Filter Position in Networks of Evolutionary Processors Does Not Matter: A Direct Proof

Paolo Bottoni[1], Anna Labella[1], Florin Manea[2,⋆], Victor Mitrana[2,3,⋆], and Jose M. Sempere[3,⋆⋆]

[1] Department of Computer Science, "Sapienza" University of Rome
Via Salaria 113, 00198 Rome, Italy
{bottoni,labella}@di.uniroma1.it
[2] Faculty of Mathematics, University of Bucharest
Str. Academiei 14, 70109 Bucharest, Romania
{flmanea,mitrana}@fmi.unibuc.ro
[3] Department of Information Systems and Computation
Technical University of Valencia,
Camino de Vera s/n. 46022 Valencia, Spain
jsempere@dsic.upv.es

Abstract. In this paper we give a direct proof of the fact that the computational power of networks of evolutionary processors and that of networks of evolutionary processors with filtered connections is the same. It is known that both are equivalent to Turing machines. We propose here a direct simulation of one device by the other. Each computational step in one model is simulated in a constant number of computational steps in the other one while a translation via Turing machines squares the time complexity.

1 Introduction

The origin of accepting networks of evolutionary processors (ANEP for short) is a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [5] as well as the Logic Flow paradigm [4], which consists of several very simple processors (called evolutionary processors), each of them being placed in a node of a virtual complete graph. By an evolutionary processor we mean an abstract 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

⋆ Work supported by the PN-II Programs 11052 (GlobalComp) and 11056 (CellSim). Victor Mitrana acknowledges support from Academy of Finland, project 132727.
⋆⋆ Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769.

R. Deaton and A. Suyama (Eds.): DNA 15, LNCS 5877, pp. 1–11, 2009.

node is organized in the form of multisets of words (each word may appear 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. Further, all the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies modelled as permitting and forbidding filters and filtering criteria, see [7]. The reader interested in a more detailed discussion about the model is referred to [7,6]. In [7] one shows that this model is computationally complete and presents a characterization of the complexity class **NP** based on accepting networks of evolutionary processors (ANEP for short).

It is clear that filters associated with each node of an ANEP allow a strong control of the computation. Indeed, every node has an input and output filter; two nodes can exchange data if it passes the output filter of the sender *and* the input filter of the receiver. Moreover, if some data is sent out by some node and not able to enter any node, then it is lost. The ANEP model considered in [7] is simplified in [2] by moving the filters from the nodes to the edges. Each edge is viewed as a two-way channel such that the input and output filters, respectively, of the two nodes connected by the edge coincide. Clearly, the possibility of controlling the computation in such networks seems to be diminished. For instance, there is no possibility to lose data during the communication steps. In spite of this fact, in the aforementioned work one proves that these new devices, called accepting networks of evolutionary processors with filtered connections (ANEPFC) are still computationally complete. This means that moving the filters from the nodes to the edges does not decrease the computational power of the model. Although the two variants are equivalent from the computational power point of view, no direct proof for this equivalence has been proposed so far. It is the aim of this paper to fill this gap. We mention that both simulations presented here are time efficient, namely each computational step in one model is simulated in a constant number of computational steps in the other. This is particularly useful when one wants to translate a solution from one model into the other. A translation via a Turing machine squares the time complexity of the new solution.

2 Basic Definitions

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

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

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

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

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

$\alpha \in \{*, l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

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

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 DNA mutations.

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

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

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, the first condition requires that all permitting symbols are present in w and no forbidding symbol is present in w, while the second one is a weaker variant of the first, requiring that at least one permitting symbol appears in w and no forbidding symbol is present in w. For every language $L \subseteq V^*$ and $\beta \in \{(s), (w)\}$, we define:

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

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

- 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.
- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor. Informally, the permitting input/output contexts are the set of symbols that should be present in a word, when it enters/leaves the processor, while the

forbidding contexts are the set of symbols that should not be present in a word in order to enter/leave the processor.

We denote the set of evolutionary processors over V by EP_V. Obviously, the evolutionary processor described here is a mathematical concept similar to that of an evolutionary algorithm, both being inspired by the Darwinian evolution. 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 [10]. Furthermore, we are not concerned here with a possible biological implementation of these processors, though a matter of great importance.

An *accepting network of evolutionary processors* (ANEP for short) is a 7-tuple $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, x_I, x_O)$, where:

- V and U are the input and network alphabets, respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph, with the set of nodes X_G and the set of edges E_G. Each edge is given in the form of a binary set. G is called the *underlying graph* of the network.
- $\mathcal{N} : X_G \longrightarrow EP_U$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $\beta : X_G \longrightarrow \{(s), (w)\}$ 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:
$$\text{input filter: } \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x),$$
$$\text{output filter: } \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x).$$
That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.
- x_I and $x_O \in X_G$ are the *input node*, and the *output node*, respectively, of the ANEP.

An *accepting network of evolutionary processors with filtered connections* (ANEPFC for short) is a 8-tuple $\Gamma = (V, U, G, \mathcal{R}, \mathcal{N}, \alpha, \beta, x_I, x_O)$, where:

- V, U, G, α, x_I, and x_O have the same meaning as for ANEPs.
- $\mathcal{R} : X_G \longrightarrow 2^{Sub_U} \cup 2^{Del_U} \cup 2^{Ins_U}$ is a mapping which associates with each node the set of evolutionary rules that can be applied in that node. Note that each node is associated only with one type of evolutionary rules, namely for every $x \in X_G$ either $\mathcal{R}(x) \subset Sub_U$ or $\mathcal{R}(x) \subset Del_U$ or $\mathcal{R}(x) \subset Ins_U$ holds.
- $\mathcal{N} : E_G \longrightarrow 2^U \times 2^U$ is a mapping which associates with each edge $e \in E_G$ the disjoint sets $\mathcal{N}(e) = (P_e, F_e)$, $P_e, F_e \subset U$.
- $\beta : E_G \longrightarrow \{s, w\}$ defines the *filter* type of an edge.

For both variants we say that $card(X_G)$ is the size of Γ.

A *configuration* of an ANEP or ANEPFC Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration

may be understood as the sets of words which are present in any node at a given moment. A configuration can change either by an *evolutionary step* or by a *communication step*.

An evolutionary step is common to both models. 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 $C \Longrightarrow C'$, if and only if

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

When changing by a communication step, each node processor $x \in X_G$ of an ANEP sends one copy of each word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x provided 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 $C \vdash C'$, if and only if

$$C'(x) = (C(x) - \tau_x(C(x))) \ \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y)))$$

for all $x \in X_G$. Note that words which leave a node are eliminated from that node. If they cannot pass the input filter of any node, they are lost.

When changing by a communication step, each node processor $x \in X_G$ of an ANEPFC sends one copy of each word it has to every node processor y connected to x, provided they can pass the filter of the edge between x and y. It keeps no copy of these words but receives all the words sent by any node processor z connected with x providing that they can pass the filter of the edge between x and z.

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

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

for all $x \in X_G$. Note that a copy of a word remains in the sending node x only if it not able to pass the filter of any edge connected to x.

Let Γ be an ANEP or ANEPFC, 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 defined by $C_0^{(w)}(x_I) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G, x \neq x_I, C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. A computation as above is said to be an *accepting computation* if there exists a configuration in which the set of words existing in the output node x_O is non-empty. The *language accepted* by Γ is

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

We denote by $\mathcal{L}(ANEP)$ and $\mathcal{L}(ANEPFC)$ the class of languages accepted by ANEPs and ANEPFCs, respectively.

3 A Direct Simulation of ANEPs by ANEPFCs

Theorem 1. $\mathcal{L}(ANEP) \subseteq \mathcal{L}(ANEPFC)$.

Proof. Let $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, x_1, x_n)$ be an ANEP with the underlying graph $G = (X_G, E_G)$ and $X_G = \{x_1, x_2, \ldots, x_n\}$ for some $n \geq 1$. We construct the ANEPFC $\Gamma' = (V, U', G', \mathcal{R}, \mathcal{N}', \alpha', \beta', x_1^s, x_n^s)$, where

$U' = U \cup \{X_i, X^d \mid X \in U, i \in \{1, \ldots, n\}\} \cup \{\$_i \mid i \in \{1, \ldots, n\}\} \cup \{\#, \$\}$.

The nodes of the graph $G' = (X'_G, E'_G)$, the sets of rules associated with them and the way in which they are applied, as well as the edges of E'_G together with the filters associated with them are defined in the following.

First, for every pair of nodes x_i, x_j from X_G such that $\{x_i, x_j\} \in E_G$ we have the following nodes in Γ'

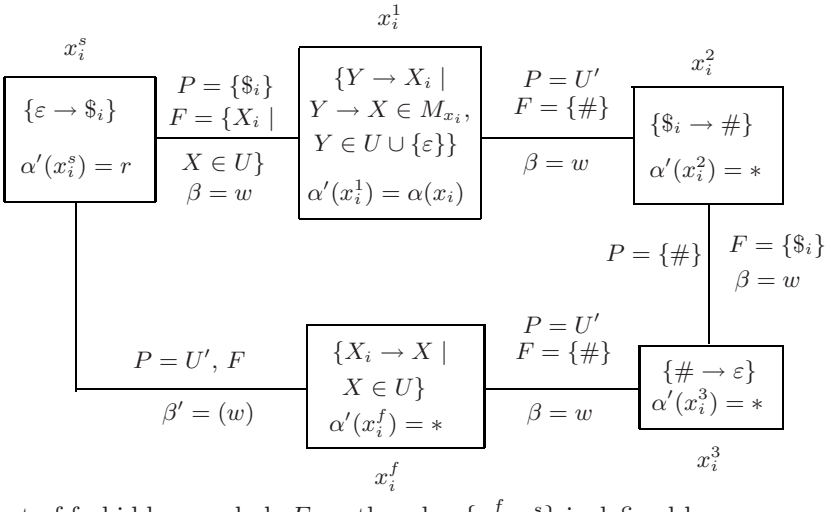
– $x^1_{i,j}$: $R(x^1_{i,j}) = \{\varepsilon \to \$\}$, $\alpha'(x^1_{i,j}) = l$
– $x^2_{i,j}$: $R(x^2_{i,j}) = \{\$ \to \varepsilon\}$, $\alpha'(x^2_{i,j}) = l$,

and the following edges

– $\{x_i^f, x^1_{i,j}\}$: $P = PO(x_i)$, $F = FO(x_i) \cup \{\$\}$, $\beta' = \beta(x_i)$
– $\{x^1_{i,j}, x^2_{i,j}\}$: $P = PI(x_j)$, $F = FI(x_j)$, $\beta' = (w)$,
– $\{x^2_{i,j}, x_j^s\}$: $P = PI(x_j)$, $F = FI(x_i) \cup \{\$, \$_j\}$, $\beta' = \beta(x_j)$.

For each node x_i in Γ we add a subnetwork to Γ' according to the cases considered in the sequel.

Case 1. For an insertion or a substitution node $x_i \in X_G$ with weak filters we have the following subnetwork in Γ'.



The set of forbidden symbols F on the edge $\{x_i^f, x_i^s\}$ is defined by:

$$F = \begin{cases} FO(x_i) \cup PO(x_i) \cup \{\$_i\}, & \text{if } x_i \text{ is an insertion node} \\ PO(x_i) \cup \{\$_i\}, & \text{if } x_i \text{ is a substitution node} \end{cases}$$

Case 2. If x_i is an insertion or a substitution node with strong filters we just add the following nodes to the above construction:

$$x_i^{s,Z} : R(x_i^{s,Z}) = \{\varepsilon \to \$_i\}, \alpha'(x_i^{s,Z}) = *,$$

and the edges

$\{x_i^{s,Z}, x_i^1\} : P = \{\$_i\}, F = \{X_i \mid X \in U\}, \beta' = (w)$

$\{x_i^f, x_i^{s,Z}\} : P = U', F = \begin{cases} FO(x_i) \cup \{Z, \$_i\}, & \text{if } x_i \text{ is an insertion node} \\ \{Z, \$_i\}, & \text{if } x_i \text{ is a substitution node} \end{cases}$

$\beta' = (w)$, for all $Z \in PO(x_i)$.

Case 3. If $x_i \in X_G$ is a deletion node, then the construction in the Case 1 is modified as follows:

– The way of applying the rules in node x_i^s is changed to l if $\alpha(x_i) = r$.
– Parameters of the node x_i^1 are now $R(x_i^1) = \{X \to X^d \mid X \to \varepsilon \in M_{x_i}\}$, $\alpha'(x_i^1) = *$.
– A new node is added: x_i^4 with $R(x_i^4) = \{X^d \to \varepsilon\}$, $\alpha'(x_i^4) = \alpha(x_i)$.
– Parameters of the node x_i^f are now $R(x_i^f) = \{X^d \to X \mid X \in U\}$, $\alpha'(x_i^f) = *$.

In this case, the edges are:
– $\{x_i^s, x_i^1\}$: $P = \{\$_i\}$, $F = \{X^d \mid X \in U\}$, $\beta' = (w)$,
– $\{x_i^1, x_i^2\}$: $P = U'$, $F = \{\#\}$, $\beta' = (w)$,
– $\{x_i^2, x_i^3\}$: $P = \{\#\}$, $F = \{\$_i\}$, $\beta' = (w)$,
– $\{x_i^3, x_i^4\}$: $P = U'$, $F = \{\#\}$, $\beta' = (w)$,
– $\{x_i^4, x_i^s\}$: $P = U'$, $F = \emptyset$, $\beta' = (w)$,
– $\{x_i^f, x_i^s\}$: $P = FO(x_i)$, $F = \{\$_i\}$, $\beta' = (w)$.

Let us follow a computation of Γ' on the input word $w \in V^*$. Let us assume that w lies in the input node x_1^s of Γ'. In the same time, we assume that w is found in x_1, the input node of Γ. Inductively, we may assume that a word w is found in some x_i, a node of Γ, as well as in x_i^s from Γ'.

In the sequel we consider two cases: x_i is a substitution or a deletion node. Since the reasoning for an insertion node is pretty similar to that for a substitution node, it is left to the reader. Let x_i be a substitution node, where a rule $Y \to X$ is applied to w producing either w_1Xw_2, if $w = w_1Yw_2$ or w, if w doesn't contain Y. Here is the first difference with respect to an insertion node where every rule that could be applied is actually applied. In Γ', the word w is processed as follows. First w becomes $w\$_i$ in x_i^s, then it can enter x_i^1 only. Here it may become $w_1X_iw_2\$_i$, if $w = w_1Yw_2$, or it is left unchanged. Further, $w\$_i$ can go back to x_i^s, where another $\$_i$ symbol is added to its righthand end. Then it returns to x_i^1 and the situation above is repeated. When x_i is an insertion node, then this "ping-pong" process cannot happen. On the other hand, $w_1X_iw_2\$_i$ enters x_i^2. It is worth mentioning that any word arriving in x_i^2 contains at most one occurrence of X_i for some $X \in U$. In x_i^2, all the symbols $\$_i$ are replaced by $\#$ which is to be deleted in x_i^3. Finally, the current word enters x_i^f where the symbol X_i, if any, is rewritten into X. Thus, in node x_i^f we have obtained the word w_1Xw_2, if $w = w_1Yw_2$, or w if w doesn't contain Y; all the other words

that may be obtained during these five steps either lead to the same word in x_i^f or have no effect on the rest of the computation.

The second case to be considered is when x_i is a deletion node containing a rule $Y \to \varepsilon$; we will assume that this node is a left deletion node, all the other cases being treated similarly. In this node, the word w is transformed into w', if $w = Yw'$, or is left unchanged, otherwise. In Γ' the word is processed as follows. First, in x_1^s a symbol $\$_i$ is inserted in the rightmost end of the word. Then the word enters x_i^1, where it is transformed into $w_1Y^dw_2\$_i$ (if $w = w_1Yw_2$, for all the possible $w_1, w_2 \in U^*$) or $w\$_i$ (if Y doesn't occur in w). After this step, $w\$_i$ goes back to x_i^s, where another $\$_i$ symbol is added. It then returns to x_i^1 and the situation above is repeated. On the other hand, all words $w_1Y^dw_2\$_i$ enter x_i^2. Again, we mention that every word arriving in x_i^2 contains at most one occurrence of X^d for some $X \in U$. Here all the symbols $\$_i$ are replaced by #. The words can now enter x_i^3 only, where all the symbols # are deleted. Further they go to node x_i^4, where the symbol X^d is deleted, provided that it is the leftmost one. Otherwise, they are left unchanged. Then each obtained word goes to x_i^f, where it is transformed back into w, if the symbol X^d was not deleted in the previous node, or can be left unchanged. If the word still contains X^d, then it goes back to node x_i^4 and the above considerations can be applied again. If the word obtained doesn't contain any X^d, then it is either w', where $w = Yw'$, or w; all the other words that we may obtain during these six steps either lead to the same word in x_i^f or have no effect on the rest of the computation.

In conclusion, if $w \in U^*$ is a word in the nodes x_i of Γ and x_i^s of Γ', then we can obtain $w' \in U^*$ in one processing step of Γ if and only if we can obtain w' in the node x_i^f of Γ' in 5 processing steps (if x_i is an insertion or substitution node) or in 6 processing steps (if x_i is a deletion node). At this point we note that w' can leave x_i and enters x_j in Γ if and only if w' can leave x_i^f and enters x_j^s via the nodes $x_{i,j}^1$ and $x_{i,j}^2$. If w' can leave x_i but cannot enter x_j in Γ, then it is trapped in $x_{i,j}^1$ in Γ'. Finally, if w' cannot leave node x_i, then it is resent by x_i^f to x_i^s (in the case of deletion nodes, and insertion and substitution nodes with weak filters) or to the nodes $x_i^{s,Z}$, for all $Z \in PO(x_i)$ (in the case of insertion and substitution nodes with strong filters); from this point the process described above is repeated, with the only difference that in the case of insertion and substitution nodes with strong filters, the role of node x_i^s is played by the nodes $x_i^{s,Z}$.

From the above considerations, it follows that Γ' simulates in at most 6 processing steps and 5 communication steps a processing step of Γ, and in another 2 processing steps and 3 communication steps a communication step of Γ. We conclude that $L_a(\Gamma) = L_a(\Gamma')$. □

4 A Direct Simulation of ANEPFCs by ANEPs

Theorem 2. $\mathcal{L}(ANEPFC) \subseteq \mathcal{L}(ANEP)$.

Proof. Let $\Gamma = (V, U, G, \mathcal{R}, \mathcal{N}, \alpha, \beta, x_1, x_n)$ be an ANEPFC with $G = (X_G, E_G)$, X_G having n nodes $x_1, x_2, \ldots, x_n$. We construct the ANEP $\Gamma' = (V, U', G', \mathcal{N}',$

α', $\beta', x_I, x_O)$, where
– $U' = V \cup X \cup \{Y\}, X = \{X_{i,j} \mid 1 \leq i \neq j \leq n, i \neq n,$ and $\{x_i, x_j\} \in E_G\}$
– $G' = (X'_G, E'_G)$,
– $X'_G = \{x_I, x_O\} \cup \{x_{i,j}, x'_{i,j} \mid 1 \leq i \neq j \leq n, i \neq n,$ and $\{x_i, x_j\} \in E_G\}$
– $E'_G = \{\{x_I, x_{1,i}\} \mid 2 \leq i \leq n\} \cup \{\{x_{i,j}, x'_{i,j}\} \mid 1 \leq i \neq j \leq n, i \neq n\} \cup$
$\{\{x'_{i,j}, x_{j,k}\} \mid 1 \leq i \neq j \leq n, 1 \leq j \neq k \leq n\} \cup \{\{x'_{i,n}, x_O\} \mid 1 \leq i \leq n-1\},$

and the other parameters defined as follows:

• node x_I: $M = \{\varepsilon \to X_{1,i} \mid 2 \leq i \leq n\}$,
– $PI = V$, $FI = X$, $PO = X$, $FO = \emptyset$,
– $\alpha' = *$, $\beta' = (w)$.
• nodes $x_{i,j}$, $1 \leq i \neq j \leq n, i \neq n$: $M = \mathcal{R}(x_i)$,
– $PI = \{X_{i,j}\}$, $FI = X \setminus \{X_{i,j}\}$, $PO = P_{\{x_i,x_j\}}$, $FO = F_{\{x_i,x_j\}}$,
– $\alpha' = \alpha(x_i)$, $\beta' = \beta(\{x_i, x_j\})$.
• nodes $x'_{i,j}$, $1 \leq i \neq j \leq n, i \neq n$:
– $M = \begin{cases} \{X_{i,j} \to X_{j,k} \mid 1 \leq k \leq n, k \neq j\}, & \text{if } j < n \\ \{X_{i,j} \to Y\}, & \text{if } j = n \end{cases}$
– $PI = \{X_{i,j}\}$, $FI = X \setminus \{X_{i,j}\}$, $PO = (X \cup \{Y\}) \setminus \{X_{i,j}\}$, $FO = \emptyset$,
– $\alpha' = *$, $\beta' = (w)$.
• node x_O: $M = \emptyset$, $PI = \{Y\}$, $FI = \emptyset$, $PO = \emptyset$, $FO = \emptyset$,
– $\alpha' = *$, $\beta' = (s)$.

Any computation in Γ' on an input word $w \in V^+$ produces in x_I all words $w_1 X_{1,i} w_2$ with $w_1, w_2 \in V^*$ such that $w = w_1 w_2$ and $2 \leq i \leq n$ provided that $\{x_1, x_i\} \in E_G$. Each word containing $X_{1,i}$ enters $x_{1,i}$. In a more general setting, we assume that a word $y_1 X_{i,j} y_2$, $y_1, y_2 \in V^*$, enters $x_{i,j}$ at a given step of the computation of Γ' on w. This means that $y = y_1 y_2$ enters x_i at a given step of the computation of Γ on w. Let y be transformed into $z = z_1 z_2$ in node x_i and z can pass the filter on the edge between x_i and x_j. Let us further assume that y_p is transformed into z_p, $p = 1, 2$. This easily implies that $y_1 X_{i,j} y_2$ is transformed into $z_1 X_{i,j} z_2$ in node $x_{i,j}$ and $z_1 X_{i,j} z_2$ can pass the output filter of $x_{i,j}$. Note that the converse is also true. Now, $z_1 X_{i,j} z_2$, $j \neq n$, enters $x'_{i,j}$ where all words $z_1 X_{j,k} z_2$, with $1 \leq k \neq j \leq n$ and $\{x_j, x_k\} \in E_G$, are produced. Each word $z_1 X_{j,k} z_2$ enters $x_{j,k}$ and the process of simulating the computation in Γ resumes. On the other hand, $z_1 X_{i,n} z_2$ enters $x'_{i,n}$ where $X_{i,n}$ is replaced by Y. All words produced in $x'_{i,n}$, for some $1 \leq i \leq n-1$, enter x_O and the computation ends. Note that by the considerations above, a word enters $x'_{i,n}$ if and only if a word from x_i was able to pass the filter on the edge between x_i and x_n in Γ.

Note that two consecutive steps (evolutionary and communication) in Γ are simulated by four steps (two evolutionary and two communication) in Γ'. □

5 Final Remarks

The simulations presented above may lead to underlying graphs of the simulating networks that differ very much from the underlying graphs of the simulated

networks. However, it looks like there is some form of duality between edges and nodes in the simulations. In network theory, some types of underlying graphs are common like *rings, stars, grids*, etc. Networks of evolutionary words processors, seen as language generating or accepting devices, having underlying graphs of these special forms have been considered in several papers, see, e.g., [8] for an early survey. Simulations preserving the type of the underlying graph of the simulated network represent a matter of interest that is not considered here.

On the other hand, in almost all works reported so far ANEPs and ANEPFCs have a complete underlying graph. Starting from the observation that every ANEPFC can be immediately transformed into an equivalent ANEPFC with a complete underlying graph (the edges that are to be added are associated with filters which make them useless), we may immediately state that Theorem 1 holds for complete ANEPs and ANEPFCs as well. Although it is true that every ANEP is equivalent to a complete ANEP (every Turing machine can be simulated by a complete ANEP), we do not know a simple and direct transformation as that for ANEPFCs. Therefore, direct simulations preserving the type of the underlying graph remain to be further investigated.

The language decided by an ANEP and ANEPFC is defined in [7] and [2], respectively. It is easy to note that the construction in the proof of Theorem 2 works for a direct simulation of ANEPFCs halting on every input. However, in the proof of Theorem 1, Γ' doesn't detect the non-accepting halting computations of Γ, since configurations obtained in consecutive processing steps of Γ are not obtained here in consecutive processing steps. Thus Γ' doesn't necessarily decide the language decided by Γ. It is known from the simulation of Turing machines by ANEPs and ANEPFCs [6,3] that the languages decided by ANEPs can be also decided by ANEPFCs. Can the construction from the proof of Theorem 1 be modified for a direct simulation of ANEPs halting on every input? Finally, as the networks of evolutionary picture processors introduced in [1] do not have insertion nodes, it would be of interest to find direct simulations for these devices.

References

1. Bottoni, P., Labella, A., Mitrana, V., Sempere, J.: Networks of evolutionary picture processors (submitted)
2. Drăgoi, C., Manea, F., Mitrana, V.: Accepting networks of evolutionary processors with filtered connections. Journal of Universal Computer Science 13, 1598–1614 (2007)
3. Drăgoi, C., Manea, F.: On the descriptional complexity of accepting networks of evolutionary processors with filtered connections. International Journal of Foundations of Computer Science 19, 1113–1132 (2008)
4. Errico, L., Jesshope, C.: Towards a new architecture for symbolic processing. In: Artificial Intelligence and Information-Control Systems of Robots 1994, pp. 31–40. World Scientific, Singapore (1994)
5. Hillis, W.: The Connection Machine. MIT Press, Cambridge (1985)

6. Manea, F., Martin-Vide, C., Mitrana, V.: On the size complexity of universal accepting hybrid networks of evolutionary processors. Mathematical Structures in Computer Science 17, 753–771 (2007)
7. Margenstern, M., Mitrana, V., Perez-Jimenez, M.: Accepting hybrid networks of evolutionary systems. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005)
8. Martín-Vide, C., Mitrana, V.: Networks of evolutionary processors: results and perspectives. In: Molecular Computational Models: Unconventional Approaches, pp. 78–114. Idea Group Publishing, Hershey (2005)
9. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin (1997)
10. Sankoff, D., et al.: Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. In: Proc. Natl. Acad. Sci. USA, vol. 89, pp. 6575–6579 (1992)