Networks of Polarized Evolutionary Processors Are Computationally Complete

Fernando Arroyo , Sandra Gómez Canaval , Victor Mitrana , and Ştefan Popescu

Abstract. In this paper, we consider the computational power of a new variant of networks of evolutionary processors which seems to be more suitable for a software and hardware implementation. Each processor as well as the data navigating throughout the network are now considered to be polarized. While the polarization of every processor is predefined, the data polarization is dynamically computed by means of a valuation mapping. Consequently, the protocol of communication is naturally defined by means of this polarization. We show that tag systems can be simulated by these networks with a constant number of nodes, while Turing machines can be simulated, in a time-efficient way, by these networks with a number of nodes depending linearly on the tape alphabet of the Turing machine.

1 Introduction

Networks of evolutionary processors (NEP) form a class of highly parallel and distributed computing models inspired and abstracted from the biological evolution. Informally, a network of evolutionary processor consists of a virtual (complete) graph in which each node hosts a very simple processor called evolutionary processor. By an evolutionary processor we mean a mathematical construction which is able to perform very simple operations inspired by the point mutations in DNA sequences (insertion, deletion or substitution of a single base pair). By an informal parallelism with the natural process of evolution, 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 processor, which is specialized just for one of these evolutionary operations, acts on the local data 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.

It is worth mentioning that NEPs resemble a pretty common architecture for parallel and distributed symbolic processing, related to the Connection Machine [7] which was defined as a network of microprocessors in the shape of a hypercube. Each microprocessor was very simple, processing one bit per unit time. Also it is closely related to the tissue-like P systems [13] in the membrane computing area [19].

NEPs as language generating devices and problem solvers have been considered in [2] and [14], respectively. They have been further investigated in a series of subsequent works. NEPs as accepting devices and problem solvers have been considered in [12]; later on, a characterization of the complexity classes **NP**, **P**, and **PSPACE** based on accepting NEPs has been reported in [10]. Universal NEPs and some descriptional complexity problems are discussed in [9]. The reader interested in a survey of the main results regarding NEPs is referred to [11].

Software implementations of NEPs have been reported, see, e.g., [3,4,16], most of them in JAVA. They encountered difficulties especially in the implementation of filters. The main idea to simulate the non-deterministic behavior of NEPs has been to consider a safe-thread model of processors, that is to have each rule and filter in a thread, respectively. Clearly the threads corresponding to the filters are much more complicated than those associated with the evolutionary rules. Configuration changes in a NEP are accomplished either by a communication step or by an evolutionary step, but these two steps may be realized in any order. This suggests that evolution or communication may be chosen depending on the thread model of processor [4]. The input and output filters are implemented as threads extending the `Runnable` interface. Therefore a processor is the parent of a set of threads, which use all objects from that processor in a mutual exclusion region. When a processor starts to run, it starts in a cascade way the rule threads and filter threads. As one can see, the filters associated with processors, especially if there are both input and output filters, seem to be hardly implementable. Consequently, it would be of interest to replace the communication based on filters among processors by another protocol. A first attempt was to move filters from each node to the edges between the nodes, see, e.g., [5]. Although this variant seems to be theoretically simpler, the attempts towards an implementation have encountered similar difficulties due to the fact that the filters associated with edges are similar to those associated with nodes.

Work [1] considers a new variant of NEP with the aim of proposing a new type of filtering process and discusses the potential of this variant for solving hard computational problems. The main and completely new feature of this variant is the valuation mapping which assigns to each string an integer value, depending on the values assigned to its symbols. Actually, we are not interested in computing the exact value of a string, but just the sign of this value. By means of this valuation, one may metaphorically say that the strings are electrically polarized. Thus, if the nodes are polarized as well, the strings migration from one node to another through the channel between the two cells seems to be more natural and easier to be implemented.

We consider here a slightly more general variant of networks of polarized evolutionary processors (NPEP) and investigate its computational power. Although the communication protocol based on the polarized processors and the valuation function seems to offer less control, the new variant is still computationally complete. We show that NPEP with a constant number of processors, namely 15, are computationally complete by devising a method for simulating 2-Tag Systems. As a 2-tag system can efficiently simulate any deterministic Turing machine but not nondeterministic ones, we propose a simulation of nondeterministic Turing machines with NPEP which maintains the working time of the Turing machine. That is, every language accepted by a one-tape nondeterministic Turing machine in time $f(n)$ can be accepted by an NPEP in time $O(f(n))$. Unlike the simulation of a 2-tag system, the size of a NPEP simulating an arbitrary Turing machine depends linearly on the number of tape symbols of the Turing machine.

2 Preliminaries

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

A homomorphism from the monoid V^* into the monoid (group) of additive integers $\mathbf{Z}$ is called *valuation* of V^* in $\mathbf{Z}$.

We consider here the following definition of 2-tag systems that appears in [20]. This type of tag-system, namely the type $\mathcal{T}_2$ 2-tag-systems that appear in Section 8 of [20], is slightly different but equivalent to those from [18,15]. A 2-tag system $T = (V, \mu)$ consists of a finite alphabet of symbols V, containing a special *halting symbol* H (denoted in [20] with *STOP*) and a finite set of rules $\mu : V \setminus \{H\} \to V^+$ such that $|\mu(x)| \geq 2$ or $\mu(x) = H$. Furthermore, $\mu(x) = H$ for just one $x \in V \setminus \{H\}$. A halting word for the system T is a word that contains the halting symbol H or whose length is less than 2. The transformation t_T (called the tag operation) is defined on the set of non-halting words as follows: if x is the leftmost symbol of a non-halting word w, then $t_T(w)$ is the result of deleting the leftmost 2 symbols of w and then appending the word $\mu(x)$ at the right end of the obtained word. A computation by a 2-tag system as above is a finite sequence of words produced by iterating the transformation t, starting with an initially given non-halting word w and halting when a halting word is produced. A computation is not considered to exist unless a halting word is produced in finitely-many iterations. Note that in [20] the halting words are defined a little bit different, as the words starting with the only symbol y such that $\mu(y) = H$, or the words whose length is less than 2. However, our way of defining halting words is equivalent to that in [20], in the sense that there exists a bijection between the valid computations obtained in each of these two cases. Indeed, if we consider the stopping condition from [20], and obtain in a valid computation a word starting with y, thus a halting word, it is enough to apply once more t_T on this word to obtain a word containing H, a halting word according to our definition, and transform the initial valid computation in a valid computation according to our definition. Conversely, if a word containing H, a halting word for our definition, is obtained in a valid computation, then the halting symbol could not have appeared in that word in other way than by applying t_T on a word starting with y, a halting word for the definition from [20], therefore we have a corresponding valid computation, by that definition. As shown in [20], such restricted 2-tag systems are universal.

A nondeterministic Turing machine is a construct $M = (Q, V, U, \delta, q_0, B, F)$, where Q is a finite set of states, V is the input alphabet, U is the tape alphabet, $V \subset U$, q_0 is the initial state, $B \in U \backslash V$ is the "blank" symbol, $F \subseteq Q$ is the set of final states, and δ is the transition mapping, $\delta : (Q \backslash F) \times U \to 2^{Q \times (U \backslash \{B\}) \times \{R,L\}}$. In this paper, we assume without loss of generality that any Turing machine we consider has a semi-infinite tape (bounded to the left) and makes no stationary moves; the computation of such a machine is described in [21,6,17]. An input word is accepted if and only if after a finite number of moves the Turing machine enters a final state. The language accepted by the Turing machine is a set of all accepted words. We say a Turing machine *decides* a language L if it accepts L and moreover halts on every input.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ and $ab \neq \varepsilon$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively. Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

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

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

- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then $\sigma^r(w) = \{wa\}$, $\sigma^l(w) = \{aw\}$.

Note that $\alpha \in \{l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. It is worth mentioning that the action mode of a substitution rule applied to a word w: it returns the set of all words that may be obtained from w depending on the position in w where the rule was actually applied.

For every evolutionary rule σ, action $\alpha \in \{l, r\}$, (α is missing when σ is a substitution rule) 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 $M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

Definition 1. A *polarized evolutionary processor* over V is a pair (M, α, π), where:

- M is a set of substitution, deletion or insertion rules over the alphabet V. Formally: $(M \subseteq Sub_V)$ 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.
- α gives the action mode of the rules of the node. If $M \subseteq Sub_V$, then α is missing.
- $\pi \in \{-, +, 0\}$ is the polarization of the node (negatively or positively charged, or neutral, respectively).

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 compared to evolutionary algorithms, the rewriting operations we have considered here 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 [22].

Definition 2. A *network of polarized evolutionary processors* (NPEP for short) is a 7-tuple $\Gamma = (V, U, G, \mathcal{R}, \varphi, \underline{In}, \underline{Out})$, 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.
- ◇ $\mathcal{R} : X_G \longrightarrow EP_U$ is a mapping which associates with each node $x \in X_G$ the polarized evolutionary processor $\mathcal{R}(x) = (M_x, \alpha_x, \pi_x)$.
- ◇ φ is a valuation of U^* in $\mathbf{Z}$.
- ◇ $\underline{In}, \underline{Out}, \in X_G$ are the *input* and the *output* node of Γ, respectively.

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

A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

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

When changing by a communication step, each node processor $x \in X_G$ sends out copies of all its words but keeping a local copy of the words having the same polarity to that of x only, to all the node processors connected to x and receives a copy of each word sent by any node processor connected with x providing that it has the same polarity as that of x. Note that, for simplicity reasons, we prefer to consider that a word migrate to a node with the same polarity and not an opposed one. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$\begin{aligned} C'(x) = (C(x) \setminus \{w \in C(x) \mid sign(\varphi(w)) \neq \pi_x\}) \ \cup \\ \bigcup_{\{x,y\} \in E_G} (\{w \in C(y) \mid sign(\varphi(w)) = \pi_x\}), \end{aligned}$$

for all $x \in X_G$. Here $sign(m)$ is the sign function which returns $+, 0, -$, provided that m is a positive integer, is 0, or is a negative integer, respectively. Note that all words with a different polarity than that of x are expelled. Further, each expelled word from a node x that cannot enter any node connected to x (no such node has the same polarity as the word has) is lost.

Let Γ be a NPEP, the computation of Γ on the input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ 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 a NPEP is deterministic.

A computation as above *halts*, if there exists a configuration in which the set of words existing in the output node $\underline{Out}$ is non-empty. Given a NPEP Γ and an input word w, we say that Γ accepts w if the computation of Γ on w halts.

Let Γ be a NPEP with the input alphabet V; the *time complexity* of the finite computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Gamma(n) = \sup\{Time_\Gamma(x) \mid |x| = n\}.$$

3 2-Tag Systems Can Be Simulated by NPEP of Constant Size

In the following we show how a 2-tag system can be simulated by an NPEP. We make use of a similar strategy to that developed in [8].

Theorem 1. *For every 2-tag system $T = (V, \mu)$ there exists a NPEP Γ of size 15 such that $L(\Gamma) = \{w \mid \ T \text{ halts on } w\}$.*

Proof. Let $V = \{a_1, a_2, \ldots, a_n, a_{n+1}\}$ be the alphabet of the tag system T with $a_{n+1} = H$ and $V' = V \setminus \{H\}$. Let S be the set of all suffixes of the words in $\{\mu(a) \mid a \in V\}$. We consider the NPEP $\Gamma = (V', U, G, \mathcal{R}, \varphi, 1, 15)$ with the 15 nodes 1, 2, ..., 15. The working alphabet of the network is defined as follows:

$$U = V \cup \{a_0', a_0''\} \cup \{a', \overline{a'}, a'', a^\circ \mid a \in V'\} \cup$$
$$\{[x], \langle x\rangle, \ll x \gg, \overline{\langle x\rangle}, \overline{\ll x \gg}, \langle a_0x\rangle, \ll a_0x \gg, \overline{\langle a_0x\rangle}, \overline{\ll a_0x \gg} \mid x \in S\}.$$

The processors placed in the 15 nodes of the network are defined in Table 1.

Table 1. The description of the nodes of Γ

Node	M	α	π	Adjacency list
1	$\{a \to [\mu(a)] \mid a \in V'\}$		0	$\{2, 14\}$
2	$\{a \to a^\circ \mid a \in V'\}$		+	$\{1, 3\}$
3	$\{\varepsilon \to a_0'\}$	r	−	$\{2, 4, 12\}$
4	$\{a_k' \to a_k'' \mid 0 \le k \le n+1\}$		0	$\{3, 5, 9\}$
5	$\{[a_kx] \to \langle a_{k-1}x\rangle \mid 1 \le k \le n+1, x \in S\} \cup$ $\{\overline{\langle a_kx\rangle} \to \langle a_{k-1}x\rangle \mid 1 \le k \le n+1, x \in S\}$		+	$\{4, 6\}$
6	$\{\langle x\rangle \to \ll x \gg \mid x \in S\}$		−	$\{5, 7\}$
7	$\{a_{k-1}'' \to \overline{a_k'} \mid 1 \le k \le n+1\}$		0	$\{6, 8\}$
8	$\{\overline{a_k'} \to a_k' \mid 1 \le k \le n+1\}$		+	$\{7, 9\}$
9	$\{\ll x \gg \to \overline{\langle x\rangle} \mid x \in S\}$		−	$\{4, 8, 10\}$
10	$\{\overline{\langle a_0x\rangle} \to \overline{\ll a_0x \gg} \mid x \in S\}$		0	$\{9, 11\}$
11	$\{a_k' \to a_k \mid 1 \le k \le n+1\}$		+	$\{10, 12, 15\}$
12	$\{\overline{\ll a_0a_kx \gg} \to [a_kx] \mid 1 \le k \le n+1, x \in S\} \cup$ $\{\overline{\ll a_0 \gg} \to [a_0]\}$		0	$\{3, 11, 13\}$
13	$\{[a_0] \to \varepsilon\}$	l	−	$\{12, 14\}$
14	$\{a^\circ \to \varepsilon\}$	l	−	$\{1, 13\}$
15	$\emptyset$		−	$\{11\}$

The construction of Γ is complete as soon as we define the valuation mapping φ. It is defined as follows:

$$\varphi(a_k) = 0, 1 \le k \le n, \qquad \varphi(H) = -10,$$
$$\varphi([x]) = \varphi(\overline{\langle x\rangle}) = \varphi(\overline{\langle a_0x\rangle}) = 1, x \in S,$$
$$\varphi(a^\circ) = -2, a \in V',$$
$$\varphi(a_k') = 1, 0 \le k \le n+1,$$
$$\varphi(a_k'') = 2, 0 \le k \le n+1,$$
$$\varphi(\langle x\rangle) = \varphi(\langle a_0x\rangle) = -1, x \in S,$$
$$\varphi(\ll x \gg) = \varphi(\ll a_0x \gg) = 0, x \in S,$$
$$\varphi(\overline{a_k'}) = 3, 1 \le k \le n+1,$$
$$\varphi(\overline{\ll x \gg}) = \varphi(\overline{\ll a_0x \gg}) = 2, x \in S.$$

We show that Γ accepts a word w that does not contain H if an only if T eventually halts on w. Let $w = aby, a, b \in V, y \in V^*$ be a word that does not contain H such that T eventually halts on w. We show how w can be accepted by Γ. At the beginning of the computation w is found in node 1, where the first symbol a can be replaced with $[\mu(a)]$. From our further explanations, it will turn out that if the symbol a replaced by $[\mu(a)]$ in node 1 is not the leftmost one, then the computation on this word is blocked. The valuation of the new word, $[\mu(a)]by$, has a positive value such that the word enters node 2 which is positively charged. In node 2, we can rewrite b as b°, getting the new word $[\mu(a)]b^\circ y$. Again, later we infer that if another symbol is replaced, the computation will be blocked. The word $[\mu(a)]b^\circ y$ can only enter node 3 where the symbol a_0' is inserted to its right end obtaining $[\mu(a)]b^\circ y a_0'$. This word can enter node 4 and 12. Note that the copy entering node 12 remains there forever. Therefore, we continue our analysis in node 4. In this node, a_0' is replaced by a_0'' which change the polarization of the word from neutral to positive which makes it to migrate to node 5.

We now assume that $\mu(a) = a_k x$ for some $1 \leq k \leq n+1$. In node 5, $[a_k x]$ is replaced by $\langle a_{k-1}x\rangle$ and the new word which is negatively charged enters node 6. After its first symbol $\langle a_{k-1}x\rangle$ is substituted with $\ll a_{k-1}x \gg$, the word has a null valuation and enters node 7. The word is successively transformed in $\ll a_{k-1}x \gg b^\circ y\overline{a_1'}$ (in node 7), $\ll a_{k-1}x \gg b^\circ y a_1'$ (in node 8), and $\overline{\langle a_{k-1}x\rangle} b^\circ y a_1'$ (in node 9). If $k > 1$, this string first returns to node 4, resulting in $\overline{\langle a_{k-1}x\rangle} b^\circ y a_1''$, and then enters node 5, resulting in $\langle a_{k-2}x\rangle b^\circ y a_1''$. This process continues by iteratively passing the sequence of nodes $4, 5, 6, 7, 8, 9$ until a string of the form $\overline{\langle a_0 x\rangle} b^\circ y a_k'$ is obtained in node 9. Now the current word enters 10 and 11, where it is rewritten into $\overline{\ll a_0 x \gg} b^\circ y a_k'$ and $\overline{\ll a_0 x \gg} b^\circ y a_k$, respectively. If $k = n+1$, then the word enters the output node 15 and the computation halts. Otherwise, the current string can only enter node 12, where its first symbol $\overline{\ll a_0 x \gg}$ is replaced either by $[a_j x']$, provided that $x = a_j x'$, or by $[a_0]$, if $x = \varepsilon$. In the former case, the whole process described above resumes from the node 3. In the latter, we actually reached a word of the form $[a_0]b^\circ y\mu(a)$. This word can enter node 13, where the symbol $[a_0]$ is deleted, provided that it is the leftmost symbol. A copy of this word remains in 13 forever, but another copy enters 14, where the symbol b° is deleted, provided that it is the leftmost symbol. One can see now that if the symbols substituted in the nodes 1 and 2 were not the right ones, the computation will get stuck. Note that the word obtained in node 14 is exactly $y\mu(a)$, which means that we have correctly simulated the step $a \to \phi(a)$ in the tag system. Now, this word enters 1 where the simulation of the next step in T starts.

By these explanations, we infer that $w \in L(\Gamma)$ if and only if T will eventually halt on w. $\square$

4 Arbitrary Turing Machines Can Be Simulated by NPEP

Although 2-tag systems efficiently simulate deterministic Turing machines, via cyclic tag systems (see, e.g., [23]), the previous result does not allow us to say much about the NPEP accepting in a computationally efficient way all recursively enumerable languages. We now discuss how recursively enumerable languages can be efficiently (from the time complexity point of view) accepted by NPEP by simulating arbitrary Turing machines.

Theorem 2. *For any recursively enumerable language L, accepted in $\mathcal{O}(f(n))$ by a Turing machine with tape alphabet U, there exists an NPEP of size $10card(U)$ accepting L in $\mathcal{O}(f(n))$ steps.*

Proof. Let $M = (Q, V, U, \delta, q_0, B, F)$ be a Turing machine with $U \cap Q = \emptyset$, and $U = \{a_1, a_2, \ldots, a_{n+1}\}$, $a_{n+1} = B$. We start the construction of the NPEP Γ accepting the language accepted by M with the definition of its working alphabet:

$$\begin{aligned}W = {} & U \cup Q \cup \{\overline{q} \mid q \in Q\} \cup \{\overline{a} \mid a \in U\} \cup \{a', a'', \widetilde{a}, \widehat{a} \mid a \in U \setminus \{B\}\} \cup \\ & \{[q, a, D] \mid q \in Q, a \in U \setminus \{B\}, D \in \{L, R\}\}.\end{aligned}$$

We now can define the valuation mapping φ as follows:

$$\begin{array}{ll}
\varphi(a) = 0, a \in U, & \varphi(\overline{a}) = 2, a \in U, \\
\varphi(a') = -1, a \in U \setminus \{B\}, & \varphi(a'') = 2, a \in U \setminus \{B\}, \\
\varphi(q) = -1, q \in Q \setminus F & \varphi(q) = 1, q \in F, \\
\varphi(\widetilde{a_i}) = -p_i, 1 \le i \le n, & \varphi(\widehat{a_i}) = p_i, 1 \le i \le n, \\
\varphi(\langle s, a_i, L\rangle) = -p_i, s \in Q, 1 \le i \le n, & \varphi(\langle s, a_i, R\rangle) = p_i, s \in Q, 1 \le i \le n, \\
\varphi([s, a, D]) = -2, & \varphi(\overline{q}) = 0, q \in Q, \\
\quad s \in Q, a \in U \setminus \{B\}, D \in \{L, R\} &
\end{array}$$

Here p_i denotes the i^{th} odd prime number. The processors placed in the nodes of the network are defined in Table 2.

We now analyze the computation of Γ on an input word, say w, which is placed in the input node $\underline{In}$. Here the B symbol is added to its right-hand end, yielding wB which has a neutral polarization. Therefore, a copy of this word remains in $\underline{In}$, while another copy migrates to $InsSt$ (Insert State). It will turn out, by our further explanations, that if w is accepted by a computation of M that uses the minimal number t of auxiliary cells (cells that initially contain B), then every word wB^j, with $j < t$, that enters $InsSt$ will be eventually blocked in a node. We assume that wB^j, for some $j \ge t$, enters $InsSt$. It is transformed into wB^jq_0 which is negatively charged, such that each node $IdS(a_i)$ (Identify Symbol), $1 \le i \le n+1$, receives one of its copies. Inductively, we may assume that the current word is yB^kqx, for some $k \ge 0$, which signifies that the Turing Machine M is in state q and has on its tape the word xy, with its head positioned on the first symbol of y. Assume that $y = a_iz$, for some $1 \le i \le n+1$; note that the copy of yB^kqx that enters $IdS(a_k)$, $k \ne i$, will be further blocked in Del.

Let us follow the copy of a_izB^kqx that enters $IdS(a_i)$. After an occurrence of a_i, not necessarily the leftmost one, is replaced by $\overline{a_i}$, the word arrives in $ChT(a_i)$ (Choose Transition). Here a symbol $[s, a_m, D]$, such that $(s, a_m, D) \in \delta(q, a_i)$, is adjoined which makes the new word to have a null value through the valuation mapping φ. This word enters Del, where a barred symbol is removed provided that it is the leftmost one. We can see now that, if the occurrence of a_i substituted in the node $IdS(a_i)$ was not the leftmost one, the word remains blocked in Del. Therefore, our current word becomes $zB^k[s, a_m, D]x$, $1 \leq m \leq n$, $D \in \{L, R\}$. As it is negatively charged, it enters $DetLR$ (Determine Left-Right). In this node, $[s, a_m, D]$ is replaced by $\langle s, a_m, D\rangle$ and the new word is $zB^k\langle s, a_m, D\rangle x$.

Table 2. The definition of the nodes of Γ

Node	M	α	π	Adjacency list
$\underline{In}$	$\{\varepsilon \to B\}$	r	0	$\{InsSt\}$
$InsSt$	$\{\varepsilon \to q_0\}$	r	0	$\{\underline{In}\}\cup$ $\{IdS(a) \mid a \in U\}$
$IdS(a)$, $a \in U$	$\{a \to \overline{a}\}$		$-$	$\{InsSt, RestS, ChT(a)\}$
$ChT(a)$, $a \in U$	$\{q \to [s, b, D] \mid$ $(s, b, D) \in \delta(q, a)\}$		$+$	$\{Del, IdS(a)\}$
Del	$\{\overline{a} \to \varepsilon \mid a \in U\}$	l	0	$\{DetLR\}\cup$ $\{ChT(a) \mid a \in U\}$
$DetLR$	$\{[s, b, D] \to \langle s, b, D\rangle \mid$ $(s, b, D) \in \delta(q, a)$, for some $q \in Q \setminus F, a \in U\}$		$-$	$\{Del\}\cup$ $\{InsL_1(a) \mid a \in U \setminus \{B\}\cup$ $\{InsR_1(a) \mid a \in U \setminus \{B\}\cup$
$InsR_1(a)$, $a \in U \setminus \{B\}$	$\{\varepsilon \to \tilde{a}\}$	r	$+$	$\{DetLR, CSR(a)\}$
$CSR(a)$ $a \in U \setminus \{B\}$	$\{\langle s, a, R\rangle \to \overline{s} \mid s \in Q\}$		0	$\{InsR_1(a), InsR_2(a)\}$
$InsR_2(a)$ $a \in U \setminus \{B\}$	$\{\tilde{a} \to a\}$		$-$	$\{CSR(a), RestS\}$
$InsL_1(a)$, $a \in U \setminus \{B\}$	$\{\varepsilon \to \hat{a}\}$	l	$-$	$\{DetLR, CSL(a)\}$
$CSL(a)$ $a \in U \setminus \{B\}$	$\{\langle s, a, L\rangle \to \overline{s} \mid s \in Q\}$		0	$\{InsL_1(a), InsL_2(a)\}$
$InsL_2(a)$ $a \in U \setminus \{B\}$	$\{\hat{a} \to a\}$		$+$	$\{CSL(a), Move\}$
$Move$	$\{a \to a' \mid a \in U \setminus \{B\}\}$		0	$\{InsL_2(a) \mid a \in U \setminus \{B\}\}\cup$ $Ins(a) \mid a \in U \setminus \{B\}$
$Ins(a)$ $a \in U \setminus \{B\}$	$\{\varepsilon \to a''\}$	l	$-$	$\{Move, Del(a)\}$
$Del(a)$ $a \in U \setminus \{B\}$	$\{a' \to \varepsilon\}\cup$	r	$+$	$\{Ins(a), FL\}$
FL	$\{a'' \to a \mid a \in U \setminus \{B\}\}$		$+$	$\{RestS\}\cup$ $\{Del(a) \mid a \in U \setminus \{B\}\}$
$RestS$	$\{\overline{s} \to s \mid s \in Q\}$		0	$\{FL, Out\}\cup$ $\{InsR_2(a) \mid a \in U \setminus \{B\}\}\cup$ $\{IdS(a) \mid a \in U\}\cup$
$\underline{Out}$	$\emptyset$		$+$	$\{RestS\}$

The value of this word is either negative, if $D = L$, or positive, if $D = R$. We first consider the case $D = R$; each node $InsR_1(a)$, $a \in U \setminus \{B\}$, receives a copy of the word $zB^k\langle s, a_m, R\rangle x$. If a copy arrives in $InsR_1(a)$, $a \neq a_m$, then it can never be transformed into a word with a null value. After a number of evolutionary steps, each of them inserting a $\widetilde{a}$ at the end, the word gets a negatively value and migrates back to $DetLR$, where it remains forever. The copy of $zB^k\langle s, a_m, R\rangle x$ that enters $InsR_1(a_m)$, is modified into $zB^k\langle s, a_m, R\rangle x\widetilde{a_m}$. As it has a null value, it enters $CSR(a_m)$ (Change State from the Right), where it becomes $zB^k\overline{s}x\widetilde{a_m}$. The word $zB^k\overline{s}x\widetilde{a_m}$ is negatively charged and migrates to $InsR_2(a_m)$, where wta_m is substituted with a_m. Now, the word enters $RestS$ (Restore State), where the word becomes zB^ksxa_m, hence we have correctly simulated a move of M to the right.

If $D = L$, then by means of the nodes $InsL_1(a_m)$, $CSL(a_m)$, and $InsL_2(a_m)$, the symbol a_m is inserted in the beginning of the word. Then, by means of the nodes $Move$, $Ins(b)$, $Del(b)$, and FL, the symbol $b \neq B$ is rotated from the end of the word to its beginning. After this rotation, the word enters $RestS$, where it becomes $ba_mzB^ksx_1$, provided that $x = x_1b$. In conclusion, we have correctly simulated a move of M to the left.

In order to simulate another move, the word enters $IdS(a)$, $a \in U$, and the whole process discussed above resumes. Note that if the state s in a word existing in $RestS$ is a final state, then the word enters $\underline{Out}$ and the computation halts.

Now, the simulation proof is complete. From Table 2, it is easy to see that the size of Γ is exactly $10card(U)$. □

An analysis of the proof, reveals that each move of M is simulated by Γ in a constant number of steps. On the other hand, if M accepts in $\mathcal{O}(f(n))$ time, then the number of necessary steps in the beginning of any computation of Γ on a word of length n is at most $\mathcal{O}(f(n))$. Consequently, $Time_\Gamma \in \mathcal{O}(f(n))$.

We finish this work with two *open problems* that naturally arise:

1. Is the size proved in Theorem 1 optimal?
2. Can arbitrary Turing machines be simulated by NPEP of constant size? In the affirmative, is such a simulation still time-efficient?

References

1. Alarcón, P., Arroyo, F., Mitrana, V.: Networks of polarized evolutionary processors as problem solvers. In: Advances in Knowledge-Based and Intelligent Information and Engineering Systems. Frontiers in Artificial Intelligence and Applications, pp. 807–815. IOS Press (2012)
2. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Inf. 39, 517–529 (2003)
3. Diaz, M.A., de Mingo, L.F., Gómez, N.: Implementation of massive parallel networks of evolutionary processors (MPNEP): 3-colorability problem. In: Krasnogor, N., Nicosia, G., Pavone, M., Pelta, D. (eds.) NICSO 2007. SCI, vol. 129, pp. 399–408. Springer, Heidelberg (2007)
4. Diaz, M.A., de Mingo, L.F., Gómez, N.: Networks of evolutionary processors: Java Implementation of a threaded processor. International Journal of Information Theories & Applications 15, 37–43 (2008)

5. Drăgoi, C., Manea, F., Mitrana, V.: Accepting networks of evolutionary processors with filtered connections. J. UCS 13, 1598–1614 (2007)
6. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Trans. Amer. Math. Soc. 117, 533–546 (1965)
7. Hillis, W.D.: The Connection Machine. MIT Press, Cambridge (1979)
8. Loos, R., Manea, F., Mitrana, V.: Small universal accepting hybrid networks of evolutionary processors. Acta Inf. 47, 133–146 (2010)
9. Manea, F., Martín-Vide, C., Mitrana, V.: On the size complexity of universal accepting hybrid networks of evolutionary processors. Mathematical Structures in Computer Science 17, 753–771 (2007)
10. Manea, F., Margenstern, M., Mitrana, V., Pérez-Jiménez, M.J.: A new characterization of NP, P, and PSPACE with accepting hybrid networks of evolutionary processors. Theory Comput. Syst. 46, 174–192 (2010)
11. Manea, F., Martín-Vide, C., Mitrana, V.: Accepting networks of evolutionary word and picture processors: A survey. In: Scientific Applications of Language Methods, Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory, vol. 2, pp. 523–560. World Scientific (2010)
12. Margenstern, M., Mitrana, V., Pérez-Jímenez, M.J.: Accepting Hybrid Networks of Evolutionary Processors. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005)
13. Martín-Vide, C., Pazos, J., Păun, G., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 290–299. Springer, Heidelberg (2002)
14. Martín-Vide, C., Mitrana, V., Pérez-Jiménez, M.J., Sancho-Caparrini, F.: Hybrid networks of evolutionary processors. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 401–412. Springer, Heidelberg (2003)
15. Minsky, M.L.: Size and structure of universal Turing machines using tag systems. In: Recursive Function Theory, Symp. in Pure Mathematics, vol. 5, pp. 229–238 (1962)
16. Navarrete, C.B., Echeanda, M., Anguiano, E., Ortega, A., Rojas, J.M.: Parallel simulation of NEPs on clusters. In: Proc. IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - WI-IAT, pp. 171–174. IEEE Computer Society (2011)
17. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
18. Post, E.L.: Formal reductions of the general combinatorial decision problem. Amer. J. Math. 65, 197–215 (1943)
19. Păun, G.: Membrane computing. An introduction. Springer, Berlin (2002)
20. Rogozhin, Y.: Small universal Turing machines. Theoret. Comput. Sci. 168, 215–240 (1996)
21. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I-III. Springer, Berlin (1997)
22. Sankoff, D., et al.: Gene order comparisons for phylogenetic inference:evolution of the mitochondrial genome. Proceedings of the National Academy of Sciences of the United States of America 89, 6575–6579 (1992)
23. Woods, D., Neary, T.: On the time complexity of 2-tag systems and small universal Turing machines. In: 47th Annual IEEE Symposium on Foundations of Computer Science FOCS 2006, pp. 439–448 (2006)