \lambda \mid a \in U \setminus \{B\}\}$, $\alpha(x_8) = r$
– $\underline{x_9}$:
$M(x_9) = \{[q_1, c, q_2, a, X] \to [q_2] \mid q_1, q_2 \in Q, c, a \in U, X \in \{L, R\}$ where $\delta(q_1, c) \ni (q_2, a, X)\} \cup \{a^\dagger \to a, a^\ddagger \to a, a^\clubsuit \to a \mid a \in U \setminus \{B\}\}$, $\alpha(x_9) = *$.

The edges of the graph together and their filters are defined as follows:

– $\mathcal{N}(\{x_I, x_1\}) = (\{B\}, \{[q_0]\})$, $\beta(\{x_I, x_1\}) = s$;

– $\mathcal{N}(\{x_1, x_2\}) = (\{[q_0]\}, \emptyset)$, $\beta(\{x_1, x_2\}) = w$;

– $\mathcal{N}(\{x_2, x_3\}) = (\{[q_1, a, q_2, b, X] \mid q_1, q_2 \in Q, a, b \in U, X \in \{L, R\}$, where $\delta(q_1, a) \ni (q_2, b, X)\}, \{a' \mid a \in U\})$, $\beta(\{x_2, x_3\}) = w$;

– $\mathcal{N}(\{x_3, x_{1,a}\}) = (\{a'\}, \{[q_1, c, q_2, b, X] \mid q_1, q_2 \in Q, c, b \in U, X \in \{L, R\}$ where $\delta(q_1, c) \ni (q_2, b, X)$ and $c \neq a\})$, $\beta(\{x_3, x_{1,a}\}) = s$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x_3, x_{1,B}\}) = (\{B'\}, \{[q_1, c, q_2, b, X] \mid q_1, q_2 \in Q, c, b \in U, X \in \{L, R\}$ where $\delta(q_1, c) \ni (q_2, b, X)$ and $c \neq B\})$, $\beta(\{x_3, x_{1,a}\}) = s$;

– $\mathcal{N}(\{x'_I, x_{1,B}\}) = (\{[q_1, B, q_2, a, X] \mid q_1, q_2 \in Q, a \in U, X \in \{L, R\}$ where $\delta(q_1, B) \ni (q_2, a, X)\}$,{B'}), $\beta(\{x'_I, x_{1,B}\}) = w$;

– $\mathcal{N}(\{x'_I, x_4\}) = (\{[q_1, a, q_2, b, R] \mid q_1, q_2 \in Q, a, b \in U$ where $\delta(q_1, a) \ni (q_2, b, R)\}, \{a^\circ\}), \beta(\{x'_I, x_4\}) = w$;

– $\mathcal{N}(\{x_{1,a}, x_4\}) = (\{[q_1, a, q_2, b, R] \mid q_1, q_2 \in Q, b \in U$ where $\delta(q_1, a) \ni (q_2, b, R)\}$, $\{c^\circ \mid c \in U \setminus \{B\}\}), \beta(\{x_{1,4}, x_4\}) = w$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x'_I, x_5\}) = (\{[q_1, a, q_2, b, L] \mid q_1, q_2 \in Q, a, b \in U$ where $\delta(q_1, a) \ni (q_2, b, L)\}, \{a^\circ\}), \beta(\{x'_I, x_4\}) = w$;

– $\mathcal{N}(\{x_{1,a}, x_5\}) = (\{[q_1, a, q_2, b, L] \mid q_1, q_2 \in Q, b \in U$ where $\delta(q_1, a) \ni (q_2, b, L)\}$, $\{c^\oplus \mid c \in U \setminus \{B\}\}), \beta(\{x_{1,4}, x_5\}) = w$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x_4, x'_4\}) = (\{a^\circ \mid a \in U \setminus \{B\}\}, \{\bar{a}^\circ \mid a \in U \setminus \{B\}\}), \beta(\{x_4, x'_4\}) = w$;

– $\mathcal{N}(\{x_5, x'_5\}) = (\{a^\oplus \mid a \in U \setminus \{B\}\}, \{\bar{a}^\oplus \mid a \in U \setminus \{B\}\}), \beta(\{x_5, x'_5\}) = w$;

– $\mathcal{N}(\{x'_4, x_{2,a}\}) = (\{\bar{a}^\circ\}, \{[q_1, c, q_2, b, R] \mid q_1, q_2 \in Q, c, b \in U$, where $\delta(q_1, c) \ni (q_2, b, R)$ and $b \neq a\})$, $\beta(\{x'_4, x_{2,a}\}) = s$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x'_5, x_{2,a}\}) = (\{\bar{a}^\oplus\}, \{[q_1, c, q_2, b, L] \mid q_1, q_2 \in Q, c, b \in U$, where $\delta(q_1, c) \ni (q_2, b, L)$ and $b \neq a\})$, $\beta(\{x'_5, x_{2,a}\}) = s$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x_6, x_{2,a}\}) = (\{a^\ddagger \mid a \in U \setminus \{B\}\}, \{a^\diamond \mid a \in U \setminus \{B\}\})$, $\beta(\{x_6, x_{2,a}\}) = w$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x_6, x'_6\}) = (\{a^\diamond \mid a \in U \setminus \{B\}\}, \{\bar{a}^\diamond \mid a \in U \setminus \{B\}\}), \beta(\{x_6, x'_6\}) = w$;

– $\mathcal{N}(\{x_7, x'_6\}) = (\{\bar{a}^\diamond \mid a \in U \setminus \{B\}\}, \{a^\bullet \mid a \in U \setminus \{B\}\}), \beta(\{x_7, x'_6\}) = w$;

– $\mathcal{N}(\{x_7, x_{2,a}\}) = (\{a^\bullet\}, \{b^\clubsuit \mid b \in U \setminus \{B\} \cup \bar{b}^\diamond \mid b \in U \setminus \{B, a\}\})$, $\beta(\{x_7, x_{2,a}\}) = s$, for all $a \in U \setminus \{B\}$;

– $\mathcal{N}(\{x_8, x_{2,a}\}) = (\{a^\spadesuit\}, \{\bar{c}^\diamond \mid c \in U \setminus \{B\}\})$, $\beta(\{x_8, x_{2,a}\}) = s$, for all $a \in U \setminus \{B\}$;

- $\mathcal{N}(\{x_9, x_{2,a}\}) = (\{a^\dagger\}, \{a^\bullet, a^\clubsuit\})$, $\beta(\{x_9, x_{2,a}\}) = w$, for all $a \in U \setminus \{B\}$;
- $\mathcal{N}(\{x_8, x_9\}) = (\{a^\clubsuit \mid a \in U \setminus \{B\}\}, \{a^\bullet \mid a \in U \setminus \{B\}\})$, $\beta(\{x_9, x_{2,a}\}) = w$;
- $\mathcal{N}(\{x_9, x_2\}) = (\{[q] \mid q \in Q \setminus Q_f\}, \emptyset), \beta(\{x_9, x_2\}) = w$;
- $\mathcal{N}(\{x_9, x_O\}) = (\{[q] \mid q \in Q_f\}, \emptyset), \beta(\{x_9, x_O\}) = w$;

In the sequel we argue that the ANEPFC constructed above accepts/decides the same language as M does. To this aim, we assume that the input word of Γ is $w \in V^*$. The computation of Γ on w is structured in three phases, as we describe below.

The first phase of the computation is a *pre-processing phase*. In the beginning of the computation of Γ on w, every node is empty, except for x_I which contains w. This word will be transformed into wB in an evolutionary step; then it verifies the conditions imposed by the filters of the edge $\{x_I, x_1\}$, hence it is sent out by x_I and enters x_1; no other communication is possible. In this node, wB is transformed into $wB[q_0]$ and the pre-processing phase is finished.

After this first phase, the computation enters the *simulation phase*. At the beginning of each iteration of this phase, we assume that in the last evolutionary step a word of the form $w_1B[q]w_2$ was produced either by the node x_1 or by the node x_9; note that this condition holds after the pre-processing phase, when the word $wB[q_0]$ was obtained in x_1. The node x_9 can communicate such a word to either x_2 or x_O only. When the word enters the output node x_O, provided that $q \in F$, the computation enters the *accepting phase*, namely the computation halts and w is accepted. Otherwise, the word enters node x_2. Here one starts to simulate the transition $\delta(q_1, a) \ni (q_2, b, X)$ applied to the ID w_2qw_1 of M. In x_2 the word $w_1B[q]w_2$ becomes $w_1B[q_1, a, q_2, b, X]w_2$ and is communicated to x_3, where an occurrence of a symbol $c \in U$ is replaced by c'. Now, the current word can be communicated to $x_{1,a}$ only, provided that $a = c$. Here a' is deleted provided that a' is the first symbol of w_1; otherwise the computation halts.

We now distinguish two cases: $a \in U \setminus \{B\}$ and $a = B$. For each of these two cases we consider two more cases: $X = R$ and $X = L$. We start analyzing the rest of the computation for $a \in U \setminus \{B\}$ and $X = R$. The current word simply enters x_4 where it receives c° in its right-hand end. Further, it enters x'_4 where c° is replaced by $\bar{c}^\circ$. In this way, we prevent from having a false infinite computation. There is only one possibility to continue the computation with this word in Γ, namely it enters the node $x_{2,b}$ providing $c = b$. The new word enters now x_9 where $b^\dagger$ (obtained in $x_{2,b}$) and $[q_1, a, q_2, b, R]$ are replaced by b and q_2, respectively. Now the simulation of the transition $\delta(q_1, a) \ni (q_2, b, R)$, $a \in U \setminus \{B\}$, on w_2qw_1 in M is completed in Γ. We now consider the case $a = B$ and $X = R$. The situation is just a bit different. First, the word $[q_1, a, q_2, b, X]w_2$ enters x'_I, where B is added as the first symbol, then it follows the same itinerary, namely x_4, x'_4,

$x_{2,b}$, x_9, as in the previous case. Now, the simulation can be resumed with the node x_2.

The situation is a bit more intricate for the other two cases. We start analyzing the case $a \in U \setminus \{B\}$ and $X = L$, mentioning that the last case can be treated analogously to the case $a = B$ and $X = R$. The current word is sent out by $x_{1,a}$ and is received by x_5. Here $c^{\oplus}$ is added to its left-hand end, then it enters x'_5 where $\bar{c}^{\oplus}$ is substituted for $c^{\oplus}$. Now the only way to continue the computation is that $x_{2,b}$ to receive the current word. This is possible only if $c = b$. In $x_{2,b}$, $\bar{b}^{\oplus}$ is substituted by $b^{\ddagger}$ and the new word enters x_6. The continuation is as follows: in x_6 the current word receives $c^{\diamond}$ in its left-hand end which is then transformed into $\bar{c}^{\diamond}$ in x'_6, and an occurrence of d is replaced by $d^{\bullet}$ in x_7. The new word is sent back to a node of the form $x_{2,t}$, but this communication is not successful unless $c = d = t$. When entering $x_{2,c}$ the current word is of the form $\bar{c}^{\diamond}b^{\ddagger}w'_1[q_1, a, q_2, b, L]w'_2c^{\bullet}w''_2$. We take $w_1 = aw'_1$ and $w_2 = w'_1cw''_2$. Here $\bar{c}^{\diamond}$ and $c^{\bullet}$ are replaced by $c^{\clubsuit}$ and $c^{\spadesuit}$, respectively. The new word enters x_8 where $c^{\spadesuit}$ is deleted provided that it is the rightmost symbol; otherwise, the computation is blocked. After leaving the node x_8, the current word enters x_9 where it becomes $cbw'_1B[q_2]w_3$, where $w_2 = w_3c$. Now the simulation can be resumed by sending the current word to x_2.

From the above considerations, one can easily state the following fact: the word $w_1B[q]w_2$ can be transformed in one iteration of the *simulation phase* into $x_1B[q']x_2$ iff the ID w_2qw_1 of the Turing machine M moves in one step to the ID $x_2q'x_1$. In conclusion, the languages accepted/decided by Γ and M are the same.

The next remarks conclude our proof. An iteration of the *simulation phase* requires at most 13 evolutionary steps and 13 communication steps. Therefore, the number of both evolutionary and communication steps required to simulate one step of the Turing machine M is at most 26. Therefore, the number of steps made by Γ in simulating the computation of M on some input word w is $\mathcal{O}(f(|w|))$, where $f(|w|)$ is the number of steps that M makes on the same input. Also, the words that may appear in the network during the simulation of a step of the Turing machine M have the length bounded by the length of the configurations of the Turing machine, that appear in the simulated step, plus a constant. Thus, if M uses $f(|w|)$ space on the input word w, Γ will use $\mathcal{O}(f(|w|))$. □

Since **NP** is defined as the class of languages *decided* by nondeterministic Turing machines in polynomial time **PSPACE** is defined as the class of languages *decided* by nondeterministic Turing machines using polynomial space, we obtain as a consequence of Theorem 1:

Corollary 2.

1. $\mathbf{NP} \subseteq \mathbf{PTime}_{ANEPFC}$.

2. $\mathbf{PSPACE} \subseteq \mathbf{PLength}_{ANEPFC}$.

The reversal of Theorem 1 holds as well.

Theorem 3. *For any ANEPFC Γ accepting the language L, there exists a Turing machine M accepting L. Moreover, M can be constructed such that:*

1. M accepts in $\mathcal{O}((Time_\Gamma(n))^2)$ computational time an input word of length n from L, and

2. M accepts in $\mathcal{O}(Length_\Gamma(n))$ space for an input word of length n from L.

Proof. We construct a nondeterministic Turing machine M as follows:

(1) M has a finite set of states associated with the evolutionary rules of each node of Γ. M has also a finite set of states associated with each edge in the underlying graph which is divided into two disjoint subsets each associated with one of the two ways in which that edge can be traversed (the state encodes the sending node and the destination node); each of these subsets is divided, in its turn, into two other disjoint sets, each associated with the two filters on that edge. In each state we encode also the way rules or, respectively, filters are to be applied.

(2) The input word of Γ is initially on the tape of M. The Turing machine simulates nondeterministically its itinerary through the underlying network of Γ. Let us suppose that the contents of the tape of M is α; M works according to the following strategy:

(i) When M enters a state associated to a rule $a \to b$ (where $a, b \in V \cup \{\lambda\}, a \neq b$) from the node x, it applies this rule to an occurrence of a in α, if any, chosen according to the way this rule should be applied: the leftmost symbol of α, the rightmost symbol of α, or a symbol chosen nondeterministically from α. The word obtained after applying this rule becomes the current word, and the state is changed to a state corresponding to a filter placed on an edge $\{x, z\}$ for some node z (x is the sending node, while z is the destination node). Otherwise, if α does not contain any occurrence of a that verifies the way the rule should be applied, M blocks the computation.

(ii) When M enters a state from the subset of states associated to a filter, it checks whether α can pass that filter. If α passes the filter, M enters a state associated with a rule from the destination node nondeterministically chosen. Otherwise, if α does not pass it, M blocks the computation.

(iii) As soon as M has checked the filter conditions from a node to the output node of Γ, it accepts its input word.

It is rather plain that M accepts L. The following complexity related observations can be made. If Γ needs at most $f(n)$ steps to accept an word of length n, then the Turing machine M needs at most $\mathcal{O}(f^2(n))$ steps to accept the same word. This is due to the fact that in the simulation of each of the $f(n)$ steps of the computation of Γ, M needs to search for the occurrences of some symbols in the word on its tape and, in the case when an evolutionary step is simulated, to replace these symbol with other ones; the number of steps needed to complete these operations is $\mathcal{O}(f(n))$, for each simulated step. Also, $\mathcal{O}(f(n))$ operations are necessary to simulate a communication step. On the other hand, if Γ produces words of length at most $f(n)$ during a computation on a word of length n, then the Turing machine M will have words of length at most $f(n)$ on its tape. □

Considering that **NP** equals the class of languages *accepted* by nondeterministic Turing machines in polynomial time and **PSPACE** equals the class of languages *accepted* by nondeterministic Turing machines using polynomial space, we can state as a direct consequence of this theorem the following result:

Corollary 4.
1. $\mathbf{PTime}_{ANEPFC} \subseteq \mathbf{NP}$.
2. $\mathbf{PLength}_{ANEPFC} \subseteq \mathbf{PSPACE}$.

Consequently, we have proved that:

Theorem 5.
1. $\mathbf{NP} = \mathbf{PTime}_{ANEPFC}$.
2. $\mathbf{PSPACE} = \mathbf{PLength}_{ANEPFC}$.

5 Conclusion

We proposed here new characterizations of **NP** and **PSPACE** using a complete computation model inspired from the evolution of communities of cells, in biology. In our view, further investigations in this area should aim descriptional complexity measures regarding ANEPFC.

References

[Errico and Jesshope 1994] Errico, L., Jesshope, C.: "Towards a new architecture for symbolic processing"; Artificial Intelligence and Information-Control Systems of Robots 94, World Scientific, Singapore (1994) 31-40.

[Fahlman et.al. 1983] Fahlman, S., Hinton, G., Seijnowski, T.: "Massively parallel architectures for AI: NETL, THISTLE and Boltzmann Machines"; Proc. AAAI National Conf. on AI, William Kaufman, Los Altos (1983) 109-113.

[Garey and Johnson 1979] Garey, M., Johnson, D.: "Computers and Intractability. A Guide to the Theory of NP-completeness"; Freeman, San Francisco, CA (1979).

[Hartmanis 1968] Hartmanis, J.: "Computational complexity of one-tape Turing machine computations"; Journal of the Association for Computing Machinery 15 (1968), 325-339.

[Hillis 1985] Hillis, W.: "The Connection Machine"; MIT Press, Cambridge (1985).

[Manea and Mitrana 2007] Manea, F., Mitrana, V.: "All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size"; Information Processing Letters 103 (2007), 112-118.

[Manea et al. 2007] Manea, F., Martin-Vide, C., Mitrana,,V.: "On the size complexity of universal accepting hybrid networks of evolutionary processors"; Mathematical Structures in Computer Science 17 (2007), 753-771.

[Margenstern et al. 2005] Margenstern, M., Mitrana, V., Perez-Jimenez, M.: "Accepting hybrid networks of evolutionary systems"; DNA Based Computers 10, LNCS 3384, Springer-Verlag, Berlin (2005) 235-246.

[Martín-Vide and Mitrana 2005] Martín-Vide, C., Mitrana, V.: "Networks of evolutionary processors: results and perspectives"; Molecular Computational Models: Unconventional Approaches, Idea Group Publishing, Hershey (2005) 78-114.

[Păun et al. 1998] Păun, G., Rozenberg, G., Salomaa, A.: "DNA Computing. New Computing Paradigms"; Springer-Verlag, Berlin (1998).

[Sankoff et al. 1992] Sankoff, D., et al.: "Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome"; Proc. Natl. Acad. Sci. USA 89 (1992), 6575-6579.