


Accepting Hybrid Networks of Evolutionary Processors

Maurice Margenstern¹, Victor Mitrana², and Mario J. Pérez-Jiménez³

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by idUS. Depósito de Investigación Universidad de Sevilla

`margens@lita.univ-metz.fr`

² Faculty of Mathematics and Computer Science, University of Bucharest,
Str. Academiei 14, 70109, Bucharest, Romania

and

Research Group in Mathematical Linguistics, Rovira i Virgili University,
Pça. Imperial Tarraco 1, 43005, Tarragona, Spain

`vmi@fll.urv.es`

³ Department of Computer Science and Artificial Intelligence,
University of Seville

`Mario.Perez@cs.us.es`

Abstract. We consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that the classical complexity class **NP** equals the set of languages accepted by AHNEPs in polynomial time.

1 Introduction

The origin of networks of evolutionary processors (NEPs for short) is twofold. In [5] 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 words which describe their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of words, where each word represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on words. Only those cells are accepted as surviving (correct) ones which are represented by a word in a given set of words, called the genotype space of the species. This feature parallels with the natural process of evolution.

On the other hand, a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [10] as well as the Logic Flow paradigm [6], 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 such data can be communicated which can pass a filtering process. 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, e.g., [7, 10].

In [1] (further developed in [2, 11, 3]), we modify this concept (considered in [4] from a formal language theory point of view) 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 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. Obviously, the computational process described here 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 [12]. Consequently, hybrid networks of evolutionary processors might be viewed as bio-inspired computing models. We want to stress from the very beginning that we are not concerned here with a possible biological implementation, though a matter of great importance.

In a series of papers, we present *linear* time solutions to some NP-complete problems using these simple mechanisms. Such solutions are presented for the Bounded Post Correspondence Problem in [1], for the “3-colorability problem” in [2] (with simplified networks), and for the Common Algorithmic Problem in [11]. In this paper, we consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that **NP** equals the class of languages accepted by AHNEPs in polynomial time.

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 $\text{alph}(x)$ denotes the minimal alphabet W such that $x \in W^*$.

A nondeterministic *Turing machine* is a construct $T = (Q, V, U, \delta, q_0, B, F)$, 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, $\delta : (Q \setminus F) \times U \longrightarrow 2^{Q \times U \times \{R, L\}}$. Moreover, if $(s, B, X) \in \delta(q, a)$ for some $s, q \in Q$ and $X \in \{R, L\}$, then $a = B$, i.e. T never write B over a symbol different than B . The variant of a Turing machine we use in this paper can be described intuitively as follows: it has a tape divided into cells that may store symbols from U (each cell may store exactly one symbol from U). The tape is semi-infinite, namely it is bounded to the left (there is a leftmost cell) and unbounded (arbitrarily long) to the right. The machine has a central unit which can be in a state from a finite set of states, and a reading/writing tape head which can scan in turn the tape cells. This head cannot go the the left-hand end of the tape. The input word is a word over V and is stored on the tape starting with the leftmost cell and all the other tape cells contain the symbol B .

Initially, the tape head scans the leftmost cell and the central unit is in the state q_0 . The machine performs moves. A move depends on the contents of the cell currently scanned by the tape head and the current state of 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 T as above is a string 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 $a \rightarrow b$, 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 as above σ and a string $w \in V^*$, we define the following *actions* of σ on w :

- If $\sigma \equiv a \rightarrow b \in \text{Sub}_V$, 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 $\sigma \equiv a \rightarrow \varepsilon \in \text{Del}_V$, then $\sigma^*(w) = \begin{cases} \{w : \exists u, v \in V^* (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$

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

- If $\sigma \equiv \varepsilon \rightarrow a \in \text{Ins}_V$, then

$$\sigma^*(w) = \{uav : \exists u, v \in V^* (w = uv)\}, \quad \sigma^r(w) = \{wa\}, \quad \sigma^l(w) = \{aw\}.$$

$\alpha \in \{*, l, r\}$ expresses the way of applying an evolution 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:

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

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 w over V , we define the predicates

$$\begin{aligned} \varphi^{(1)}(w; P, F) &\equiv P \subseteq \text{alph}(w) && \wedge F \cap \text{alph}(w) = \emptyset \\ \varphi^{(2)}(w; P, F) &\equiv \text{alph}(w) \subseteq P \\ \varphi^{(3)}(w; P, F) &\equiv P \subseteq \text{alph}(w) && \wedge F \not\subseteq \text{alph}(w) \\ \varphi^{(4)}(w; P, F) &\equiv \text{alph}(w) \cap P \neq \emptyset && \wedge F \cap \text{alph}(w) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*).

For every language $L \subseteq V^*$ and $\beta \in \{(1), (2), (3), (4)\}$, we define:

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

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

- Either $(M \subseteq \text{Sub}_V)$ or $(M \subseteq \text{Del}_V)$ or $(M \subseteq \text{Ins}_V)$. 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 .

An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-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 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 words existing in that node.
- $\beta : X_G \longrightarrow \{(1), (2), (3), (4)\}$ 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:

$$\begin{aligned} \text{input filter:} \quad & \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x), \\ \text{output filter:} \quad & \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x). \end{aligned}$$

That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the string w can pass the input (resp. output) filter of x . More generally, $\rho_x(L)$ (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 $\text{card}(X_G)$ 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, e.g., *rings, stars, grids*, etc. Networks of evolutionary processors with underlying graphs having these special forms have been considered in [1, 2, 11, 3]. We focus here on *complete* AHNEPs, i.e., AHNEPs having a complete underlying graph denoted by K_n , where n is the number of vertices.

A *configuration* of a AHNEP 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 $x \in X_G - \{x_I\}$.

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 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, each node processor $x \in X_G$ sends one copy of each string it has, which is able to pass the output filter of x , to all the node processors connected to x and receives all the strings 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 $x \in X_G$.

Let Γ be an AHNEP, the computation of Γ on the input string $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \dots$, where $C_0^{(w)}$ is the initial configuration of Γ on w , $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. Otherwise stated, each computation in an AHNEP 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 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 consecutive identical configurations.

The *language accepted* by Γ is

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

3 Complexity Classes

The reader is referred to [8, 9] for the classical time and space complexity classes defined on the standard computing model of Turing machine.

We define some computational complexity measures by using AHNEP as the computing model. To this aim we consider a AHNEP Γ and the language L accepted by Γ . The *time complexity* of the accepting computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \dots, C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Time_\Gamma(x)$ and equals m . The time complexity of Γ is the partial function from \mathbf{N} to \mathbf{N} ,

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

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

$$\mathbf{Time}_{AHNEP}(f(n)) = \{L \mid \text{there exists an AHNEP } \Gamma \text{ and } n_0 \text{ such that} \\ L = L(\Gamma) \text{ and } \forall n \geq n_0 (Time_\Gamma(n) \leq f(n))\}$$

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

Now we prove a result which establishes a strong connection between the complexity classes defined on Turing machines and those defined on AHNEPs.

Proposition 1. *For any nondeterministic Turing machine, M , recognizing a language L there exists an AHNEP, Γ , accepting the same language L . Moreover, if M works within time $f(n)$ then $Time_\Gamma(n) \in O(f(n))$.*

Proof. Let $M = (Q, V_1, V_2, \delta, q_0, B, F)$ be an arbitrary Turing machine. We define the new alphabets:

$$U_1^{(K)} = \{\langle s, b, K, a \rangle \mid (s, b, K) \in \delta(q, a), s, q \in Q, a, b \in V_2 \setminus \{B\}\}, K \in \{R, L\},$$

$$U_2 = \{[a, b] \mid a, b \in V_2 \setminus \{B\}\} \quad U_3 = \{X_a \mid a \in V_2 \setminus \{B\}\},$$

$$U_4 = \{Y_a^{(b)} \mid a, b \in V_2 \setminus \{B\}\} \quad U_5 = \{Z_a \mid a \in V_2 \setminus \{B\}\},$$

$$U_6 = \{W_a \mid a \in V_2 \setminus \{B\}\} \quad U_7 = \{\bar{s}a \mid s \in Q, a \in V_2 \setminus \{B\}\},$$

$$U_8 = \{Y_a \mid a \in V_2 \setminus \{B\}\} \quad U_{10} = \{\tilde{s}a \mid s \in Q, a \in V_2 \setminus \{B\}\}$$

$$U_9^{(K)} = \{\langle\langle s, a, K, q \rangle\rangle \mid (s, a, K) \in \delta(q, B), s, q \in Q, a \in V_2 \setminus \{B\}\}, K \in \{R, L\}.$$

Furthermore, for an alphabet T we denote by T' the alphabet consisting of the primed copies of all symbols in T . Now, we put

$$U = U_1^{(R)} \cup U_1^{(L)} \cup U_2 \cup U_3 \cup U_4 \cup U_5 \cup U_6 \cup U_7 \cup U_8 \cup U_9^{(R)} \cup U_9^{(L)} \cup \\ U_{10} \cup V_2 \cup Q \cup U_3' \cup U_5' \cup U_6' \cup U_8' \cup (V_2 \setminus \{B\})'$$

We define the AHNEP $\Gamma = (V_1, U, G, N, \alpha, \beta, x_I, x_O)$, where G is a complete graph whose nodes are described below.

$$M_{x_I} = \{\varepsilon \rightarrow q_0\}, PI_{x_I} = \emptyset, FI_{x_I} = U, PO_{x_I} = \emptyset, FO_{x_I} = \emptyset, \alpha(x_I) = r, \\ \beta(x_I) = (1).$$

Table 1

Node	M	PI	FI	PO	FO	α	β
$x_1^{(\neq B)}$	$\{q \rightarrow \langle s, b, K, a \rangle \mid (s, b, K) \in \delta(q, a)\}$	\emptyset	$U \setminus (V_2 \cup Q)$	\emptyset	\emptyset	*	(1)
$x_1^{(\neq B)}(a, b)$	$\{\varepsilon \rightarrow Y_b^{(a)}\}$	$\{\langle s, b, R, a \rangle\}$	$U \setminus (V_2 \cup U_1^{(R)} \cup U_4)$	\emptyset	\emptyset	r	(1)
$x_1^{(\neq B)}(a)$	$\{a \rightarrow X_a\}$	$\{Y_b^{(a)}\}$	$U \setminus (V_2 \cup U_1^{(R)} \cup U_3 \cup U_4)$	U_3	\emptyset	*	(4)
$x_2^{(\neq B)}$	$\{\langle s, b, R, a \rangle \rightarrow s, Y_b^{(a)} \rightarrow b\}$	U_3	$U \setminus (V_2 \cup U_1^{(R)} \cup U_3 \cup U_4)$	$U \setminus (U_1^{(R)} \cup U_4)$	$U_1^{(R)} \cup U_4$	*	(4)
$x_3^{(\neq B)}$	$\{X_a \rightarrow \varepsilon\}$	U_3	$U \setminus (V_2 \cup U_3)$	$U \setminus U_3$	U_3	l	(4)

The nodes described in Table 1 are used for simulating a move of M which consists in reading a symbol different from B , possibly changing the state as well as the read symbol, and moving the tape head to the right. In this table, $s, q \in Q$, $a, b \in V_2 \setminus \{B\}$ and $K \in \{R, L\}$. Each table is accompanied by some explanations which emphasize the simulation mode. By the definition of the input node x_I , for any input string $w \in V_1^*$, $C_1^{(w)}(x_I) = \{wq_0\}$. In the next communication step both nodes $x_1^{(\neq B)}$ and $x_1^{(=B)}$ (which will be defined later) receive a copy of wq_0 . Note that the initial ID of a computation of M on w is q_0w . Let us consider now an ID $\alpha q\beta$, which can be obtained by a computation in M starting with q_0w . By induction, we may assume that $\beta q\alpha \in C_m^{(w)}(x_1^{(\neq B)}) \cap C_m^{(w)}(x_1^{(=B)})$ for some $m \geq 1$. Let us suppose that $\beta = a\beta'$, $a \in V_2 \setminus \{B\}$, $\beta' \in (V_2 \setminus \{B\})^*$. Clearly, $C_{m+1}^{(w)}(x_1^{(=B)}) \supseteq \{\beta\langle s, b, K, c \rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}$. Obviously, only those strings with $c = a$ from the above ones are useful for our simulating process. Now, let us follow what happens with a string $\beta\langle s, b, R, a \rangle\alpha$ for some fixed $s \in Q, b \in V_2 \setminus \{B\}$ in the following steps. This string is accepted by $x_1^{(\neq B)}(a, b)$ only, where $Y_b^{(a)}$ is appended to its right-hand end. The resulting string $\beta\langle s, b, R, a \rangle\alpha Y_b^{(a)}$ is sent out by $x_1^{(\neq B)}(a, b)$ and $x_1^{(\neq B)}(a)$ is the unique node which can receive it. Here, exactly one occurrence of a in different copies of $\beta\langle s, b, R, a \rangle\alpha Y_b^{(a)}$ is replaced by X_a and all the obtained strings leave $x_1^{(\neq B)}(a)$. (We shall see later that only those strings starting with a in which this first occurrence of a is replaced by X_a can further navigate through the network; the others remain in $x_3^{(\neq B)}$ forever.) Then, all of them enter the node $x_2^{(\neq B)}$ where $\langle s, b, R, a \rangle$ and $Y_b^{(a)}$ are replaced by s and b , respectively. Both symbols must be replaced in two consecutive evolutionary steps since the output filter of $x_2^{(\neq B)}$ prevents leaving of this node by the strings containing symbols from $U_1^{(R)}$ or U_4 . All the strings leaving $x_2^{(\neq B)}$ arrive in $x_3^{(\neq B)}$ where those starting with X_a can leave $x_3^{(\neq B)}$ after having removed X_a from their left-hand end, while the others remain in $x_3^{(\neq B)}$ forever. In this way, we check whether or not the first letter of β is indeed a . By the above explanations, we infer that

$$C_{m+14}^{(w)}(x_1^{(\neq B)}) \supseteq \{\beta' s a b \mid (s, b, R) \in \delta(q, a), s \in Q, b \in V_2 \setminus \{B\}\}.$$

The nodes described in Table 2, together with $x_1^{(\neq B)}$ are used for simulating a move of M which consists in reading a symbol different from B , possibly changing the state as well as the read symbol, and moving the tape head to the left, provided that this is possible. In this table, $s, q \in Q$ and $a, b \in V_2 \setminus \{B\}$.

We start our explanation by returning to the configuration $C_{m+1}^{(w)}(x_1^{(=B)}) \supseteq \{\beta\langle s, b, K, c \rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}$. In the sequel, we follow a string $\beta\langle s, b, L, a \rangle\alpha$ for some fixed $s \in Q, b \in V_2 \setminus \{B\}$. This string enters $x_2^{(\neq B)}(a, b)$ where, similarly to the situation described above when the followed string reached $x_1^{(\neq B)}(a)$, exactly one occurrence of a in different copies of $\beta\langle s, b, L, a \rangle\alpha$ is replaced by $[a, b]$. As we shall see later, the node $x_5^{(\neq B)}$ blocks all the strings obtained in $x_2^{(\neq B)}(a, b)$ which do not start with $[a, b]$ for further navigation through the network. Until that moment, we continue our

explanations. The strings obtained in $x_2^{(\neq B)}(a, b)$ enter $x_2^{(\neq B)}(b)$, where W_b is appended to their right-hand end. Now, all these strings enter $x_4^{(\neq B)}$, where exactly one occurrence of each letter $c \in V_2 \setminus \{B\}$ is replaced by Z_c . The role of this node is to check whether or not $\alpha = \varepsilon$ since a move of the tape head to the left in the ID $\alpha q \beta$ is possible provided that $\alpha \neq \varepsilon$. More clearly, $C_{m+5}^{(w)}(x_4^{(\neq B)})$ has just received all strings of the form $\beta_1 \langle s, b, R, a \rangle \alpha W_b$ and $\beta \langle s, b, R, a \rangle \alpha_1 W_b$, where β_1 and α_1 differ from β and α , respectively, on exactly one position where a in β or α is replaced by $[a, b]$.

Table 2

Node	M	PI	FI	PO	FO	α	β
$x_2^{(\neq B)}(a, b)$	$\{a \rightarrow [a, b]\}$	$\{\langle s, b, L, a \rangle\}$	$U \setminus (V_2 \cup U_1^{(L)})$	U_2	\emptyset	*	(4)
$x_2^{(\neq B)}(b)$	$\{\varepsilon \rightarrow W_b\}$	$\{[a, b]\}$	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2)$	\emptyset	\emptyset	r	(1)
$x_4^{(\neq B)}$	$\{a \rightarrow Z_a\}$	U_6	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2 \cup U_6)$	U_5	\emptyset	*	(4)
$x_5^{(\neq B)}$	$\{[a, b] \rightarrow \varepsilon\}$	U_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2 \cup U_5 \cup U_6)$	$U \setminus U_2$	U_2	l	(4)
$x_3^{(\neq B)}(a)$	$\{\varepsilon \rightarrow W'_a\}$	$\{W_a\}$	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U_6)$	$U \setminus U_3$	U_3	l	(4)
$x_6^{(\neq B)}$	$\{W_a \rightarrow \varepsilon\}$	U'_6	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U_6 \cup U'_6)$	$U \setminus U_6$	U_6	r	(4)
$x_4^{(\neq B)}(a)$	$\{\varepsilon \rightarrow Z'_a\}$	$\{Z_a\}$	$U \setminus (V_2 \cup U_1^{(L)})$	$U \setminus U_3$	U_3	l	(4)
$x_7^{(\neq B)}$	$\{Z_a \rightarrow \varepsilon\}$	U'_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U'_5 \cup U'_6)$	$U \setminus U_5$	U_5	r	(4)
$x_8^{(\neq B)}$	$\{W'_a \rightarrow a\} \cup$ $\{Z'_a \rightarrow a\} \cup$ $\{\langle s, b, L, a \rangle \rightarrow s\}$	U'_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U'_5 \cup U'_6)$	$U \setminus (U_1^{(L)} \cup U'_5 \cup U'_6)$	$U_1^{(L)} \cup$ $U'_5 \cup U'_6$	*	(4)

Now, $C_{m+6}^{(w)}(x_4^{(\neq B)})$ contains all strings $h(\beta_1) \langle s, b, L, a \rangle \alpha W_b$, $\beta_1 \langle s, b, L, a \rangle h(\alpha) W_b$, $\beta \langle s, b, L, a \rangle h(\alpha_1) W_b$, $h(\beta) \langle s, b, L, a \rangle \alpha_1 W_b$, where $h : ((V_2 \setminus \{B\}) \cup U_2 \cup U_6 \cup U_1^{(L)})^* \rightarrow 2^{((V_2 \setminus \{B\}) \cup U_2 \cup U_6 \cup U_1^{(L)} \cup U_5)^*}$ is a finite substitution which leaves unchanged all the symbols from $U_2 \cup U_6 \cup U_1^{(L)}$ and $h(c) = \{c, Z_c\}$, for all $c \in V_2 \setminus \{B\}$. But $C_{m+6}^{(w)}(x_4^{(\neq B)})$ contains no string $\beta_1 \langle s, b, L, a \rangle \alpha W_b$ or $\beta \langle s, b, L, a \rangle \alpha_1 W_b$ from above. Later, it will turn out that only the strings $[a, b] \beta' \langle s, b, L, a \rangle \alpha' Z_c W_b$ are useful for the rest of computation. Indeed, the strings which do not start with a symbol in U_2 remain blocked in $x_5^{(\neq B)}$. The others leave $x_5^{(\neq B)}$ and enter $x_3^{(\neq B)}(a)$ where they receive W'_a in their left-hand end, provided that they have W_a in their right-hand end. After that, W_a is deleted. This is actually the way of rotating a symbol from the right-hand end to the left-hand end of a string. The role of $x_4^{(\neq B)}(a)$ and $x_7^{(\neq B)}$ is the same and now we can easily notice that only the strings proceeding from $[a, b] \beta' \langle s, b, L, a \rangle \alpha' Z_c W_b$ can continue the computation. Finally, we deduce that

$C_{m+22}^{(w)}(x_1^{(\neq B)}) \supseteq \{cb\beta's\alpha' \mid \alpha = c\alpha', (s, b, L) \in \delta(q, a), s \in Q, b, c \in V_2 \setminus \{B\}\}$.
The nodes described in Table 3 are used for simulating a move of M which consists in reading B and changing it into a symbol from $V_2 \setminus \{B\}$, possibly changing the current state, and moving the tape head to the right. In this table, $s, q \in Q$, $a \in V_2 \setminus \{B\}$ and $K \in \{R, L\}$.

Table 3

Node	M	PI	FI	PO	FO	α	β
$x_1^{(=B)}$	$\{\varepsilon \rightarrow \langle\langle s, a, K, q \rangle\rangle \mid (s, a, K) \in \delta(q, B)\}$	\emptyset	$U \setminus (V_2 \cup Q)$	\emptyset	\emptyset	r	(1)
$x_1^{(=B)}(q)$	$\{q \rightarrow \varepsilon\}$	$\{\langle\langle s, a, K, q \rangle\rangle\}$	$U \setminus (V_2 \cup U_9 \cup Q)$	U	\emptyset	l	(4)
$x_1^{(=B)}(s, a)$	$\{\varepsilon \rightarrow \bar{s}a\}$	$\{\langle\langle s, a, R, q \rangle\rangle\}$	$U \setminus (V_2 \cup U_9^{(R)})$	\emptyset	\emptyset	l	(1)
$x_1^{(=B)}(a)$	$\{\varepsilon \rightarrow Y_a\}$	$\{\bar{s}a\}$	$U \setminus (V_2 \cup U_9^{(R)} \cup U_7)$	U	\emptyset	r	(4)
$x_2^{(=B)}$	$\{\langle\langle s, a, R, q \rangle\rangle \rightarrow \varepsilon\}$	U_8	$U \setminus (V_2 \cup U_9^{(R)} \cup U_7 \cup U_8)$	\emptyset	$U_9^{(R)}$	$*$	(1)
$x_3^{(=B)}$	$\{Y_a \rightarrow a\} \cup \{\bar{s}a \rightarrow s\}$	U_8	$U \setminus (V_2 \cup U_8 \cup U_7)$	$U \setminus (U_8 \cup U_7)$	$U_8 \cup U_7$	$*$	(4)

We consider a string $\beta q \alpha \in C_m^{(w)}(x_1^{(=B)})$ and $(s, a, R) \in \delta(q, B)$ a transition which the move of M we want to simulate is based on. First, $\beta q \alpha \langle\langle s, a, R, q \rangle\rangle$ is produced in $x_1^{(=B)}$ and then sent out. The string enters $x_1^{(=B)}(q)$ where one checks whether or not $\beta = \varepsilon$. Only $q \alpha \langle\langle s, a, R, q \rangle\rangle$, after deleting q , is able to leave $x_1^{(=B)}(q)$, the others being blocked in this node. Now, $\alpha \langle\langle s, a, R, q \rangle\rangle$ enters $x_1^{(=B)}(s, a)$, where the symbol $\bar{s}a$ is appended to its left-hand end, and the resulting string enters $x_1^{(=B)}(a)$, where Y_a is appended to its right-hand end.

Table 4

Node	M	PI	FI	PO	FO	α	β
$x_2^{(=B)}(s, a)$	$\{\varepsilon \rightarrow \bar{s}a\}$	$\{\langle\langle s, a, L, q \rangle\rangle\}$	$U \setminus (V_2 \cup U_9^{(L)})$	U	\emptyset	l	(4)
$x_4^{(=B)}$	$\{\langle\langle s, a, L, q \rangle\rangle \rightarrow \varepsilon\}$	U_7	$U \setminus (V_2 \cup U_9^{(L)} \cup U_{10})$	$U \setminus U_9^{(L)}$	$U_9^{(L)}$	$*$	(4)
$x_2^{(=B)}(a)$	$\{\varepsilon \rightarrow X'_a\}$	$\{\bar{s}a\}$	$U \setminus (V_2 \cup U_{10})$	U	\emptyset	l	(4)
$x_5^{(=B)}$	$\{a \rightarrow Y'_a\}$	U'_3	$U \setminus (V_2 \cup U_{10} \cup U'_3)$	U'_8	\emptyset	$*$	(4)
$x_3^{(=B)}(a)$	$\{\varepsilon \rightarrow a'\}$	$\{Y'_a\}$	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup U'_8)$	\emptyset	\emptyset	l	(1)
$x_6^{(=B)}$	$\{Y'_a \rightarrow \varepsilon\}$	V'_2	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup U'_8 \cup V'_2)$	$U \setminus U'_8$	U'_8	r	(4)
$x_7^{(=B)}$	$\{X'_a \rightarrow a\} \cup \{\bar{s}a \rightarrow s\} \cup \{a' \rightarrow a\}$	U'_3	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup V'_2)$	$U \setminus (V'_2 \cup U'_3 \cup U_{10})$	$V'_2 \cup U'_3$	$*$	(4)

Then, $\langle\langle s, a, R, q \rangle\rangle$ is removed and Y_a , as well as \overline{sa} , are replaced by a and s , respectively. Hence

$$C_{m+14}^{(w)}(x_1^{(=B)}) \supseteq \{s\alpha a \mid (s, a, R) \in \delta(q, B), s \in Q, a \in V_2 \setminus \{B\}\}.$$

The nodes described in Table 4 are used, together with the nodes $x_1^{(=B)}$ and $x_1^{(=B)}(q)$, $q \in Q$, for simulating a move of M which consists in reading B and changing it into a symbol from $V_2 \setminus \{B\}$, possibly changing the current state, and moving the tape head to the left. In this table, $s, q \in Q$ and $a \in V_2 \setminus \{B\}$.

We consider again a string $\beta q \alpha \in C_m^{(w)}(x_1^{(=B)})$ and $(s, a, L) \in \delta(q, B)$ a transition which the move of M we want to simulate is based on. As above, after producing $\beta q \alpha \langle\langle s, a, L, q \rangle\rangle$ in $x_1^{(=B)}$, this string enters $x_1^{(=B)}(q)$, where one checks whether or not $\beta = \varepsilon$ and q is removed. Then, $\alpha \langle\langle s, a, L, q \rangle\rangle$ enters $x_2^{(=B)}(s, a)$, where \overline{sa} is appended to its left-hand end. The new string, after having removed $\langle\langle s, a, L, q \rangle\rangle$ receives X'_a in its left-hand end resulting in $X'_a \overline{sa} \alpha$. Now, the last symbol of α , say b , is shifted as b' before X'_a by means of the nodes $x_5^{(=B)}$, $x_6^{(=B)}$, and $x_3^{(=B)}(b)$. The obtained string is now $b'X'_a \overline{sa} \alpha'$, with $\alpha = \alpha'b$. Therefore,

$$C_{m+22}^{(w)}(x_1^{(=B)}) \supseteq \{bas\alpha' \mid (s, a, L) \in \delta(q, B), s \in Q, a \in V_2 \setminus \{B\}, \alpha = \alpha'b\}.$$

The construction of Γ is completed with the output node x_O defined by $M_{x_O} = \emptyset$, $PI_{x_O} = F$, $FI_{x_O} = U \setminus (V_2 \cup F)$, $PO_{x_O} = \emptyset$, $FO_{x_O} = U$, $\alpha(x_O) = *$, $\beta(x_O) = (4)$. By the aforementioned explanations, we infer that $L(M) = L(\Gamma)$.

It is worth mentioning that the underlying graph G is the complete graph K_p , with $p = 15 + 7(\text{card}(V_2) - 1) + \text{card}(Q) + 2(\text{card}(V_2) - 1)^2 + 2\text{card}(Q)(\text{card}(V_2) - 1)$. That is, the number of nodes of Γ is bounded by a quadratic function depending on the number of states and symbols of M . Also, the total number of symbols used by Γ in the above simulation is bounded by a cubic function depending on the number of states and symbols of M . More precisely,

$$\begin{aligned} \text{card}(U) &= 4\text{card}(Q)(\text{card}(V_2) - 1)^2 + 2(\text{card}(V_2) - 1)^2 + \text{Card}(V_2) + \\ &2\text{card}(Q)(\text{card}(V_2) - 1) + 9(\text{card}(V_2) - 1) + \text{card}(Q) \quad \square \end{aligned}$$

Now we are ready to prove the main result of this paper.

Theorem 1. $\mathbf{NP} = \mathbf{PTime}_{AHNEP}$.

Proof. Let L be a language accepted by a nondeterministic Turing machine M with k tapes such that for each $x \in L$, $|x| = n$, M can accept x in no more than $p(n)$ moves. We write this as $T_M(n) \leq p(n)$. Clearly, we can construct a Turing machine M' such that $T_{M'}(n) \leq p(n)/\sqrt{22}$. By the well-known results regarding tape compression, we can construct a Turing machine M'' with one tape only, such that $T_{M''}(n) \leq p^2(n)/22$. Now, by the previous proof, we construct an AHNEP Γ such that $L(M'') = L(\Gamma)$ and $\text{Time}_\Gamma(n) \leq 22T_{M''} \leq p^2(n)$, which concludes the proof of $\mathbf{NP} \subseteq \mathbf{PTime}_{AHNEP}$.

Conversely, let L be a language accepted by an AHNEP Γ in polynomial time $p(n)$. We construct a nondeterministic Turing machine M as follows:

(1) M has a finite set of states associated with each node of Γ . This set is divided into disjoint subsets such that each filter (input or output) and each rule has an associated subset of states.

(2) M chooses nondeterministically a copy of the input word from those existing in the initial node of Γ (this word is actually on the tape of M in its initial ID) and follows 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 labelled by (*):

(i) When M enters a state from the subset of states associated to a rule $a \rightarrow b$, it applies this rule to an occurrence of a in α , if any, nondeterministically chosen. If α does not contain any occurrence of a , 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 does not pass it, M blocks the computation. Clearly, M checks first the condition of the current node (sending node) output filter and then the condition of the receiving node input filter (which becomes the current node).

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

It is rather plain that M accepts L . If the input word w in the initial node of Γ is in L , then there exists a computation in Γ of time complexity $O(p(|w|))$. Since in any evolutionary step one inserts at most one letter, the length of α in (*) is at most $p(|w|) + |w|$. Clearly, each step (i) and (ii) of (*) can be accomplished in time $O(|\alpha|)$. Therefore, w is accepted by M in $O(p^2(|w|))$ time and we are done. \square

References

1. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors, *IWANN 2001* (J. Mira, A. Prieto, eds.), LNCS 2084, Springer-Verlag, 2001, 621–628.
2. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, *Acta Informatica* 39 (2003), 517–529.
3. J. Castellanos, P. Leupold, V. Mitrana, Descriptive and computational complexity aspects of hybrid networks of evolutionary processors, submitted.
4. E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors. In: *New Trends in Formal Languages* (Gh. Păun, A. Salomaa, eds.), LNCS 1218, Springer Verlag, 1997, 299–318.
5. E. Csuhaj-Varjú, V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* 36 (2000), 913–926.
6. L. Errico, C. Jesshope, Towards a new architecture for symbolic processing. In *Artificial Intelligence and Information-Control Systems of Robots '94* (I. Plander, ed.), World Sci. Publ., Singapore, 1994, 31–40.
7. S. E. Fahlman, G. E. Hinton, T. J. Sejnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.
8. J. Hartmanis, P.M. Lewis II, R.E. Stearns, Hierarchies of memory limited computations. *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, 1965, 179 - 190.
9. J. Hartmanis, R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* 117 (1965), 533–546.

10. W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.
11. C. Martin-Vide, V. Mitrana, M. Perez-Jimenez, F. Sancho-Caparrini, Hybrid networks of evolutionary processors, *Proc. of GECCO 2003*, LNCS 2723, Springer Verlag, Berlin, 401 - 412.
12. D. Sankoff et al. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, 89 (1992) 6575–6579.