A New Characterization of NP, P, and PSPACE with Accepting Hybrid Networks of Evolutionary Processors

Florin Manea · Maurice Margenstern · Victor Mitrana · Mario J. Pérez-Jiménez

Abstract We consider three complexity classes defined on Accepting Hybrid Networks of Evolutionary Processors (AHNEP) and compare them with the classical 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 family of languages decided by AHNEPs in polynomial time. A language is in **P** if and only if it is decided by an AHNEP in polynomial time and space. We also show that **PSPACE** equals the family of languages decided by AHNEPs in polynomial length.

Keywords Evolution strategies · Evolutionary processor · Network of evolutionary processors · Turing machine · Computational complexity classes

A preliminary version of this work entitled *Accepting Hybrid Networks of Evolutionary Processors* was presented in *10th International Workshop on DNA Computing (DNA 10)*.

F. Manea · V. Mitrana (✉)
Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 010014 Bucharest, Romania
e-mail: mitrana@fmi.unibuc.ro

F. Manea
e-mail: flmanea@gmail.com

M. Margenstern
LITA, UFR MIM, University of Metz, Ile du Saulcy, 57045 Metz-Cedex, France
e-mail: margens@univ-metz.fr

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

M.J. Pérez-Jiménez
Department of Computer Science and Artificial Intelligence, University of Seville, Seville, Spain
e-mail: mario.perez@cs.us.es

1 Introduction

The origin of the networks of evolutionary processors (NEPs for short) is twofold. In [4] 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.

On the other hand, a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [11] as well as the Logic Flow paradigm [7], 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 [8, 11].

In [2] we modify this concept (considered from a formal language theory point of view in [6]) 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 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 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 [15]. 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. We are aware of the fact that modeling genetic evolutionary steps in this simple form is a demanding task requiring more than the system described in this paper.

The computational power of the device proposed in [2], viewed as a language generating device (Generating Hybrid Networks of Evolutionary Processors—GHNEP

for short), has been further investigated in the papers [3] and [5]; it turned out to be computationally complete. In a series of papers, we present *linear* time solutions to some NP-complete problems using these rather simple mechanisms. Such solutions are presented for the Bounded Post Correspondence Problem [2], for the 3-colorability problem [3], and for the Common Algorithmic Problem [14]. It is worth mentioning that the GHNEPs solving the aforementioned problems have all resources (size, number of rules and symbols) linearly bounded by the size of the given instance. Work in [1] is devoted to the study of a descriptional complexity measure of these networks, namely the size.

In [12], we propose two linear time solutions for two much celebrated NP-complete problems, namely the 3CNF-SAT and the HPP (Hamiltonian Path Problem), based on Accepting Networks of Evolutionary Processors (AHNEPs for short) having all the resources (size, number of rules and symbols) linearly bounded by the size of the given instance of the problems. This paper presented for the first time such solutions based on AHNEPs, and not GHNEPs as [3], and more important, one rigorously defined problem solvers based on AHNEPs. By the definition of this model, one can evaluate the descriptional (number of nodes, rules, symbols) and computational (time and space) complexity of these AHNEPs with respect to their input string, which is actually the given instance of the problem. Furthermore, an universal AHNEP has been reported in [13].

In this paper, we consider three complexity classes defined on AHNEPs similarly to the classical time and space 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 family of languages decided by AHNEPs in polynomial time. A language is in **P** if and only if it is decided by an AHNEP in polynomial time and space. We also show that **PSPACE** equals the family of languages decided by AHNEPs in polynomial length. It is questionable whether this approach gives something back to the evolutionary genetics studies, but we believe that it gives a little to the complexity theory. Characterizing **NP**, **P**, and **PSPACE** is not sufficient but this work is just a first step in this direction. It is worth mentioning that the general idea of the model is to show that very simple processors (based on pretty simple replacements) working synchronously in parallel and exchanging data to each other under a simple control mechanism (filters based on the symbol presence and absence) are able to efficiently simulate Turing machines and characterize complexity classes.

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 $x \in W^*$.

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 $a = B$; that is T never writes 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 never goes the left-hand end of the tape. The input string is a string over V and is stored on the tape starting with the leftmost cell and all the other tape cells containing the symbol B.

An *instantaneous description* (ID for short) of a Turing machine T as above is a string in $(U \setminus \{B\})^* Q (U \setminus \{B\})^*$. 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.

Initially, the tape head scans the leftmost cell and the central unit is in the state q_0. This is called the initial ID. 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. If no move is possible for the current state and contents of the scanned cell, then we say that the machine is in a halting ID. Note that any ID with a final state is halting. A computation of this machine is any finite or infinite sequence of IDs, starting with the initial ID, each ID being reached from the previous one by one move. We say that a Turing machine always halts if all computations on every input eventually reach a halting ID. An input string is *accepted* by a Turing machine if there exists a computation that eventually reaches a final state. A language L is decided by a Turing machine M if M always halts and accepts exactly all words of L.

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

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

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

– If $\sigma \equiv \varepsilon \to a \in Ins_V$, then

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

$\alpha \in \{*, l, r\}$ 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:

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

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 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 in the Introduction, 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 [15].

An *accepting hybrid network of evolutionary processors* (AHNEP 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:

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

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. Networks of evolutionary processors, seen as language generating devices, with underlying graphs having these special forms have been considered in several papers [1–3, 14]. We focus here on *complete* AHNEPs (hence accepting devices and not generating ones as those considered in the papers cited above), i.e., AHNEPs having a complete underlying graph denoted by K_n, where n is the number of vertices.

A *configuration* of an 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

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

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

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \quad \text{for all } x \in X_G.$$

Let Γ be an AHNEP, the computation of Γ on the input string $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, 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 $i \geq 0$. Note that the configurations are changed by alternative steps. 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 *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 Γ is

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

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

3 Complexity Classes

The reader is referred to [9, 10] 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 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 $\mathbf{N}$,

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

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

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

Moreover, we write

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

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:

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

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

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

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

$$\mathbf{Space}_{AHNEP}(f(n)) = \{L \mid \text{ there exists an } AHNEP\ \Gamma \text{ which decides } L \text{ and } n_0 \text{ such that } \forall n \geq n_0 (Space_\Gamma(n) \leq f(n))\}.$$

Moreover, we write

$$\mathbf{PSpace}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Space}_{AHNEP}(n^k).$$

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 $\mathbf{N}$,

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

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

$$\mathbf{Length}_{AHNEP}(f(n)) = \{L \mid \text{ there exists an } AHNEP\ \Gamma \text{ which decides } L \text{ and } n_0 \text{ such that } \forall n \geq n_0 (Length_{\Gamma}(n) \leq f(n))\}.$$

Moreover, we write

$$\mathbf{PLength}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Length}_{AHNEP}(n^k).$$

4 Relationships Between the Complexity Classes Defined on Turing Machines and AHNEPs

Now we prove a first result which will eventually show a strong connection between the complexity classes defined on Turing machines and those defined on AHNEPs. It is worth emphasizing that the next result states the computational completeness of AHNEPs; a similar result is presented in [5] for GHNEPs but the proof in [5], based on a simulation of a phrase-structure grammar in the Geffert normal form, is completely different than the proof given here.

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

Proof Let $M = (Q, V_1, V_2, \delta, q_0, B, F)$ be an arbitrary Turing machine. We intend to construct an AHNEP Γ that, for every input string w, simulates the computation of M on w. First, we define the new alphabets:

$$U_1^{(K)} = \{\langle s, b, K, a\rangle \mid (s, b, K) \in \delta(q, a), s, q \in Q, a, b \in V_2 \setminus \{B\}\}, \quad K \in \{R, L\},$$
$$U_2 = \{[a, b] \mid a, b \in V_2 \setminus \{B\}\}, \qquad U_3 = \{X_a \mid a \in V_2 \setminus \{B\}\},$$
$$U_4 = \{Y_a^{(b)} \mid a, b \in V_2 \setminus \{B\}\}, \qquad U_5 = \{Z_a \mid a \in V_2 \setminus \{B\}\},$$
$$U_6 = \{W_a \mid a \in V_2 \setminus \{B\}\}, \qquad U_7 = \{\overline{sa} \mid s \in Q, a \in V_2 \setminus \{B\}\},$$
$$U_8 = \{Y_a \mid a \in V_2 \setminus \{B\}\}, \qquad U_{10} = \{\tilde{sa} \mid s \in Q, a \in V_2 \setminus \{B\}\},$$
$$U_9^{(K)} = \{\langle\langle s, a, K, q\rangle\rangle \mid (s, a, K) \in \delta(q, B), s, q \in Q, a \in V_2 \setminus \{B\}\}, \quad K \in \{R, L\}.$$

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

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

We define the AHNEP $\Gamma = (V_1, U, K_p, N, \alpha, \beta, x_I, x_O)$, where $p = 15 + 7(card(V_2) - 1) + card(Q) + 2(card(V_2) - 1)^2 + 2card(Q)(card(V_2) - 1)$ nodes.

Table 1 Simulation of a non-B-reading move of M to the right

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

To begin with, the evolutionary processor placed in the input node x_I is defined as follows: $M_{x_I} = \{\varepsilon \to q_0\}$, $PI_{x_I} = \emptyset$, $FI_{x_I} = U$, $PO_{x_I} = \emptyset$, $FO_{x_I} = \emptyset$, $\alpha(x_I) = r$, $\beta(x_I) = s$.

The definitions of the evolutionary processors placed in the rest of the nodes are presented in the following way: we group these definitions in 4 tables, according to the role they play in the simulation of the Turing Machine M. Each table is accompanied by some explanations which emphasize this role.

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 $K \in \{R, L\}$.

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,

$$C_{m+1}^{(w)}(x_1^{(\neq B)}) \supseteq \{\beta\langle s, b, K, c\rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}.$$

It is clear that 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:

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

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 continue our explanation by returning to the configuration

$$C_{m+1}^{(w)}(x_1^{(\neq B)}) \supseteq \{\beta\langle s, b, K, c\rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}.$$

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 the letter $c \in V_2 \setminus \{B\}$ in different copies of each of these strings is replaced by Z_c. As we shall

Table 2 Simulation of a non-B-reading move of M to the left

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

see later, only those words in which the rightmost occurrence of c is replaced by Z_c can be further processed. The role of this node is to check whether or not $\alpha = \varepsilon$ since

Table 3 Simulation of a B-reading move of M to the right

Node	$M, PI, FI, PO, FO, \alpha, \beta$
$x_1^{(=B)}$	$M = \{\varepsilon \to \langle\langle s, a, K, q\rangle\rangle \mid (s, a, K) \in \delta(q, B)\}$ $PI = \emptyset, FI = U \setminus (V_2 \cup Q)$ $PO = \emptyset, FO = \emptyset$ $\alpha = r, \beta = s$
$x_1^{(=B)}(q)$	$M = \{q \to \varepsilon\}$ $PI = \{\langle\langle s, a, K, q\rangle\rangle\}, FI = U \setminus (V_2 \cup U_9 \cup Q)$ $PO = U\ , FO = \emptyset$ $\alpha = l, \beta = w$
$x_1^{(=B)}(s, a)$	$M = \{\varepsilon \to \overline{sa}\}$ $PI = \{\langle\langle s, a, R, q\rangle\rangle\}, FI = U \setminus (V_2 \cup U_9^{(R)})$ $PO = \emptyset, FO = \emptyset$ $\alpha = l, \beta = s$
$x_1^{(=B)}(a)$	$M = \{\varepsilon \to Y_a\}$ $PI = \{\overline{sa}\}, FI = U \setminus (V_2 \cup U_9^{(R)} \cup U_7)$ $PO = U, FO = \emptyset$ $\alpha = r, \beta = w$
$x_2^{(=B)}$	$M = \{\langle\langle s, a, R, q\rangle\rangle \to \varepsilon\}$ $PI = U_8, FI = U \setminus (V_2 \cup U_9^{(R)} \cup U_7 \cup U_8)$ $PO = \emptyset, FO = U_9^{(R)}$ $\alpha = *, \beta = s$
$x_3^{(=B)}$	$M = \{Y_a \to a\} \cup \{\overline{sa} \to s\}$ $PI = U_8, FI = U \setminus (V_2 \cup U_8 \cup U_7)$ $PO = U \setminus (U_8 \cup U_7), FO = U_8 \cup U_7$ $\alpha = *, \beta = w$

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 $\beta_1\langle s, b, L, a\rangle\alpha W_b$ and $\beta\langle s, b, L, a\rangle\alpha_1 W_b$, 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 $[a, b]$.

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 $K \in \{R, L\}$.

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, the string $\beta q \alpha \langle\langle s, a, R, q\rangle\rangle$ 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 $\overline{sa}$ 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. 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 $\tilde{sa}$ 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 $X'_a \tilde{sa} \alpha$. 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 $b' X'_a \tilde{sa} \alpha'$, 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 $M_{x_O} = \emptyset$, $PI_{x_O} = F$, $FI_{x_O} = U \setminus (V_2 \cup F)$, $PO_{x_O} = \emptyset$, $FO_{x_O} = U$, $\alpha(x_O) = *$, $\beta(x_O) = w$. By the aforementioned explanations we infer that $L(M) = L(\Gamma)$. On the other hand, if M halts on w without accepting, namely all computations of M on w eventually reach deadlock states, then the computation of Γ on w is halting but not accepting. Further, if M works in $f(n)$ time, then $Time_\Gamma(n) \leq 22 f(n)$. □

Recall that the underlying graph of the network Γ is the complete graph K_p, with

$$p = 15 + 7(card(V_2) - 1) + card(Q) + 2(card(V_2) - 1)^2 + 2card(Q)(card(V_2) - 1).$$

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

Table 4 Simulation of a B-reading move of M to the left

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

Γ is the above simulation is bounded by a cubic function depending on the number of states and symbols of M. More precisely,

$$card(U) = 4card(Q)(card(V_2) - 1)^2 + 2(card(V_2) - 1)^2 + Card(V_2) + 2card(Q)(card(V_2) - 1) + 9(card(V_2) - 1) + card(Q).$$

Theorem 1

1. $\mathbf{NP} \subseteq \mathbf{PTime}_{AHNEP}$.
2. $\mathbf{PSPACE} \subseteq \mathbf{PLength}_{AHNEP}$.

Proof Let L be a language decided 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 $Time_\Gamma(n) \leq 22T_{M''} \leq p^2(n)$, which concludes the proof of $\mathbf{NP} \subseteq \mathbf{PTime}_{AHNEP}$.

As $\mathbf{NPSPACE} = \mathbf{PSPACE}$, the second statement follows immediately. □

Theorem 2

1. $\mathbf{PTime}_{AHNEP} \subseteq \mathbf{NP}$.
2. $\mathbf{PLength}_{AHNEP} \subseteq \mathbf{PSPACE}$.

Proof 1. Let L be a language decided by an AHNEP Γ in polynomial time $p(n)$. It is sufficient to construct a nondeterministic Turing machine that does not necessarily halt on every input but accepts L in polynomial time. Such a machine M may be constructed 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 string from those existing in the initial node of Γ (this string 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 labeled by $(*)$:

(i) When M enters a state from the subset of states associated to a rule $a \to b$, 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 string.

It is rather plain that M accepts L. If the input string 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 which concludes the proof of the first statement.

2. From this reasoning it also follows that the working space of M on every input string of length n is bounded by $Length_\Gamma(n)$.

Now, the main result of this paper follows.

Theorem 3

1. $\mathbf{PTime}_{AHNEP} = \mathbf{NP}$.
2. $\mathbf{PLength}_{AHNEP} = \mathbf{PSPACE}$.

Before going on, we want to say a few words about the last result. It may be viewed a bit unfair if one considers that a computation of a nondeterministic Turing machine could be defined as a sequence of sets of IDs. In this setting, the machine may be viewed as computing deterministically. However, this is not a natural definition comparing to our approach: it is considered to be biologically feasible to have sufficiently many identical copies of a molecule. By techniques of genetic engineering, in a polynomial number of lab operations one can get an exponential number of identical molecules.

In the sequel we propose a characterization of the class **P** of languages that are recognized by Turing machines in polynomial time in terms AHNEPs working in polynomial time and space.

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

Proof "if" Let $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$ with $G = (X_G, E_G)$. We describe an algorithm for deciding whether or not a given string is in L. We use two tables $(C(x))_{x \in X_G}$ and $(D(x))_{x \in X_G}$, with the following interpretation: $C(x)$ represents the set of strings that are contained in the node x after a communication step, and $D(x)$ represents the set of strings that are contained in the node x after an evolutionary step. The algorithm works as follows:

Algorithm 1
INPUT: $w \in V^*$
OUTPUT: **accept** *iff* $w \in L$
begin
for *every* $x \in X_G \setminus \{x_I\}$ do
  $D(x) := \emptyset$; $C(x) := \emptyset$;
endfor;
$D(x_I) := \emptyset$; $C(x_I) := \{w\}$;
$i := 0$;
while $i \leq Q(|w|)$
  for *every* $x \in X_G$
    $D(x) := M_x^{\alpha(x)}(C(x))$;
  endfor;
  if $C(x_O) \neq \emptyset$ then **accept**; halt;
  $i := i + 1$;
  for *every* $x \in X_G$
    $C(x) := (D(x) - \tau_x(D(x))) \cup \bigcup_{y \in E_G} (\tau_y(D(y)) \cap \rho_x(D(y))), x \in X_G$;
  endfor;
  if $C(x_O) \neq \emptyset$ then **accept**; halt;
  $i := i + 1$;

endwhile;
reject;
end.

The working space of this algorithm is polynomial because the number of strings contained in $\bigcup_{x \in X_G} C(x)$ and $\bigcup_{x \in X_G} D(x)$ is bounded by $P(n)$ in any step of the algorithm. Moreover, every string in these sets is of a length bounded by $|w| + Q(|w|)$, since in any step of the algorithm a string can become longer with at most one symbol.

The time needed to compute $M_x^{\alpha(x)}(C(x))$, $x \in X_G$, is at most $K(|w| + Q(|w|)) \cdot P(|w|)$, where K is the maximum number of rules in a node. This assertion holds because, in the most time consuming case, in every node all the evolutionary rules can be applied on any occurrence of a symbol (in the case of deletion or substitution) or any position in the string (in the case of insertion). Note that this computation can produce at most $P(|w|)$ valid strings that are stored in the table D. Note also that checking whether a string from $D(x)$, $x \in X_G$, can pass a filter takes $\mathcal{O}(T(|w| + Q(|w|)))$ time, where T is the maximal cardinality of a set filter. It follows that, in every iteration of the while loop, $C(x)$ and $D(x)$ can be computed in $\mathcal{O}((n + Q(n))P(n))$ time provided that $|w| = n$. The soundness of the algorithm is easy to be observed. Concluding, it is not hard to deduce that the total time needed by the algorithm presented above is: $\mathcal{O}((n + Q(n))P(n)Q(n))$.

Also it is important to note that the algorithm can be implemented on a Turing Machine, with 4 tapes: on the first tape there are encoded the filters and rules of each node separated by some marker. Tables C and D are stored on the second and third tape, respectively. Finally the value of variable i is stored on the forth tape together with the value of $Q(n)$. The computing strategy follows the facts presented above, D being computed by rewriting the third tape according to the strings present on the first and second tapes; C is computed in a similar manner, but according to the strings on the first and third tapes. The computational time needed for such a Turing Machine is a polynomial, hence $L \in \mathbf{P}$.

For the "only if" part we make the following considerations concerning the proof of Proposition 1. Assume that the Turing machine M is deterministic and decides L in time $p(n)$, where p is a polynomial. Further let $m = card(V_2) - 1$. We start noting that in any computational step of Γ there is exactly one node containing a single string that is useful for the computation all the other strings, if any, being useless for the computation. Clearly, Γ works in time $\mathcal{O}(p(n))$. We have to calculate the number of useless strings that can be accumulated in different nodes. As M works in time $p(n)$, all strings in the nodes of Γ at any moment are of length $\mathcal{O}(p(n))$. The nodes of Γ that can collect useless strings are:

- $x_3^{\neq B}$: it can receive $\mathcal{O}(m \cdot p(n))$ strings in any step, therefore a total amount of $\mathcal{O}(m \cdot p^2(n))$ strings during the whole computation.
- $x_5^{\neq B}$: it can receive $\mathcal{O}(m^3 \cdot p^2(n))$ strings in any step, therefore a total amount of $\mathcal{O}(m^3 \cdot p^3(n))$ strings during the whole computation.
- $x_7^{\neq B}$: it can receive $\mathcal{O}(m \cdot p(n))$ strings in any step, therefore a total amount of $\mathcal{O}(m \cdot p^2(n))$ strings during the whole computation.
- $x_1^{=B}(q)$, $q \in Q$: it can receive $\mathcal{O}(p(n))$ strings in any step, therefore a total amount of $\mathcal{O}(p^2(n))$ strings during the whole computation.

– $x_6^{=B}$: it can receive $\mathcal{O}(m \cdot p(n))$ strings in any step, therefore a total amount of $\mathcal{O}(m \cdot p^2(n))$ strings during the whole computation.

□

It is worth mentioning that the last theorem does not say that the inclusion $\mathbf{PSpace}_{AHNEP} \cap \mathbf{PTime}_{AHNEP} \subseteq \mathbf{P}$ holds. 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 $\mathbf{PTime}_{AHNEP}$; 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 AHENP that uses polynomial space (but exponential time as well). This shows that 3-CNF-SAT is in $\mathbf{PTime}_{AHNEP} \cap \mathbf{PSpace}_{AHNEP}$, but it is not in $\mathbf{P}$, unless $\mathbf{P} = \mathbf{NP}$.

Acknowledgements We thank to an anonymous reviewer whose comments improved the presentation.

References

1. Castellanos, J., Leupold, P., Mitrana, V.: On the size complexity of hybrid networks of evolutionary processors. Theor. Comput. Sci. **330**, 205–220 (2005)
2. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.: Solving NP-complete problems with networks of evolutionary processors. In: International Work-Conference on Artificial and Natural Neural Networks (IWANN 2001), LNCS, vol. 2084, pp. 621–628. Springer, Berlin (2001)
3. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.: Networks of evolutionary processors. Acta Inf. **39**, 517–529 (2003)
4. Csuhaj-Varjú, E., Mitrana, V.: Evolutionary systems: a language generating device inspired by evolving communities of cells. Acta Inf. **36**, 913–926 (2000)
5. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V.: Hybrid NEPs are computationally complete. Acta Inf. **41**, 257–272 (2005)
6. Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In: Păun, G., Salomaa, A. (eds.) New Trends in Formal Languages, LNCS, vol. 1218, pp. 299–318. Springer, Berlin (1997)
7. Errico, L., Jesshope, C.: Towards a new architecture for symbolic processing. In: Artificial Intelligence and Information-Control Systems of Robots '94, pp. 31–40. World Sci., Singapore (1994)
8. Fahlman, S.E., Hinton, G.E., Seijnowski, T.J.: Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In: Genesereth, M.R. (ed.) Proc. of the National Conference on Artificial Intelligence, pp. 109–113. AAAI Press, Menlo Park (1983)
9. Hartmanis, J., Lewis II, P.L., Stearns, R.: Hierarchies of memory limited computations. In: Proc. 6th Annual Symposium on Foundations of Computer Science, pp. 179–190. IEEE Press, New York (1965)
10. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Trans. Am. Math. Soc. **117**, 533–546 (1965)
11. Hillis, W.D.: The Connection Machine. MIT Press, Cambridge (1985)
12. Manea, F., Martín-Vide, C., Mitrana, V.: Solving 3CNF-SAT and HPP in linear time using WWW. In: Machines, Computations, and Universality (MCU 2004), LNCS, vol. 3354, pp. 269–280. Springer, Berlin (2004)
13. Manea, F., Martín-Vide, C., Mitrana, V.: A universal accepting hybrid network of evolutionary processors. Electron. Not. Theor. Comput. Sci. **135**, 95–105 (2005)
14. Martín-Vide, C., Mitrana, V., Perez-Jimenez, M., Sancho-Caparrini, F.: Hybrid networks of evolutionary processors. In: Genetic and Evolutionary Computation (GECCO 2003), LNCS, vol. 2723, pp. 401–412. Springer, Berlin (2003)
15. Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B.F., Cedergren, R.: Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. Proc. Nat. Acad. Sci. USA **89**, 6575–6579 (1992)