Journal of Applied Mathematics and Computing (2022) 68:553–569
https://doi.org/10.1007/s12190-021-01542-9

ORIGINAL RESEARCH

Simulating polarization by random context filters in networks of evolutionary processors

Victor Mitrana[1,2] · **Jose Angel Sanchez Martin**[1]

Received: 12 January 2021 / Revised: 14 March 2021 / Accepted: 25 March 2021 / Published online: 1 April 2021

Abstract
Networks of evolutionary processors (NEP for short) form a class of models within the new computational paradigms inspired by biological phenomena. They are known to be theoretically capable of solving intractable problems. So far, there are two main categories that differ from each other by the nature of filtering process controlling the communication step: random-context clauses or polarization. Several studies have proven that both of them are computationally complete through efficient simulations of universal computational models such as Turing machines and 2-tag systems. Nevertheless, the indirect conversion between the two network variants results in an exponential increase of the computational complexity. In this paper, we suggest a direct simulation of polarized NEP through NEP with random-context filters which incurs in lower complexity costs.

Keywords Evolutionary processor · Network of evolutionary processors · Polarization · Random context filter

Mathematics Subject Classification 68Q04 · 68Q07 · 68Q10

This work was supported by a grant of the Romanian Ministry of Education and Research, CCCDI-UEFISCDI, Project No. PN-III-P2-2.1-PED-2019-2391, within PNCDI III.

✉ Victor Mitrana
victor.mitrana@upm.es

Jose Angel Sanchez Martin
joseangel.sanchez.martin@alumnos.upm.es

[1] Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, C/Alan Turing s/n, 28031 Madrid, Spain

[2] National Institute for Research and Development of Biological Sciences, Independentei Bd. 296, 060031 Bucharest, Romania

1 Introduction

In the last two decades, new computational paradigms taken after natural behaviors and phenomena have been of great interest for the researching community. The aim of considering these new paradigms has been to overcome the limitations encountered by the classic computational models. Theoretical scrutiny of many proposed systems has proven their feasibility for the efficient solving of intractable problems. Networks of bio-inspired processors embody a subcategory of massively parallel and distributed computational paradigms identified by two properties: their inspiration from nature and their exceptional parallel performance (in theory, unlimited). These networks are analogous to several computational paradigms: tissue-like P systems [22] in the membrane computing area [27], evolutionary systems abstracted from the evolution of various cell populations [9], networks of parallel language processors, which has been introduced as a parallel language-theoretic model [8], flow-based programming, which is a programming model widely known [25], connection machine, which may be viewed as a network, in the shape of a hypercube, consisting of microprocessors that process one bit per unit time [12], distributed computing using mobile programs [26], etc.

Informally, networks of bio-inspired processors can be visualized as graphs with processors positioned in the vertices manipulating information from a wide spectrum of data types: strings, pictures, graphs, multisets, etc. Thus far, research on two main categories of networks of bio-inspired processors handling strings, namely splicing and evolutionary processors, has been reported. The former [15], emulating a computational counterpart of the splicing procedure for DNA molecules regulated by different types of restriction and ligase enzymes [11], was considered as a theoretical prototype for solving problems in [16]. The latter, introduced in [7], is a mathematical construction designed for performing one of the following simple operations inspired by the point mutations in DNA sequences: insertion, deletion or substitution of a single base pair. In this paradigm, each node performs the computational counterpart of a cell with genetic information encoded in DNA sequences which may evolve by local evolutionary events. Networks of evolutionary processors (NEP) have been considered as tools for producing and accepting languages as well as means for problem resolution in [7,18,21], respectively. Characterization of the complexity clases NP, P and PSPACE according to accepting NEPs has been reported in [14], universal NEPs and descriptional complexity problems are examined in [13,17]. An early survey of the conclusions regarding NEPs is available in [19]. The computation in a network of evolutionary processors follows after a succession of alternate steps (evolution and communication) which is terminated once the calculations reach a predefined status. Firstly, every processor yields new strings by means of application of all evolutionary rules that can be applied to duplicates of the strings the processor hosts in. Thus, a node actually contains enough string replications in order to allow the synchronous execution of each applying rule. This simultaneous process is labeled as an evolutionary step. Next in order, the newly obtained data is concurrently shared between the connected nodes in the following way, designated as a communication step: (i) the data hosted by a processor leaves that processor after meeting the output requirements associated to the processor; (ii) each of the adjacent nodes receives a copy of the

previous data and accepts it based on its input specifications. The information flow according to these communication strategies is managed through the introduction of filters deciding on the acceptance or refusal of any string attempting to enter or exit a node. Initially, two different filters have been considered in the literature based on the nature of their limitations: random-contexts and polarization. Furthermore, three versions of random-context filters have been discussed: (i) a pair of different filters to handle the incoming (input) and exiting strings (output) [21], (ii) the same filters for both cases [6] and (iii) the two filters of every pair of adjacent nodes collapse in an unique filter associated to the edge between them [10]. Later on, several variants of filters that are generalizations of those defined by random-context conditions have been considered for both accepting and generating networks of evolutionary processors. These filters are defined by different classes of regular languages, see [20] and the references therein. It is worth mentioning that filters defined by regular languages permit to networks with at most two types of nodes be still sufficiently powerful, see [23].

The polarized branch of networks of evolutionary processors was introduced in [1]. In this version, the filters based on random-context requirements are replaced with a filtering strategy built around the concept of polarization. A generalization of the model of this kind of networks of polarized evolutionary processors (NPEP) was examined in [2]. The paradigm proposes the removal of the node filters in favor of a filtering process established upon a polarity affinity between the nodes and strings present in the network. A node has a polarity in the range $\{-, 0, +\}$. On the other hand, the computation for the strings polarization is done by a valuation mapping which is a morphism between two monoids: the free monoid of all strings and the additive monoid of integers. The communication strategy is abstracted from the electromagnetic theory. The polarization of the string is taken after the sign of the integer value computed by this function. Therefore, the acceptance of a string by a node relies on the polarization of the node and the string which in this model is required to be the same.

We say a simulation between two computational paradigms preserves the time complexity if there is a constant factor between the numbers of computational steps required by each model to achieve any desired result. Although the NPEP and NEPRC are determined to be computationally complete because of the complexity-preserving simulations of Turing machines in [3,14], respectively, an indirect simulation between both of them and their variants through the intermission of a Turing machine produces a great spike in the complexity making this procedure unfeasible. Research for efficient conversion procedures between the first and third submodels of NEPRC is documented in [5] and further extended to include the subcategory (ii) with the investigations in [6]. Consequently, it is proven the three variants are equivalent from the computational point of view. This paper attempts to advance further on this line of research with the proposition of a direct simulation of networks of polarized evolutionary processors by networks of evolutionary processors with random-context filters (NEPRC). The converse simulation has been reported in [24].

2 Definitions and notations

We define the basic concepts and notations employed in this paper. The reader can refer to [28] for all unexplained notions. A finite and nonempty set of symbols is denoted as an *alphabet*. Given a finite set S, its cardinality is denoted by $card(S)$. Any finite sequence of symbols from an *alphabet* V is called *string* over V. The set of *strings* over V is denoted by V^* and the empty string is denoted by ε. The length of a string w is symbolized by $|w|$ while $alph(w)$ denotes the minimal alphabet W such that $w \in W^*$. Furthermore, $|w|_a$ specifies the number of occurrences of the symbol a in w. Further on, the absolute value of an integer k is denoted by $|k|$.

We now recall some definitions from a few papers where the networks of evolutionary processors have been introduced, see, e.g., [7], for the generating model, and [13,14,17], for the accepting model which will be further considered here. A rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$, $a \neq b$ 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 string $w \in V^*$, we define the following *actions* of σ on w:

– If $\sigma \equiv a \to b \in Sub_V$, then

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

$$\sigma^r(w) = \begin{cases} \{ub \mid w = ua)\}, \\ \{w\}, \ \text{otherwise.} \end{cases}$$

$$\sigma^l(w) = \begin{cases} \{bu \mid w = au)\}, \\ \{w\}, \ \text{otherwise.} \end{cases}$$

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

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

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

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

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

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

In the aforementioned definitions, $\alpha = \{l, r, *\}$ conveys the limitations in the application of a rule σ to the string w, namely in the left ($\alpha = l$), in the right ($\alpha = r$) or at any position ($\alpha = *$). Given a substitution rule $\sigma \equiv a \rightarrow b$ with $\alpha = *$, its application to a string w with k occurrences of a yields the set of all k different strings that can be obtained from w depending on the position in w where the rule is applied. As a special case, if $k = 0$, applying the rule doesn't yield any new string and thus, the string w remains unmodified. A deletion and insertion rule can be considered as a specific substitution rule with $b = \varepsilon$ and $a = \varepsilon$ respectively. Consequently, a deletion rule that deletes a at any position in a string w with k occurrences of a, as above, generates all the strings with one occurrence of a removed in different positions in w. Analogously, an insertion rule of a at any position in w produces $|w| + 1$ new strings as a consequence of the possible insertion of the symbol a in the beginning of w, the end of w, and at any intermediate positions between two consecutive letters of w. For $\alpha = l$ (resp. $\alpha = r$), a deletion rule $a \rightarrow \varepsilon$ applied to a string w produces a new string if a is the first symbol of w (resp. final symbol). Otherwise, the string w remains unchanged. Analogously, an insertion rule $\varepsilon \rightarrow a$ applied to w yields a new string by concatenating a to the left (resp. right) of w.

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 and non-empty set of rules M, we define the α-action of M on the string w and the language L by:

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

respectively. In the original papers referenced above, the rewriting operations defined above were designated as *evolutionary operations* since they may viewed as formal operations abstracted from local DNA mutations.

Let V be an alphabet, we define the following predicates for a string $w \in V^+$ and two disjoint subsets P, F of V:

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

In the prior definitions, the set P (permitting contexts) includes the allowed symbols while F (forbidding contexts) encompasses those which are forbidden. Both clauses require the nonexistence of any symbol $a \in F$ in the string w. The former specification imposes stronger (s) restrictions since it requires that all permitting symbols are present in w while the latter demands a weaker (w) condition of at least one permitting symbol appearing in w.

Given a $\beta \in \{w, s\}$, we generalize the above predicates to a language $L \subseteq V^*$ by

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

An *evolutionary processor* with random context filters (EPRC) over an alphabet 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 contexts are sets of symbols that should be present in a string for it to be allowed to enter/leave the processor, while the forbidding contexts are set of symbols that are required to not exist in a string entering/leaving the processor.

A homomorphism from the monoid V^* into the monoid (group) of additive integers $\mathbb{Z}$ is called *valuation* of V^* in $\mathbb{Z}$. We now recall the definitions of polarized evolutionary processors and networks of polarized evolutionary processors from [1] and [2].

A *polarized evolutionary processor* (PEP for short) over an alphabet V is a pair (M, π), where:

– M is a set of substitution, deletion, or insertion rules following the same prescriptions defined for the EPRCs.
– $\pi \in \{-, +, 0\}$ is the polarization of the node (negatively or positively charged, or neutral, respectively).

We denote the set of evolutionary processors with random context filters and polarized evolutionary processors over V by $EPRC_V$ and PEP_V, respectively.

A *network of evolutionary processors with random context filters* (NEPRC for short) is a 8-tuple $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, \underline{In}, \underline{Out})$, 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. Each edge is given in the form of a binary set. G is called the *underlying graph* of the network.
– $\mathcal{N} : X_G \rightarrow EPRC_U$ is a mapping which associates with each node $x \in X_G$ an evolutionary processor with random context filters $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
– $\alpha : X_G \rightarrow \{l, r, *\}$; $\alpha(x)$ gives the action mode of the rules of node x on the strings existing in that node.
– $\beta : X_G \rightarrow \{(s), (w)\}$ 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. $\tau_x(w)$) indicates whether or not the string w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp: $\tau_x(L)$) is the set of strings of L that can pass the input (resp. output) filter of x.
– $\underline{In}$ and $\underline{Out} \in X_G$ are the *input* node, and the *output* node, respectively, of the NEPRC.

A *network of polarized evolutionary processors* (NPEP for short) is a 8-tuple $\Gamma = (V, U, G, \mathcal{N}, \alpha, \varphi, \underline{In}, \underline{Out})$, where:

- $V, U, G, \alpha, \underline{In}, \underline{Out}$ follow the same specifications as the parameters in NEPRC, while $\mathcal{N} : X_G \rightarrow PEP_U$ associates with each node $x \in X_G$ the polarized evolutionary processor $\mathcal{N}(x) = (M_x, \pi_x)$.
- φ is a valuation of U^* in $\mathbf{Z}$.

We denote the number of nodes of the underlying graph Γ as the size of the network and specify it by $card(X_G)$. A *configuration* of Γ is a mapping $C : X_G \longrightarrow 2^{U^*}$ which associates a set of strings $C(x)$ with every node x of Γ. A configuration $C(x)$ may be understood as the support sets of the multsets of strings which are present in a node x, in an arbitrarily large number of copies, at a given moment. Given a string $w \in U^*$, the initial configuration of Γ is defined by $C_0^{(w)}(\underline{In}) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G \backslash \underline{In}$.

A configuration can be modified either by an *evolutionary step* or by a *communication step*. In an evolutionary step, each component $C(x)$ of the configuration C is altered according to a set of evolutionary rules M, associated with the node x. This process is identical for the two variants of networks of evolutionary processors, namely NPEP and NPERC. Formally, a configuration C' is achieved from a previous one C through the execution of an evolutionary step, denoted as $C \Longrightarrow C'$, if and only if:

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

After each evolutionary step the network proceeds on with the communication step which is different for each network variant.

In the communication step of a NEPRC the subsequent actions take place simultaneously for every node $x \in X_G$:

- The data contained in a node x leaves it, granted that they meet the requirements set by the output filters PO and FO of that node.
- The previous data enter the nodes connected to the source node x, provided that they are not blocked by the input filters PI and FI of the receiving nodes. Otherwise, these strings are lost.

Formally, a configuration C' follows a configuration C after a communication step (we write $C \models C'$) if for all $x \in X_G$

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

A communication step in an NPEP is defined as follows for each $x \in X_G$:

- Copies of the data present in a node x are sent to each neighbouring node y connected to it. From these strings, x only keeps a copy of the strings having the same polarity to that of the node.
- All the nodes receiving the incoming data sent by x take in a copy of the strings with the same polarity while unaccepted copies are lost.

Note that, for the sake of simplicity, we follow a same polarity strategy to decide on the acceptance or refusal of a string w by a node x. Formally, the configuration C' is

produced from a configuration C as a consequence of a communication step, written as $C \models C'$, if:

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

for all $x \in X_G$. Here, $sign(m)$ is the sign of the integer m.

Let Γ be a $NEPRC/NPEP$ the computation of Γ on the input string $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ on w, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \models C_{2i+2}^{(w)}$, for all $i \geq 0$. Note that the configurations are changed by alternative steps.

A computation as above *halts*, if there exists a configuration in which the set of strings existing in the output node $\underline{Out}$ is non-empty. Given a $NEPRC/NPEP\Gamma$ and an input string w, we say that Γ accepts w if the computation of Γ on w halts. Therefore, we define the *language* accepted by Γ by

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

The *time complexity* of the halting computation $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots, C_m^{(w)}$ of Γ on $w \in V^*$ is denoted by $time_\Gamma(z)$ and equals m. The time complexity of Γ is the function from $\mathbf{N}$ to $\mathbf{N}$, $Time_\Gamma(n) = max\{time_\Gamma(w) \mid w \in L(\Gamma), |w| = n\}$. In other words, $Time_\Gamma(n)$ delivers the maximal number of computational steps done by Γ for accepting an input string of length n.

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

$$Time_{NEPRC/NPEP}(f(n)) = \{L \mid \text{ there exists a NEPRC/NPEP } \Gamma \text{ which accepts } L, \text{ such that } Time_\Gamma(n) \leq f(n), \text{ for all } n \geq n_0, \text{ for some } n_0 \geq 1.\}$$

3 Efficient simulation of NPEP by NEPRC

The predominant purpose of this research is a direct simulation of NPEP by NEPRC that induces a time complexity improvement in comparison to an indirect simulation through an intermediate Turing machine.

Theorem 3.1 *For every NPEP Γ, there exists a NEPRC Γ' such that $L(\Gamma) = L(\Gamma')$. Moreover, $size(\Gamma)(card(U) + 15) \leq size(\Gamma') \leq size(\Gamma)(3card(U) + 15)$.*

Proof Let $\Gamma = (V, U, G, \mathcal{N}, \alpha, \varphi, In, Out)$ be a NPEP with an arbitrary underlying graph G. The working alphabet U' of Γ' is defined as follows:

$$U' = U \cup \{a_i \mid a \in U, 1 \leq i \leq |\varphi(a)|\} \cup \{a' \mid a \in U, \varphi(a) \neq 0\} \cup \{\bar{a} \mid a \in U\} \cup \{\mathrm{T}, \mathrm{T}', \mathrm{T}'', R_+, R_-\}.$$

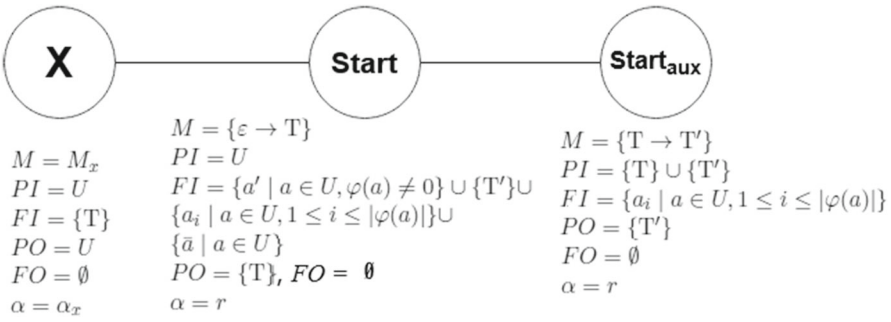

Fig. 1 Subnetwork 1: setup

We associate with every node x of Γ the subnetwork of Γ' composed in its turn by the subnetworks illustrated in figures Figs. 1, 2, 3 and 4. It is worth mentioning that all filters of Γ' are weak filters. These subnetworks are connected as follows in Γ:

(i) The four subnetworks associated with a node x of Γ are connected such that the nodes $Start_{aux}$ in the subnetworks in Figs. 1 and 2 is the same node, while the nodes $Check$ in Figs. 2, 3 and 4 is the same node.
(ii) Each node x of Γ is connected to y of Γ provided that $\{x, y\}$ is an edge in G.

Let is suppose that $w \in U^*$ leaves the node x at a given moment in Γ. We consider now that string w enters $Start$ from x of Γ' at that moment. In this node, the symbol T is inserted in the end of w. Note that the action mode of the insertion rule $\varepsilon \to \mathrm{T}$ is in the right. Further on, the string $w\mathrm{T}$ enters $Start_{aux}$, where the symbol T is ultimately replaced by T', blocking the route back of $w\mathrm{T}'$ to $Start$.

For each symbol $a \in U$, we distinguish two cases: $\varphi(a) \neq 0$ and $\varphi(a) = 0$. Let us consider the case $\varphi(a) \neq 0$, for some $a \in U$. From $Start_{aux}$, the string $w\mathrm{T}'$ enters $Start_a$. Here, an occurrence of a is replaced by a_1. From now on, a ping-pong process between the nodes $Start_a$ and $Insert_a$ starts: each time when the string is in $Start_a$, either an occurrence of a is replaced by a_1 or a_i is replaced by a_{i+1}, provided that $1 \leq i < |\varphi(a)|$, while whenever the string is in $Insert_a$, a symbol a' is inserted in its end. The total effect of this ping-pong process is that all occurrences of a are eventually replaced by $a_{|\varphi(a)|}$ and a total number of $|\varphi(a)| \cdot |w|_a$ of occurrences of a' are appended. When all occurrences of a are replaced by $a_{|\varphi(a)|}$, the strings enter $Limit_a$, where all these occurrences are replaced by $\bar{a}$. If the strings still contains letters from U, the strings cannot enter $Check$, but they go back to $Start_{aux}$, where the process described above is resumed for another letter in U.

Returning to the two cases, if $\varphi(a) = 0$, then the strings enter the node $Limit_a$ associated with this case, where all occurrences of a are replaced directly by $\bar{a}$. In conclusion, from $w\mathrm{T}'$ one obtains all the strings $w'\mathrm{T}'u$, where w' is obtained from w by replacing each occurrence of any symbol $a \in U$ by $\bar{a}$ and u is any permutation of the strings in the set

$$\{(a')^{|w|_a \cdot |\varphi(a)|} \mid a \in U, \varphi(a) \neq 0\}.$$

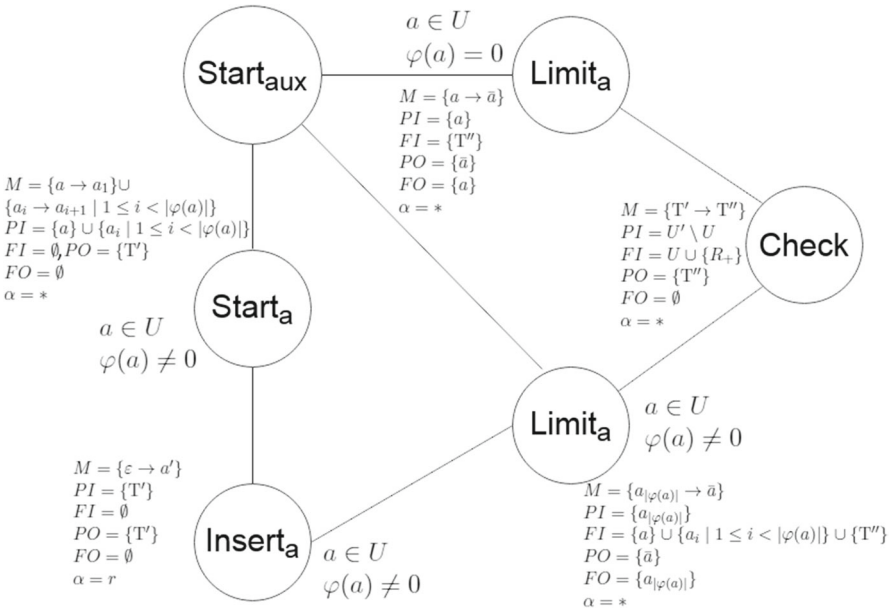

Fig. 2 Subnetwork 2: introducing $\varphi(a)$ information

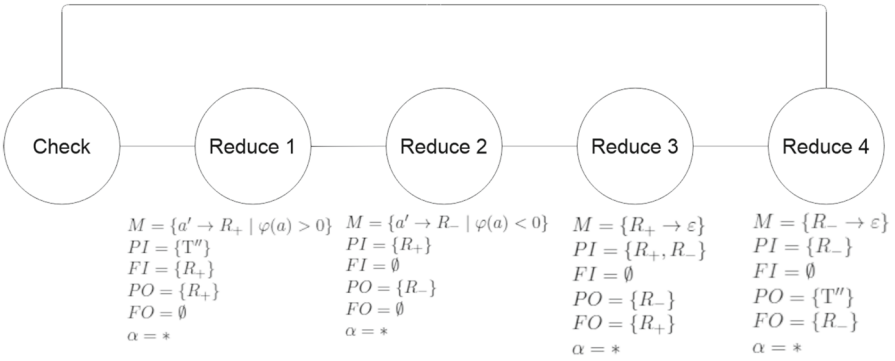

Fig. 3 Subnetwork 3: simplification

The aforementioned behavior carries on until all symbols $a \in U$ are replaced by $\bar{a}$. Once this requirement is met, the strings enter $Check$ and T′ is changed to T″. Note that all these strings also return to $Start_{aux}$, but they are ultimately lost in the next communication step after not meeting the requirements imposed by the filters of the connected nodes.

The computation continues on with the simplification process handled by the third subnetwork illustrated in Fig. 3. Let us assume that a string z exits $Check$ and enters $Reduce1$. In this node one occurrence of a symbol a' with $\varphi(a) > 0$ is changed to R_+, and the new string leaves $Reduce1$ for entering $Reduce2$. Here, an occurrence of a symbol b' with $\varphi(b) < 0$ is replaced with R_-. These two new symbols are removed in

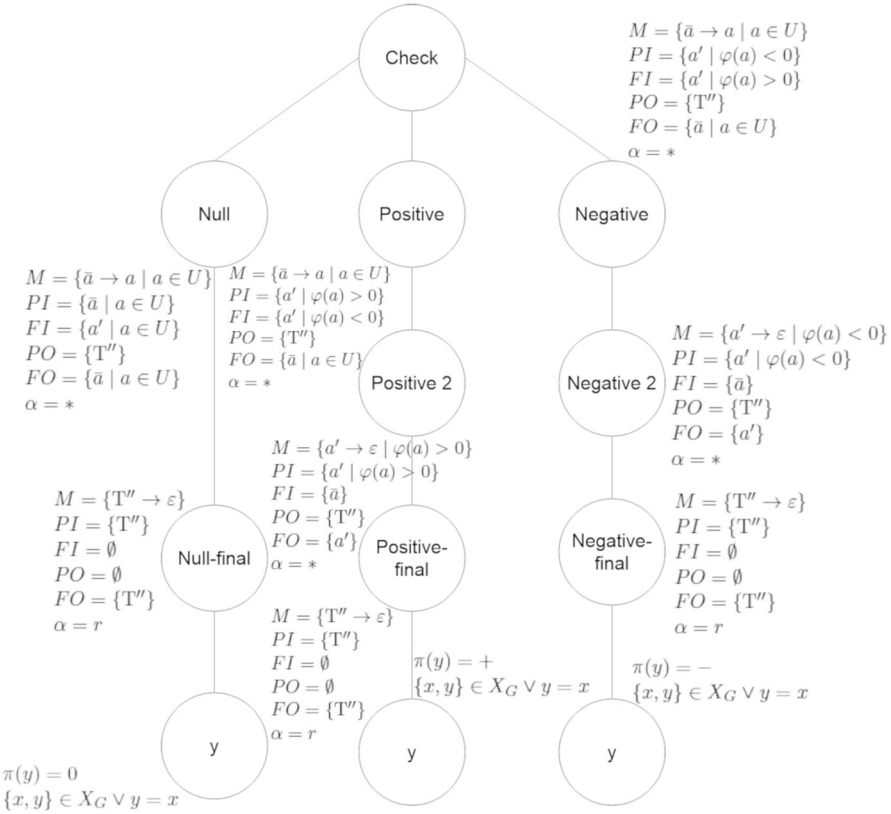

Fig. 4 Subnetwork 4: polarization choice and string reset

$Reduce3$ and $Reduce4$ and the new string is returned back to $Check$. Consequently, z is simplified until it does not contain two symbols a', b' with $\varphi(a) \cdot \varphi(b) < 0$. Note that any string which does not contains a pair of symbols a', b' with $\varphi(a) \cdot \varphi(b) < 0$ can still enter $Reduce1$ but it ends up being blocked in either $Reduce1$ or $Reduce2$.

Lastly, the simulation ends with the steps executed in the subnetwork portrayed in Fig. 4. From $Check$ a string can enter either $Null$ or $Positive$ or $Negative$. It can enter $Null$ provided that it does not contain any symbol a', $a \in U$. It can enter $Positive$ if it still contains a' with $\varphi(a) > 0$ after the computational steps in the third network, but no b' with $\varphi(b) < 0$. Finally, it can enter $Negative$ if it only contains a' with $\varphi(a) < 0$, but no b' with $\varphi(b) > 0$. In all of these nodes, the symbols $\bar{a}$ return to the original $a \in U$. Hereafter, the symbols a' are removed in the $Positive2$ and $Negative2$ nodes. Then, the simulation ends by removing the special symbol T'' and the strings, which collapse now to w, are sent to the nodes of the original network Γ given that the node polarity is the same as that considered in each branch, namely $+$ for $Positive$, $-$ for $Negative$ and 0 for $Null$.

In this simulation, the symbols $a \in U$ are replaced by their counterparts $\bar{a} \in U'$ without altering their position in w and returned back to the original state in the last

subnetwork before connecting with the nodes of Γ. Similarly, the special symbols T and a' used for the polarization simulation are removed at the end of the computation. Therefore, the same string w ultimately enters the common nodes of the networks Γ and Γ'. From these propositions, we conclude $L(\Gamma) = L(\Gamma')$.

Finally, it is easier to evaluate the $size(\Gamma')$. For each node x of Γ the new network Γ' has four subnetworks: Subnetwork 1 which has 2 new nodes (the node x belongs already to Γ), Subnetwork 2 has at most $3card(U) + 1$ nodes (note that $Start_{aux}$ has been counted with Subnetwork 1), but at least $card(U) + 1$ nodes, Subnetwork 3 has 4 nodes, while Subnetwork 4 has 8 nodes, hence a total of at most $15 + 3card(U)$ nodes, but at least $15 + card(U)$ nodes. Therefore, $size(\Gamma)(card(U) + 15) \leq size(\Gamma') \leq size(\Gamma)(3card(U) + 15)$ holds.

Theorem 3.2 *For every NPEP Γ, with $Time_\Gamma(n) \in \mathcal{O}(f(n))$, there exists a NEPRC Γ' which directly simulates Γ and $Time_{\Gamma'}(n) \in \mathcal{O}(P \cdot f(n)(n + f(n)))$, where P is the biggest absolute value of the valuation mapping φ of Γ.*

Proof We evaluate now the number of evolutionary steps performed by Γ' for simulating a communication in Γ.

Let w be the string that enters $Start$ in the simulation of a communication step in Γ. The Subnetwork 1 always requires a constant number of 2 evolutionary steps. The number of evolutionary steps in Subnetworks 1 and 2 is at most $2 + |w| * (2P + 1) + 1$, hence $\mathcal{O}(P|w|)$. The computation of Subnetwork 3 requires $\mathcal{O}(P|w|)$ evolutionary steps as well. Lastly, the computation continues in exactly one of the three mutually disjoint branches starting with the nodes: $Null$, $Positive$, and $Negative$, respectively. Thus, the Subnetwork 4 requires a number of $\mathcal{O}(|w|)$, $\mathcal{O}(P|w|)$, and $\mathcal{O}(P|w|)$ evolutionary steps, respectively. Now, for an input string of length n, the length of any string moving throughout the network at any step is bounded by $f(n) + n$. Therefore, each simulation of a communication step takes $\mathcal{O}(P(n + f(n)))$ steps in Γ'. Consequently, $Time_{\Gamma'}(n) \in \mathcal{O}(P \cdot f(n)(n + f(n)))$.

We now compare our direct simulation with an indirect simulation via a Turing machine. To this aim, we recall the following two statements from [4,14,17], respectively.

Theorem 3.3 *1. Every NPEP working within $\mathcal{O}(f(n))$ time can be simulated by a nondeterministic Turing machine in $\mathcal{O}(f(n)(f(n) + n)^2)$.*
2. Every nondeterminstic Turing machine working within $\mathcal{O}(f(n))$ time can be simulated by an NEPRC in $\mathcal{O}(f(n)$ time.

From these statements it immediately follows that every NPEP working within $\mathcal{O}(f(n))$ time can be simulated by an NEPRC in $\mathcal{O}(f(n)(f(n) + n)^2)$ time. As one can easily see, the direct simulation proposed by us in this note is far more efficient than the indirect one.

An immediate and attractive question arises: Is the P factor necessary? Can it be discarded? We are going to answer this question in the next section.

Table 1 Changes in Subnetwork 1

Node	Rules	PI	FI	PO	FO	α
$Start$	$\{\varepsilon \to \mathrm{T}\}$	U	$\{\mathrm{T}'\} \cup \{\bar{a} \mid a \in U\} \cup$ $\{a_{\lvert\varphi(a)\rvert} \mid a \in U, \varphi(a) \neq 0\}$	$\{\mathrm{T}\}$	$\emptyset$	r
$Start_{aux}$	$\{\mathrm{T} \to \mathrm{T}'\}$	$\{\mathrm{T}, \mathrm{T}'\}$	$\{a_{\lvert\varphi(a)\rvert}\}$	$\{\mathrm{T}'\}$	$\emptyset$	r

Table 2 Changes in Subnetwork 2

Node	Rules	PI	FI	PO	FO	α
$Start_a$	$\{a \to a_{\lvert\varphi(a)\rvert}\}$	$\{a\}$	$\emptyset$	$\{\mathrm{T}'\}$	$\emptyset$	$*$
$Insert_a$	$\{\varepsilon \to R_{\varphi(a)}\}$	$\{\mathrm{T}'\}$	$\emptyset$	$\{\mathrm{T}'\}$	$\emptyset$	r
$Limit_a$	$\{a_{\lvert\varphi(a)\rvert} \to \bar{a}\}$	$\{R_{\varphi(a)}\}$	$\{a, \mathrm{T}''\}$	$\{\bar{a}\}$	$\{a_{\lvert\varphi(a)\rvert}\}$	$*$

4 Another efficient simulation

The answer is yes, the factor P can be removed at the price of a larger size of the simulating network. In what follows, we discuss this construction.

Theorem 4.1 *For every NPEP* $\Gamma = (V, U, G, \mathcal{N}, \alpha, \varphi, \underline{In}, \underline{Out})$ *working in* $f(n)$ *time, there exists a NEPRC* Γ' *such that* $L(\Gamma) = L(\Gamma')$ *and* $Time_{\Gamma'}(n) \in \mathcal{O}(f(n)(f(n) + n))$. *Moreover,* $size(\Gamma)(13 + 4P + card(U)) \leq size(\Gamma') \leq size(\Gamma)(13 + 4P + 3card(U))$, *where* $P = \max\{|\varphi(a)| : a \in U\}$.

Proof The construction of Γ' is very similar to that from the proof of Theorem 3.1. We shall just point out the differences. The structure of Subnetwork 1 remains the same, but the description of nodes $Start$ and $Start_{aux}$ are to be changed as shown in Table 1.

As one can see, the simulation starts in nodes $Start$ and $Start_{aux}$ exactly as explained in the proof of Theorem 3.1.

The structure of Subnetwork 2 remains unchanged, but the description of nodes $Start_a$, $Insert_a$, and $Limit_a$, for all $a \in U$ such that $\varphi(a) \neq 0$, are to be changed as shown in Table 2.

A similar ping-pong process between the nodes $Start_a$ and $Insert_a$ in the previous construction starts: each time when the string is in $Start_a$, an occurrence of a is replaced by $a_{|\varphi(a)|}$, while whenever the string is in $Insert_a$, a symbol $R_{\varphi(a)}$ is inserted in its end. The total effect of this ping-pong process is that all occurrences of a are eventually replaced by $a_{|\varphi(a)|}$ and a total number of $|w|_a$ of occurrences of $R_{\varphi(a)}$ are appended. When all occurrences of a are replaced by $a_{|\varphi(a)|}$, the strings enter $Limit_a$, where all these occurrences are replaced by $\bar{a}$. If the strings still contains letters from U, the strings cannot enter $Check$, but they go back to $Start_{aux}$, where the process described above is resumed for another letter in U.

In conclusion, from a string $w\mathrm{T}'$ that goes out from $Start_{aux}$ one obtains all the strings $w'\mathrm{T}'u$, where w' is obtained from w by replacing each occurrence of any symbol $a \in U$ by $\bar{a}$ and u is any permutation of the strings in the set $\{R_{\varphi(a)}^{|w|_a} \mid a \in U, \varphi(a) \neq 0\}$.

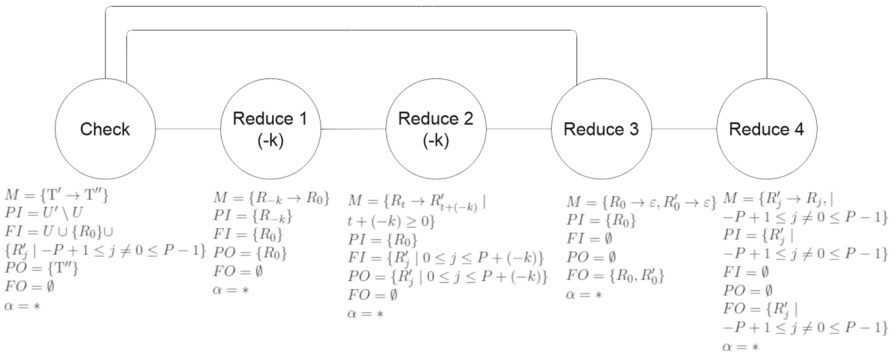

Fig. 5 New Subnetwork 3

The Subnetwork 3 in the proof of Theorem 3.1 is now replaced by a subnetwork having $2P$ branches. We depicted such a branch in Fig. 5, for an arbitrary $-P \le -k \le -1$. Other $2P$ branches whose nodes are called $Reduce1(k)$ and $Reduce2(k)$, $1 \le k \le P$ are defined analogously.

It is important to note that there is an edge between $Reduce3$ and $Reduce4$ and any node $Reduce2(q)$, $-P \le q \ne 0 \le P$.

We explain the role of this subnetwork. Let us assume that a string z exists $Check$ and enters $Reduce1(q)$ for some $-P \le q \ne 0 \le P$. In this node one occurrence of a symbol R_q is changed to R_0, and the new string leaves $Reduce1(q)$ for entering $Reduce2(q)$. Here, an occurrence of a symbol R_t is replaced with R'_{t+q}, provided that $(t+q)q \le 0$. These two new symbols are removed in $Reduce3$ and $Reduce4$ and the new string is returned back to $Check$. Consequently, z is simplified until it does not contain two symbols R_i, R_j with $i \times j < 0$.

Although the structure of Subnetwork 4 remains the same to that in the proof of Theorem 3.1, the parameters of all nodes have to be changed, hence we prefer to give it completely in Fig. 6.

The simulation ends in the subnetwork portrayed in Fig. 6. From $Check$ a string can enter either $Null$ or $Positive$ or $Negative$. It can enter $Null$ provided that it does not contain any symbol R_q, $-P \le q \ne 0 \le P$. It can enter $Positive$ if it still contains R_q with $q > 0$, but no R_q with $q < 0$. Finally, it can enter $Negative$ if it only contains R_q with $q < 0$, but no R_q with $q > 0$. In all of these nodes, the symbols $\bar{a}$ return to the original $a \in U$. Hereafter, the symbols R_q are removed in the $Positive2$ and $Negative2$ nodes. Then, the simulation ends by removing the special symbol T'' and the strings, which collapse now to w, are sent to the nodes of the original network Γ given that the node polarity is the same as that considered in each branch, namely $+$ for $Positive$, $-$ for $Negative$ and 0 for $Null$. From these propositions, we conclude $L(\Gamma) = L(\Gamma')$.

We now evaluate the $size(\Gamma')$. For each node x of Γ the new network Γ' has four subnetworks: Subnetwork 1 which has 2 new nodes (the node x belongs already to Γ), Subnetwork 2 has at most $3card(U)+1$ nodes (note that $Start_{aux}$ has been counted with Subnetwork 1), but at least $card(U)+1$ nodes, Subnetwork 3 has $4P+2$ nodes, while Subnetwork 4 has 8 nodes, hence a total of at most $13+4P+3card(U)$ nodes,

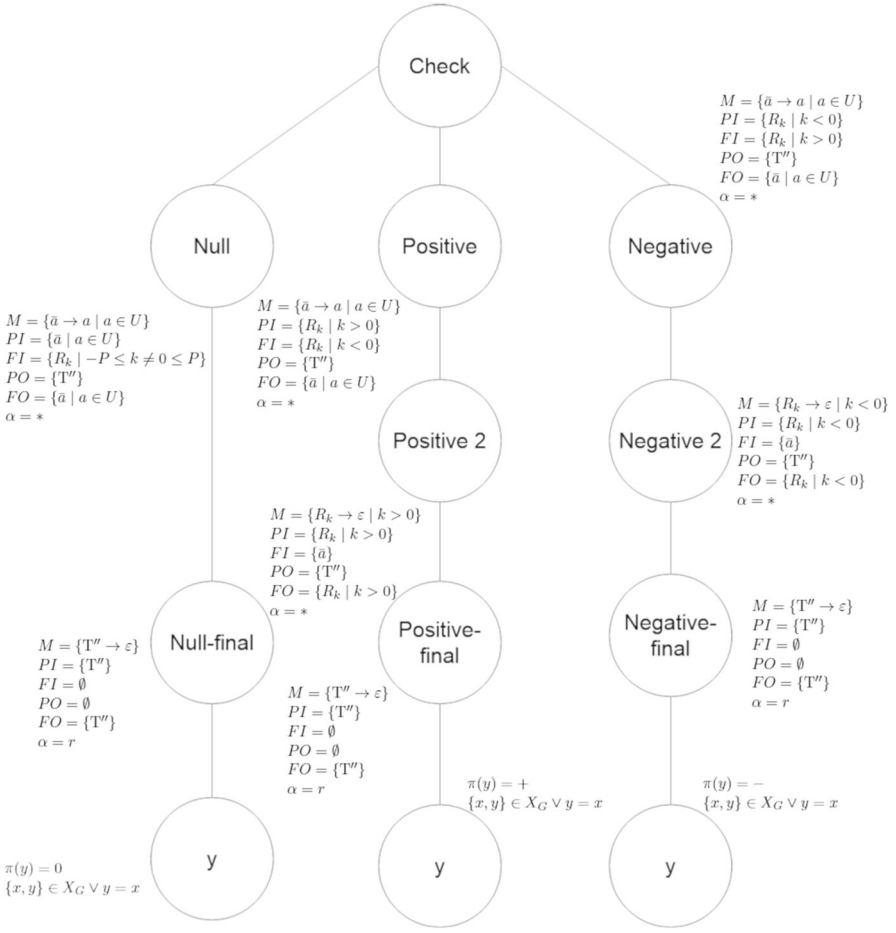

Fig. 6 New Subnetwork 4

but at least $13 + 4P + card(U)$ nodes. Therefore, $size(\Gamma)(card(U) + 13 + 4P) \le size(\Gamma') \le size(\Gamma)(3card(U) + 13 + 4P)$ holds.

Finally, it is easy to note that the computation on an input string w in Subnetworks 1 and 2 together requires $\mathcal{O}(|w|)$ evolutionary steps and the same time is required by the computation in each of the Subnetworks 3 and 4. Therefore, if Γ works in $f(n)$ time, then $Time_{\Gamma'}(n) \in \mathcal{O}(f(n)(n + f(n)))$.

Finally, it is worth mentioning that this direct simulation completes the result reported in [24] that states the following:

Theorem 4.2 *For every NPERC Γ, there exists a NEPP Γ' such that $L(\Gamma) = L(\Gamma')$. Furthermore, $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$ holds.*

5 Conclusion

Direct simulations between different bio-inspired computational models are a matter of interest in the community of the area. We propose here two direct simulations of the communication controlled by polarization with a communication regulated by random-context conditions. The differences between them is given by the trade-off between the size of the networks and running times. A more time efficient simulation requires an increase of the network size. This is the converse simulation of that reported in [24] and completes the picture of the direct simulations initiated in [5] and [6]. Finally, we would like to mention that a similar approach for networks of splicing processors appears very attractive to us.

References

1. Alarcón, P.P., Arroyo, F., Mitrana, V.: Networks of polarized evolutionary processors as problem solvers. In: Advances in Knowledge-Based and Intelligent Information and Engineering Systems Volume 243 of Frontiers in Artificial Intelligence and Applications, pp. 807–815. IOS Press (2012)
2. Alarcón, P.P., Arroyo, F., Mitrana, V.: Networks of polarized evolutionary processors. Inf. Sci. **265**, 189–197 (2014)
3. Arroyo, F., Gómez Canaval, S., Mitrana, V., Popescu, S.: Networks of polarized evolutionary processors are computationally complete. In: Language and Automata Theory and Applications, Volume 8370 of Lecture Notes in Computer Science, pp. 101–112, Springer (2014)
4. Arroyo, F., Gómez Canaval, S., Mitrana, V., Popescu, S.: On the computational power of networks of polarized evolutionary processors. Inf. Comput. **253**, 371–380 (2017)
5. Bottoni, P., Labella, A., Manea, F., Mitrana, V., Sempere, J.M.: Filter position in networks of evolutionary processors does not matter: a direct proof. In: DNA Computing and Molecular Programming, Volume 5877 of Lecture Notes in Computer Science, pp. 1–11, Springer (2009)
6. Bottoni, P., Labella, A., Manea, F., Mitrana, V., Petre, I., Sempere, J.M.: Complexity-preserving simulations among three variants of accepting networks of evolutionary processors. Nat. Comput. **10**, 429–445 (2011)
7. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Inform. **39**, 517–529 (2003)
8. Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In: New Trends in Formal Languages—Control, Cooperation, and Combinatorics, Volume 1218 of Lecture Notes in Computer Science, pp. 299–318. Springer (1997)
9. Csuhaj-Varjú, E., Mitrana, V.: Evolutionary systems: a language generating device inspired by evolving communities of cells. Acta Inform. **36**, 913–926 (2000)
10. Dragoi, C., Manea, F., Mitrana, V.: Accepting networks of evolutionary processors with filtered connections. J. Univ. Comput. Sci. **13**, 1598–1614 (2007)
11. Head, T., Păun, G., Pixton, D.: Language theory and molecular genetics: generative mechanisms suggested by DNA recombination. In: Handbook of Formal Languages, Volume 2. Linear Modeling: Background and Application, pp. 295–360. Springer (1997)
12. Hillis, W.D.: The Connection Machine. MIT Press, Cambridge (1979)
13. Loos, R., Manea, F., Mitrana, V.: Small universal accepting hybrid networks of evolutionary processors. Acta Inform. **47**, 133–146 (2010)
14. 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)
15. Manea, F., Martín-Vide, C., Mitrana, V.: Accepting networks of splicing processors. In: New Computational Paradigms, Volume 3526 of Lecture Notes in Computer Science, pp. 300–309. Springer (2005)

16. Manea, F., Martín-Vide, C., Mitrana, V.: All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size. In: DNA Computing, Volume 4287 of Lecture Notes in Computer Science, pp. 47–57. Springer (2006)
17. Manea, F., Martín-Vide, C., Mitrana, V.: On the size complexity of universal accepting hybrid networks of evolutionary processors. Math. Struct. Comput. Sci. **17**, 753–771 (2007)
18. Manea, F., Mitrana, V.: All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. Inf. Process. Lett. **103**, 112–118 (2007)
19. 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)
20. Manea, F., Truthe, B.: Accepting networks of evolutionary processors with subregular filters. Theory Comput. Syst. **55**, 84–109 (2014)
21. Margenstern, M., Mitrana, V., Pérez-Jiménez, M.J.: Accepting hybrid networks of evolutionary processors. In: DNA Computing, volume 3384 of Lecture Notes in Computer Science, pp. 235–246. Springer (2004)
22. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: tissue P systems. In: 8th Annual International Conference, COCOON 2002, Volume 2387 of Lecture Notes in Computer Science, pp. 290–299. Springer-Verlag (2002)
23. Mitrana, V., Truthe, B.: On accepting networks of evolutionary processors with at most two types of nodes. In: Third International Conference on Language and Automata Theory and Applications LATA 2009, Volume 5457 of Lecture Notes in Computer Science, pp. 588–600. Springer-Verlag (2009)
24. Mitrana, V., Negru, M.C.: Filters defined by random contexts versus polarization in networks of evolutionary processors. Theor. Comput. Sci. (2021). https://doi.org/10.1016/j.tcs.2020.08.003. (**in press**)
25. Morrison, J.P.: Flow-Based Programming: A New Approach to Application Development, 2nd edn. J.P. Enterprises, Ltd, Pune (2010)
26. Nog, S., Rus, D., Cybenko, G., Gray, R., Kotz, D.: Mobile agents: the next generation in distributed computing. In: Proceedings of the 2nd AIZU International Symposium on Parallel Algorithms/Architecture Synthesis, PAS–97, pp. 8–24. Springer (1997)
27. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin (2002)
28. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I–III. Springer-Verlag, Berlin (1997)