Computational Completeness of Networks of Evolutionary Processors with Elementary Polarizations and a Small Number of Processors

Rudolf Freund, Vladimir Rogojin, Sergey Verlan

Computational Completeness of Networks of Evolutionary Processors with Elementary Polarizations and a Small Number of Processors

Rudolf Freund[1], Vladimir Rogojin[2], and Sergey Verlan[3]

[1] Faculty of Informatics,
TU Wien
Favoritenstraße 9–11, 1040 Vienna, Austria
rudi@emcc.at
[2] Department of Information Technologies,
Åbo Akademi University,
Joukahainengatan 3-5, 20520 Turku
vladimir.rogojin@abo.fi
[3] Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est Créteil,
61 av. du général de Gaulle, 94010 Créteil, France
verlan@u-pec.fr

Abstract. We improve previous results obtained for networks of evolutionary processors with elementary polarizations $-1, 0, 1$ by showing that only the very small number of seven processors is needed to obtain computational completeness. In the case of not requiring a special output node even only five processors are shown to be sufficient.

1 Introduction

Networks of evolutionary processors (NEPs) consist of cells (processors) each of them allowing for specific operations on strings. Computations in such a network consist of alternatingly performing two steps – an *evolution step* where in each cell all possible operations on all strings currently present in the cell are performed, and a *communication step* in which strings are sent from one cell to another cell provided specific conditions are fulfilled. Examples of such conditions are (output and input) filters which have to be passed, and these (output and input) filters can be specific types of regular languages or permitting and forbidden context conditions. The set of strings obtained as results of computations by the NEP is defined as the set of objects which appear in some distinguished node in the course of a computation. In networks of evolutionary processors with polarizations each symbol has assigned a fixed integer value; the polarization of a string is computed according to a given evaluation function, and in the communication step copies of strings are moved to all cells having the same polarization. As in [12], in this paper we only consider the elementary polarizations $-1, 0, 1$ for the symbols as well as for the cells.

Seen from a biological point of view, networks of evolutionary processors are a collection of cells communicating via membrane channels which makes them to be seen as tissue-like P systems (see [11]) considered in the area of membrane computing (see [14]); as in membrane computing, the computations are carried out in a parallel and synchronized way in all cells The operations considered for the processors (cells) in networks of evolutionary processors usually are the point mutations insertion, deletion, and substitution, well-known from biology as operations on DNA.

Networks of Evolutionary Processors (NEPs) were introduced in [7] and [8] as a model of string processing devices distributed over a graph, with the processors carrying out the operations insertion, deletion, and substitution. NEPs with a very small number of nodes are very powerful computational devices: already with two nodes, they are as powerful as Turing machines, e.g., see [3, 4]. For a survey of the main results regarding NEPs the interested reader is referred to [10].

In *hybrid networks of evolutionary processors* (HNEPs), each language processor performs only one of these operations on a certain position of the strings. Furthermore, the filters are defined by some variants of random-context conditions, i.e., they check the presence and the absence of certain symbols in the strings. For an overview on HNEPs and the so far known best results, we refer the reader to [1].

Networks of polarized evolutionary processors were considered in [6] (a new version of that paper is going to appear, [5]), and networks of evolutionary processors with elementary polarizations $-1, 0, 1$ were investigated in [12]. In this paper we consider the same model of networks of evolutionary processors with elementary polarizations $-1, 0, 1$ as in [12], yet we considerably improve the number of processors (cells) needed to obtain computational completeness from 35 to 7, which makes these results already comparable with those obtained in [1] for hybrid networks of evolutionary processors using permitting and forbidden contexts as filters for the communication of strings between cells.

The rest of the paper is structured as follows: In Section 2 we give the definitions of the model of a network of evolutionary processors with elementary polarizations $-1, 0, 1$ (*NePEP* for short) and of the variant of a circular Post machine we are going to simulate by the NePEP. In Section 3 we show our main result proving that any circular Post machine can be simulated by an NePEP with only seven processors (cells), and in the case of not requiring a special output processor even only five processors are needed. A summary of the results and an outlook to future research conclude the paper.

2 Prerequisites

We start by recalling some basic notions of formal language theory. An alphabet is a non-empty finite set. A finite sequence of symbols from an alphabet V is called a *string* over V. The set of all strings over V is denoted by V^*; the *empty string* is denoted by λ; moreover, we define $V^+ = V^* \setminus \{\lambda\}$. The *length* of a

string x is denoted by $|x|$, and by $|x|_a$ we denote the number of occurrences of a letter a in a string x. For a string x, *alph(x)* denotes the smallest alphabet Σ such that $x \in \Sigma^*$. For more details of formal language theory the reader is referred to the monographs and handbooks in this area, such as [15].

We only remark that in this paper, string rewriting systems as Turing machines, Post systems, etc. are called *computationally complete* if these systems are able to compute any partial recursive relation R on strings over any alphabet U. Computational completeness in the usual sense with respect to acceptance and generation directly follows from this general kind of computational completeness; for more details we refer to [1]. The definitions of the succeeding subsections are mainly taken from [1] and [12].

2.1 Insertion, Deletion, and Substitution

For an alphabet V, let $a \to b$ be a rewriting rule with $a, b \in V \cup \{\lambda\}$, and $ab \neq \lambda$; we call such a rule a *substitution rule* if both a and b are different from λ; such a rule is called a *deletion rule* if $a \neq \lambda$ and $b = \lambda$, and it is called an *insertion rule* if $a = \lambda$ and $b \neq \lambda$. The set of all substitution rules, deletion rules, and insertion rules over an alphabet V is denoted by Sub_V, Del_V, and Ins_V, respectively.

Given such rules $\pi \equiv a \to b \in Sub_V$, $\rho \equiv a \to \lambda \in Del_V$, and $\sigma \equiv \lambda \to a \in Ins_V$ as well as a string $w \in V^*$, we define the following *actions* of π, ρ, and σ on w:

- If $\pi \equiv a \to b \in Sub_V$, then
  $\pi(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, & \text{if } \mid w \mid_a > 0, \\ \{w\}, & \text{otherwise.} \end{cases}$
- If $\rho \equiv a \to \lambda \in Del_V$, then
  $\rho^r(w) = \begin{cases} \{u : \exists u \in V^* \ (w = ua)\}, & \text{if } \mid w \mid_a > 0, \\ \{w\}, & \text{otherwise.} \end{cases}$
  $\rho^l(w) = \begin{cases} \{v : \exists v \in V^* \ (w = av)\}, & \text{if } \mid w \mid_a > 0, \\ \{w\}, & \text{otherwise.} \end{cases}$
- If $\sigma \equiv \lambda \to a \in Ins_V$, then $\sigma^r(w) = \{wa\}$ and $\sigma^l(w) = \{aw\}$.

The symbol $\alpha \in \{*, l, r\}$ denotes the mode of applying a substitution, insertion or deletion rule to a string, namely, at any position ($\alpha = *$), on the left-hand end ($\alpha = l$), or on the right-hand end ($\alpha = r$) of the string, respectively.

For any rule β, $\beta \in \{\pi, \rho, \sigma\}$, any mode $\alpha \in \{*, l, r\}$, and any $L \subseteq V^*$, we define the *α-action* of β on L by $\beta^\alpha(L) = \bigcup_{w \in L} \beta^\alpha(w)$. For a given finite set of rules M, we define the α-action of M on a string w and on a language L by $M^\alpha(w) = \bigcup_{\beta \in M} \beta^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively. In the following, substitutions will only be used at arbitrary positions, i.e., with $\alpha = *$, which will be omitted in the description of the rule.

2.2 Post Systems and Circular Post Machines

The left and right insertion, deletion, and substitution rules defined in the preceding subsection are special cases of string rewriting rules only working at the

ends of a string; they can be seen as restricted variants of Post rewriting rules as already introduced by Emil Post in [13]: for a *simple Post rewriting rule* $\Pi_s \equiv u\$x \to y\$v$, where $u, v, x, y \in V^*$, for an alphabet V, we define

$$\pi_s(w) = \{yzv \mid w = uzx,\ z \in V^*\}.$$

A *normal Post rewriting rule* $\pi_n \equiv \$x \to y\$$ is a special case of a simple Post rewriting rule $u\$x \to y\$v$ with $u = v = \lambda$ (we also assume $xy \neq \lambda$); this normal Post rewriting rule $\$x \to y\$$ is the mirror version of the normal form rules $u\$ \to \$v$ as originally considered in [13] for Post canonical systems; yet this variant has already been used several times for proving specific results in the area of membrane computing, e.g., see [9]. A *Post system of type* X is a construct (V, T, A, P) where V is a (finite) set of *symbols*, $T \subseteq V$ is a set of *terminal symbols*, $A \in V^*$ is the *axiom*, and P is a finite set of *Post rewriting rules* of type X; for example, X can mean simple or normal Post rewriting rules. In both cases it is folklore that these Post systems of type X are computationally complete.

The basic idea of the computational completeness proofs for Post systems is the "rotate-and-simulate"-technique, i.e., the string is rotated until the string x to be rewritten appears on the right-hand side, where it can be erased and replaced by the string y on the left-hand side, which in total can be accomplished by the rule $\$x \to y\$$. By rules of the form $\$a \to a\$$ for each symbol a the string can be rotated. In order to indicate the beginning of the string in all its rotated versions, a special symbol B (different from all others) is used; B is to be erased at the end of a successful computation.

Circular Post machines are machine-like variants of Post systems using specific variants of simple Post rewriting rules; the variant of $CPM5$ we use in this paper was investigated in [2].

Definition 1. *A (non-deterministic)* CPM5 *is a construct*

$$M = (\Sigma, T, Q, q_1, q_0, R),$$

where Σ is a finite alphabet, $T \subseteq \Sigma$ is the set of terminal symbols, Q is the set of states, $q_1 \in Q$ is the initial state, $q_0 \in Q$ is the only terminal state, and R is a set of simple Post rewriting rules of the following types (we use the notation $Q' = Q \setminus \{q_0\}$):

- *$px\$ \to q\$$ (*deletion rule*) with $p \in Q'$, $q \in Q$, $x \in \Sigma$; we also write $px \to q$ and, for any $w \in \Sigma^*$, the corresponding computation step is $pxw \overset{px \to q}{\longrightarrow} qw$;*
- *$p\$ \to q\$y$ (*insertion rule*) with $p \in Q'$, $q \in Q$, $y \in \Sigma$; we also write $p \to yq$ and, for any $w \in \Sigma^*$, the corresponding computation step is $pw \overset{p \to yq}{\longrightarrow} qwy$.*

The CPM5 is called deterministic *if for any two deletion rules $px \to q_1$ and $px \to q_2$ we have $q_1 = q_2$ and for any two insertion rules $p \to q_1y_1$ and $p \to q_2y_2$ we have $q_1y_1 = q_2y_2$.*

The name circular Post machine comes up from the idea of interpreting the machines to work on circular strings where both deletion and insertion rules have local effects, as for circular strings the effect of the insertion rule $p\$ \to q\$y$ is the same as the effect of $p \to yq$ directly applied to a circular string, which also justifies writing $p\$ \to q\$y$ as $p \to yq$.

For a given input string w, $w \in T^*$, the CPM5 M starts with q_1w and applies rules from R until it eventually reaches a configuration q_0v for some $v \in T^*$; in this case we say that (w, v) is in the relation computed by M .

Definition 2. *A CPM5 $M = (\Sigma, T, Q, q_1, q_0, R)$ is said to be in* normal form *if*

- $Q \setminus \{q_0\} = Q_1 \cup Q_2$ *where* $Q_1 \cap Q_2 = \emptyset$*;*
- *for every $p \in Q_1$ and every $x \in \Sigma$, there is exactly one instruction of the form $px \to q$, i.e., Q_1 is the set of states for deletion rules;*
- *for every insertion rule $p \to yq$ we have $p \in Q_2$, i.e., Q_2 is the set of states for insertion rules, and moreover, if $p \to y_1q_1$ and $p \to y_2q_2$ are two different rules in R, then $y_1 = y_2$.*

Theorem 1. *(see [2]) CPM5s in normal form are computationally complete.*

2.3 Networks of Evolutionary Processors with Elementary Polarizations

Definition 3. *A* polarized evolutionary processor *over V is a triple (M, α, π) where*

- *M is a set of substitution, deletion or insertion rules over the alphabet V, i.e., $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$;*
- *α gives the action mode of the rules of the node;*
- *$\pi \in \{-1, 0, +1\}$ is the polarization of the node (negative, neutral, positive).*

The set M represents the set of evolutionary rules of the processor. It is important to note that a processor is "specialized" in one type of evolutionary operation only as in HNEPs. The set of evolutionary processors over V is denoted by EP_V.

Definition 4. *A* network of polarized evolutionary processors *(NPEP for short) is a 7-tuple $\Gamma = (V, T, H, \mathcal{R}, \varphi, n_{in}, n_{out})$ where*

- *V is the alphabet of the network;*
- *T is the input/output alphabet, $T \subseteq V$; $T \subseteq V$;*
- *$H = (X_H, E_H)$ is an undirected graph (without loops) with the set of vertices (nodes) X_H and the set of (undirected) edges E_H; H is called the underlying* communication graph *of the network;*
- *$\mathcal{R} : X_H \longrightarrow EP_V$ is a mapping which with each node $x \in X_H$ associates the polarized evolutionary processor $\mathcal{R}(x) = (M_x, \alpha_x, \pi_x)$;*
- *φ is an evaluation function from V^* into the set of integers;*
- *$n_{in}, n_{out} \in X_H$ are the* input *and the* output *node, respectively.*

The number of nodes in X_H, $card(X_H)$, is called the *size* of Γ. If the evaluation mapping φ takes values in the set $\{-1, 0, 1\}$ only, the network is said to be with *elementary polarization* of symbols (an NePEP for short).

A *configuration* of an NPEP Γ, as defined above, is a mapping $C : X_H \longrightarrow 2^{V^*}$ which associates a set of strings over V with each node x of the graph. A component $C(x)$ of a configuration C is the set of strings that can be found in the node x of this configuration, hence, a configuration can be considered as a list of the sets of strings which are present in the nodes of the network at a given moment.

A computation of Γ consists of alternatingly applying an *evolutionary step* and 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 thus yielding the new configuration C', and we write $C \Longrightarrow C'$ if and only if

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

In a *communication step*, each node processor $x \in X_H$ sends out copies of all its strings, but keeping a local copy of the strings having the same polarization 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 polarization as that of x, thus yielding the new configuration C' from configuration C, and we write $C \vdash C'$,

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

for all $x \in X_H$. Here $sign(m)$ is the sign function which returns $+1, 0, -1$, provided that m is a positive integer, is 0, or is a negative integer, respectively. Note that all strings with a different polarization than that of x are expelled. Further, each expelled word from a node x that cannot enter any node connected to x is lost.

In the following, we will only use the evaluation function φ with $\varphi(\lambda) = 0$ and $\varphi(aw) = \varphi(a) + \varphi(w)$ for all $a \in V$ and $w \in V^*$, i.e., the value a string is the sum of the values of the symbols contained in it; we write φ_s for this function.

Given an input word $w \in T^*$, the initial configuration C_0 of Γ for w is defined by $C_0^{(w)}(n_{in}) = \{w\}$ and $C_0^{(w)}(n) = \emptyset$ for all other nodes $x \in X_H \setminus \{n_{in}\}$. 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 $i \geq 0$.

As results we take all terminal strings appearing in the output cell n_{out} during a computation of Γ. In fact, in [1] this variant was called *with terminal extraction*. On the other hand, we may require a special output where *only* the terminal strings appear, which we will consider as the *standard* variant.

3 Main result

In this section we now show our main result how a given CPM5 can be simulated by an NePEP (with terminal extraction) with only 7 (5) cells provided that for a given input string $w \in T^*$ we start with the initial string $q_1 w$ in the input cell, where q_1 is the initial state of the CPM5. In order to start with the input string w directly we would have to add two more nodes to carry out this initial procedure of adding the initial state q_1.

Theorem 2. *For any CPM5 $M = (\Sigma, T, Q, q_1, q_0, R)$ in normal form there exists a standard NePEP with only seven cells $\Gamma = (V, T, H, \mathcal{R}, \phi_s, i_1, i_0)$ being able to simulate the computations of M.*

Proof. Let $n = |T|$, $m = |Q|$, $0 \leq i \leq m$ and $0 \leq k \leq n$. We define

$$\begin{aligned} V = T &\cup \{q_i^0, \hat{q}_i^+, \hat{q}_i^-, X_i^-, D_i^0, D_i^+, \hat{D}_i^+ \mid 0 \leq i \leq m\} \\ &\cup \{q_{k,i}^-, \hat{q}_{k,i}^0 \mid 0 \leq k \leq n, 0 \leq i \leq m\} \\ &\cup \{A_{i,k}^-, A_{i,k}^0 \mid 0 \leq k \leq n, 0 \leq i \leq m\} \\ &\cup \{A_k^0, A_k^+, \hat{A}_k^+, \check{A}_k^+ \mid 0 \leq k \leq n\} \cup \{\varepsilon^-\} \end{aligned}$$

The evaluation ϕ_s for the symbols in V corresponds to the superscript of the symbol, i.e., for $\alpha^z \in V$ with $z \in \{+, 0, -\}$ we define $\phi_s(\alpha^0) = 0$, $\phi_s(\alpha^+) = +1$, $\phi_s(\alpha^-) = -1$, and, moreover, for $a \in T$, we take $\phi_s(a) = 0$.

The communication graph H consists of the set of nodes $\{1, 2, 3, 4, 5, 6, 7\}$ and of the following set of undirected edges:
$\{\{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{4,5\}, \{4,6\}, \{6,7\}\}$.

Node 1 is the input and node 7 is the output node.

For the seven nodes i, $1 \leq i \leq 7$, the corresponding evolutionary processors $\mathcal{N}(i)$ are defined as follows:

$\alpha(1) = \alpha(3) = \alpha(4) = \alpha(7) = *$, $\alpha(2) = r$ and $\alpha(5) = \alpha(6) = l$.

$\pi(1) = \pi(7) = 0$, $\pi(2) = \pi(4) = \pi(5) = -$ and $\pi(3) = \pi(6) = +$.

For the types of rules in the rule sets M_i we have $M_1, M_3, M_4 \subset SUB_V$, $M_2 \subset INS_V$, $M_5, M_6 \subset DEL_V$, and $M_7 = \emptyset$, i.e., M_7 could be assumed to be any type of rules.

Processor 1 has polarization (charge) 0 and uses substitution rules to contribute to the simulation of insertion and deletion rules of M:

Insertion: $q_s \to q_j a_k$ $(1 \leq l \leq k)$	**Deletion:** $q_s a_k \to q_j$ $(1 \leq i \leq s)$
1.1: $q_s^0 \to q_{k,j}^-$	1.7: $q_s^0 \to X_s^-$
1.2: $q_{l,j}^- \to q_{l-1,j}^0$	1.8: $A_{i,l}^- \to A_{i+1,l}^0$
1.3: $A_l^0 \to \hat{A}_l^+$	1.9: $A_{i,l}^0 \to \hat{A}_{i,l}^0$
1.4: $A_l^0 \to \check{A}_l^+$	1.10: $D_i^0 \to \hat{D}_i^+$
1.5: $q_{l,j}^0 \to \hat{q}_{l,j}^0$	1.11: $\hat{D}_i^+ \to D_i^+$
1.6: $\hat{A}_l^+ \to A_l^+$	1.12: $A_{i,l}^- \to \hat{q}_j^+$
	1.13: $\hat{q}_j^+ \to \hat{q}_j^-$

Processor 3 has polarization (charge) +1 and uses substitution rules to contribute to the simulation of insertion and deletion rules of M:

Insertion: $q_s \to q_j a_k$ $(1 \le l \le k)$	**Deletion:** $q_s a_k \to q_j$ $(1 \le i \le s)$
3.1: $A_l^+ \to A_{l+1}^0$	3.5 : $D_i^+ \to D_{i-1}^0$, $i > 0$
3.2: $\hat{q}_{l,j}^0 \to q_{l,j}^-$, $l > 0$	3.6: $\hat{A}_{i,l}^0 \to A_{i,l}^-$
3.3: $\hat{q}_{0,j}^0 \to q_j^0$	3.7 : $D_0^+ \to \varepsilon^-$
3.4: $\hat{A}_l^+ \to a_l^0$	

Processor 4 has polarization (charge) −1 and uses substitution rules to contribute to the simulation of deletion rules of M only:

Deletion: $q_s a_k \to q_j$
4.1: $a_l^0 \to A_{0,l}^-$
4.2: $X_s^- \to D_s^+$
4.3: $\hat{q}_j^- \to q_j^0$, $j > 0$
4.4: $\hat{q}_0^- \to q_0^+$

Processor 2 has polarization (charge) −1 and only contains the single insertion rule 2.1: $\lambda \to A_0^+$.

Processor 5 has polarization (charge) −1 and only contains the single deletion rule 5.1: $\varepsilon^- \to \lambda$. Processor 6 has polarization (charge) +1 and only contains the single deletion rule 6.1: $q_0^+ \to \lambda$.

The proof closely follows the idea from [1] (Theorem 2), which is itself based on the rotate-and-simulate method. We recall the main steps of that proof below.

The configuration of M is represented as $q_s w$, $s \ge 0$, where q_s is the current state. Suppose that $q_s \to q_j a_k$ is the associated instruction. Then the following evolution is performed in Γ (for readability, we omit the superscripts (charges) of the symbols):

$$q_s w \Rightarrow^* q_{k,j} w A_0 \Rightarrow^* q_{k-1,j} w A_1 \Rightarrow^* q_{k-t,j} w A_t \Rightarrow^* q_{0,j} w A_k \Rightarrow^* q_j w a_k$$

As in the classical rotate-and-simulate method, A_0 is appended to the string and then the indices of $q_{k,j}$ (respectively A_0) are decreased (respectively increased) simultaneously. When the first index of $q_{k,j}$ reaches zero, its initial value is stored as an index of A_k, allowing to produce the right symbol a_k afterwards.

The instruction $q_s a_k \to q_j$ is simulated in the following way:

$$q_s a_k w \Rightarrow^* D_s A_{0,k} w \Rightarrow^* D_{i-1} A_{1,k} w \Rightarrow^* D_{i-t} A_{i,k} w A \Rightarrow^* \\ \Rightarrow^* D_0 A_{s,k} w \Rightarrow^* \varepsilon A_{s,k} w \Rightarrow^* \varepsilon q_j w \Rightarrow q_j w$$

Here the state symbol q_s is replaced by D_s and the first symbol a_k by $A_{0,k}$. Then in a loop the index of D decreases, while the first index of A increases. At the end of this loop the string $D_0 A_{s,k} w$ is obtained, hence the information about the state s has been transferred to the symbol A, so it now encodes the state and the current symbol of the machine M. Based on this information, the new state q_j is chosen. Finally, symbol D_0 is transformed to symbol ε, which is further deleted.

We remark that it could be possible that another symbol from the string is transformed to $A_{0,k}$ (not necessarily the first one). In this case the computation will not yield a valid result because the state symbol will not be present in the first position and the corresponding symbol ε will never be erased, see [1] for more details.

Now we explain the simulation in more details. We start with the remark that in each step only one symbol of a string w can be changed (by substitution, insertion or deletion) yielding w'. This implies that $|\phi(w) - \phi(w')| \leq 2$. In many cases this allows us to predict the change in the polarization (and thus the communication to another node), based on the above difference and the current node polarization.

Assume that the string $q_s^0 w$ is present in node 1. First, we suppose that there is an instruction $q_s \to q_j a_k$ in M. Then, only the rule 1.1 is applicable, producing the string $q_{k,j}^- w$. Since the initial string had neutral polarization, this rule application changes the polarization of the string to negative and during the communication step this string is sent to nodes 2, 4 and 5. In node 5 there is no rule applicable to this string, so it will never exit this node. In node 4, rule 4.1 can be applied several times, but this will further decrease the value of the string, which will remain negative, so it will never be able to get out of nodes 4 and 5.

In node 2 the insertion rule 2.1 is applied yielding $q_{k,j}^- w A_0^+$. Clearly, this string has a neutral polarization, so it will return back to node 1. Next, we discuss the evolution of strings of form $q_{k-t,j}^- w A_t^+$, $0 \leq t \leq k-2$ in node 1:

$$\begin{aligned} q_{k-t,j}^- w A_t^+ &\Rightarrow_{1.2} q_{k-t-1,j}^0 w A_t^+ \Rightarrow_{3.1} q_{k-t-1,j}^0 w A_{t+1}^0 \Rightarrow_{1.5} \\ &\Rightarrow_{1.5} \hat{q}_{k-t-1,j}^0 w A_{t+1}^0 \Rightarrow_{1.3} \hat{q}_{k-t-1,j}^0 w \hat{A}_{t+1}^0 \Rightarrow_{3.2} \\ &\Rightarrow_{3.2} q_{k-t-1,j}^- w \hat{A}_{t+1}^+ \Rightarrow_{1.6} q_{k-t-1,j}^- w A_{t+1}^+ \end{aligned}$$

During first two steps only rules 1.2 and 3.1 are applicable (and they change the polarization of the string). Next, rules 1.3, 1.4 and 1.5 are applicable. It can be easily seen that if 1.3 is applied yielding the string $q_{k-t-1,j}^0 w \hat{A}_{t+1}^+$, then no more applicable rule is present in node 3. If rule 1.4 is applied then the only possible continuation is the application of the sequence of rules 3.4 and 1.5 yielding the string $\hat{q}_{k-t-1,j}^0 w a_{t+1}^0$ in node 1. Clearly, there are no more applicable rules and this string cannot evolve anymore.

So, rule 1.5 has to be applied. Next, there is a choice between the application of 1.3 and 1.4. In case of the application of 1.4, either 3.4 or 3.2 is applicable. The

first application yields to the case discussed before, while the second application produces the following evolution not yielding any result:

$$\hat{q}^0_{k-t-1,j} w \check{A}^+_{t+1} \Rightarrow_{3.2} q^-_{k-t-1,j} w \check{A}^+_{t+1} \Rightarrow_{1.2} q^0_{k-t-2,j} w \check{A}^+_{t+1} \Rightarrow_{3.4} \\ \Rightarrow_{3.4} q^0_{k-t-2,j} w a^0_{t+1} \Rightarrow_{1.5} \hat{q}^0_{k-t-2,j} w a^0_{t+1}$$

So, on the fourth step rule 1.3 should be applied. Then the only applicable rule is 3.2. Now, if rule 1.6 is not applied, then on the next step (after the application of 1.2) no more rules will be applicable to the corresponding string in node 3.

Hence, the procedure described above permits to evolve the string $q^-_{k,j} w A^+_0$ into $q^-_{1,j} w A^+_{k-1}$. Now, the sequence described above produces the string $q^-_{0,j} w A^+_k$, which cannot evolve anymore. However, another evolution now becomes possible (by choosing 1.4 instead of 1.3):

$$q^-_{1,j} w A^+_{k-1} \Rightarrow_{1.2} q^0_{0,j} w A^+_{k-1} \Rightarrow_{3.1} q^0_{0,j} w A^0_k \Rightarrow_{1.5} \hat{q}^0_{0,j} w A^0_{t+1} \Rightarrow_{1.4} \\ \Rightarrow_{1.4} \hat{q}^0_{0,j} w \check{A}^0_k \Rightarrow_{3.3} q^0_j w \check{A}^+_k \Rightarrow_{3.4} q^0_j w a^0_k$$

We remark that if rule 3.4 is applied instead of 3.3, then corresponding string cannot evolve. This concludes the discussion of the simulation of the rule $q_s \to q_j a_k$ of M.

Now consider that there is an instruction $q_s a_k \to q_j$ in M to be simulated. Then, only rule 1.7 is applicable, producing the string $X^-_s w$. Since the initial string had neutral polarization, this rule application changes the polarization of the string to negative and during the communication step this string is sent to the nodes 2, 4 and 5. In node 5 there is no rule applicable to this string, so it will never exit this node. In node 2, rule 2.1 can be applied yielding $X^-_s w A^+_0$ in node 1 to which no further rule is applicable.

In node 4 rules 4.1 and 4.2 are applicable. If 4.2 is applied, then the polarization of the resulting string is positive and the string will be lost. Hence 4.1 should be applied, yielding $X^-_s A^-_{0,k} w'$ ($w = a_k w$). Now again both rules 4.1 and 4.2 are applicable. This time using rule 4.2 allows to obtain the neutral string $D^+_s A^-_{0,k} w'$ which further goes to node 1. In the other case, the corresponding string will always be negative.

Now let us consider the evolution of strings of type $D^+_{s-t} A^-_{t,k} w'$, $0 \le t \le s-1$ being in node 1. Using the same technique as in the case above the decrement of the index of D and the increment of the index of A is performed, with the rules 1.8, 1.9, 1.10, 1.11, 2.5, and 2.6 now having a similar function as the rules 1.2, 1.5, 1.3, 1.6, 2.1, and 2.2. Hence, we obtain:

$$D^+_{s-t} A^-_{t,k} w' \Rightarrow_{1.8} D^+_{s-t} A^0_{t+1,k} w' \Rightarrow_{2.5} D^0_{s-t-1} A^0_{t+1,k} w' \Rightarrow_{1.9} \\ \Rightarrow_{1.9} D^0_{s-t-1} \hat{A}^0_{t+1,k} w' \Rightarrow_{1.10} \hat{D}^+_{s-t-1} \hat{A}^0_{t+1,k} w' \Rightarrow_{2.6} \\ \Rightarrow_{2.6} \hat{D}^+_{s-t-1} A^-_{t+1,k} w' \Rightarrow_{1.6} \hat{D}^+_{s-t-1} A^-_{t+1,k} w' \Rightarrow_{1.11} D^+_{s-t-1} A^-_{t+1,k} w'$$

It can easily be verified that the few possible variations of the computation above (not using rule 1.9 or using 1.12 instead of 1.9) immediately yield strings that cannot evolve anymore. Hence, we obtain that from $D_s^+ A_{0,k}^- w'$ only the string $D_0^+ A_{s,k}^- w'$ can be obtained (in node 1). At this point two rules are applicable: 1.8 and 1.12. Using rule 1.8 yields $D_0^+ A_{s+1,k}^0 w'$ in node 3, where only rule 3.7 is applicable, yielding $\varepsilon^- A_{s+1,k}^0 w'$. However, the last string is negative, so it is lost during the communication step. The other possibility gives the following evolution:

$$D_0^+ A_{s,k}^- w' \Rightarrow_{1.12} D_0^+ \hat{q}_j^+ w' \Rightarrow_{3.7} \varepsilon^- \hat{q}_j^+ w' \Rightarrow_{1.13} \varepsilon^- \hat{q}_j^- w' \Rightarrow_{4.3} \varepsilon^- q_j^0 w' \Rightarrow_{5.1} q_j^0 w'$$

We remark that last two operations can be done in a reverse order (if the string first travels to node 5 and then to node 4). The additional application of rule 4.1 traps the string in nodes 4 and 5.

Finally, we show how a terminal string is obtained as a result. We can assume that the last instruction of M is an instruction of type $q_s a_k \to q_0$. Then rule 4.4 produces the word $q_0^+ w'$, which being positive is sent to node 6, where rule 6.1 is applied producing a neutral string w', which further arrives in node 7.

Corollary 1. *For any CPM5 $M = (\Sigma, T, Q, q_1, q_0, R)$ in normal form there exists a NePEP with terminal extraction with five cells $\Gamma = (V, T, H, \mathcal{R}, \phi_s, i_1, i_0)$ being able to simulate the computations of M.*

Proof. The assertion can be easily proved by deleting nodes 6 and 7 from the previous construction, as well as by adding the rule $q_0^0 \to \varepsilon^-$ to processor 1.

4 Conclusion and Future Research

In this paper we have improved the number of cells necessary to obtain computational completeness with networks of polarized evolutionary processors with elementary polarizations $-1, 0, 1$ of symbols to seven. In the case of not requiring a special output node and just taking all terminal strings as results even only five nodes have been shown to be sufficient.

The construction given in this paper, like the previous ones for networks of polarized evolutionary processors makes intensive use of the control given by the structure of the communication graph. On the other hand, in [1] the results were obtained for several specific regular graph structures as complete graphs, star-like and even linear graphs. Hence, an interesting question for future research arises when asking for the ingredients and the number of cells needed to obtain computational completeness for variants of networks of polarized evolutionary processors based on such specific graph structures. Finally, we also may look for reducing the number seven (five for the case of terminal extraction) of cells needed to obtain computational completeness with networks of polarized evolutionary processors with elementary polarizations $-1, 0, 1$ of symbols.

References

1. Alhazov, A., Freund, R., Rogozhin, V., Rogozhin, Yu.: Computational completeness of complete, star-like, and linear hybrid networks of evolutionary processors with a small number of processors. Natural Computing 15(1), 51–68 (2016)
2. Alhazov, A., Krassovitskiy, A., Rogozhin, Yu.: Circular Post machines and P systems with exo-insertion and deletion. In: Gheorghe, M., Păun, Gh., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7184, pp. 73–86. Springer (2011)
3. Alhazov, A., Martín-Vide, C., Rogozhin, Yu.: On the number of nodes in universal networks of evolutionary processors. Acta Informatica 43(5), 331–339 (2006)
4. Alhazov, A., Martín-Vide, C., Rogozhin, Yu.: Networks of evolutionary processors with two nodes are unpredictable. In: Loos, R., Fazekas, S.Z., Martín-Vide, C. (eds.) LATA 2007. Proceedings of the 1st International Conference on Language and Automata Theory and Applications. Report, vol. 35/07, pp. 521–528. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2007)
5. Arroyo, F., Canaval, S., Mitrana, V., Ş. Popescu: On the computational power of networks of polarized evolutionary processors. Information and Computation 253(3), 371–380 (2017)
6. Arroyo, F., Canaval, S.G., Mitrana, V., Popescu, Ş.: Networks of polarized evolutionary processors are computationally complete. In: International Conference on Language and Automata Theory and Applications. pp. 101–112. Springer (2014)
7. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Solving NP-complete problems with networks of evolutionary processors. In: Mira, J., Prieto, A. (eds.) Connectionist Models of Neurons, Learning Processes and Artificial Intelligence, 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15, 2001, Proceedings, Part I. Lecture Notes in Computer Science, vol. 2084, pp. 621–628. Springer (2001)
8. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta informatica 39(6-7), 517–529 (2003)
9. Freund, R., Rogozhin, Yu., Verlan, S.: Generating and accepting P systems with minimal left and right insertion and deletion. Natural Computing 13(2), 257–268 (2014)
10. Manea, F., Martín-Vide, C., Mitrana, V.: Accepting networks of evolutionary word and picture processors: A survey. Scientific Applications of Language Methods 2, 525–560 (2010)
11. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Computing and Combinatorics, pp. 290–299. Springer (2002)
12. Popescu, S.: Networks of polarized evolutionary processors with elementary polarization of symbols. In: NCMA 2016. pp. 275–285 (2016)
13. Post, E.L.: Formal reductions of the general combinatorial decision problem. American Journal of Mathematics 65(2), 197–215 (1943)
14. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)
15. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1-3. Springer (1997)