axioms

Article

Simulations between Network Topologies in Networks of Evolutionary Processors

José Ángel Sánchez Martín [1] and Victor Mitrana [1,2,*]

1 Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, C/Alan Turing s/n, 28031 Madrid, Spain; joseangel.sanchez.martin@alumnos.upm.es
2 National Institute for Research and Development of Biological Sciences, Independentei Bd. 296, 060031 Bucharest, Romania
* Correspondence: victor.mitrana@upm.es

Abstract: In this paper, we propose direct simulations between a given network of evolutionary processors with an arbitrary topology of the underlying graph and a network of evolutionary processors with underlying graphs—that is, a complete graph, a star graph and a grid graph, respectively. All of these simulations are time complexity preserving—namely, each computational step in the given network is simulated by a constant number of computational steps in the constructed network. These results might be used to efficiently convert a solution of a problem based on networks of evolutionary processors provided that the underlying graph of the solution is not desired.

Keywords: evolutionary processor; network of evolutionary processors; network topology; theory of computation; computational models

check for updates

Citation: Sánchez Martín, J.Á.; Mitrana V. Simulations between Network Topologies in Networks of Evolutionary Processors. *Axioms* **2021**, *10*, 183. https://doi.org/10.3390/axioms10030183

Academic Editor: Cristian S. Calude

Received: 19 July 2021
Accepted: 6 August 2021
Published: 11 August 2021

1. Introduction

Networks of evolutionary processors (NEPs for short) have been extensively investigated in the last two decades since their generative variant has been introduced in [1]. An informal description of a NEP is as follows: it is a graph whose nodes are hosts for some very simple processors inspired by the basic mutations at the DNA nucleotide level, namely insertion, deletion, and substitution. Each processor is able to make just one of these operations on the data existing in the node that hosts it. Data may be organized as strings, multisets, two-dimensional pictures, graphs, etc. In this work, we consider that the data consist of strings. A very important assumption is that each string appears in an arbitrarily large number of identical copies such that if the processor can apply an operation to different sites of a string, the operation is actually applied simultaneously to each of these sites in different copies of the string. Furthermore, if more that one rule can be applied to a string, each rule is applied to a different copy of that string. This process described above is considered to be an evolutionary step. Each evolutionary step alternates with a communication step. In a communication step, all the strings that can leave a node (they can pass the output filter associated with that node) actually leave the node and copies of them enter each node connected to the left node, provided that they can pass the input filter of the arriving node. We say that an input string, which initially is in a designated node, called the input node, is accepted if another designated node, called the output node, is non-empty after a finite number of computational steps (evolution, communication). The complexity of a computation is defined in the usual way.

From the very beginning, NEPs have been proven to be computationally complete models [2,3], such that they have been used to solve hard problems [4]. Several variants have been considered depending on the positions of filters: filters associated with nodes (different filters [3], uniform filters [5], polarization [6]) or filters associated with edges [7]. Later on, several ways of simulating and implementing different variants of these networks have been reported [8–11]. A rather new and attractive direction of research has been to investigate the possibility of simulating directly and efficiently one variant by another without the intermediate step of an extra computational model (Turing machine, tag-system, register machine, etc.) in between, see, e.g., [5].

This work continues this line of research by proposing direct simulations between two NEPs such that the input one is an arbitrary NEP while the output one has a predefined topology that can be a complete graph, a star graph, or a grid. Thus, after a preliminary section with the basic definitions and concepts, we give the construction of a complete NEP equivalent to a given NEP. We continue with another section, where we give such a construction for a star graph and finally a construction for a grid NEP. A short conclusion ends the paper.

2. Basic Definitions

The basic concepts and notations that are to be used throughout the paper are defined in the sequel; the reader may consult [12] for basic concepts that are not defined here. We use the following concepts and notations:

- V^* is the set of all strings formed by symbols in V;
- $|x|$ is the length of string x;
- $\varepsilon \in V^*$ is the empty string, $|\varepsilon| = 0$;
- $alph(x)$ is the minimal alphabet V such that $x \in V^*$.

We now recall some definitions from a few papers where the networks of evolutionary processors have been introduced, see, e.g., [1], for the generating model, and [3,13,14], for the accepting model. Let $a \to b$ be a rule, where $a, b \in (V \cup \{\varepsilon\})$:

- If $a, b \in V$, then the rule is called a *substitution* rule;
- If $a \in V$ and $b = \varepsilon$, then the rule is called a *deletion* rule;
- If $a = \varepsilon$ and $b \in V$, then the rule is called an *insertion* rule.

The set of all substitution, deletion, and insertion rules over V is 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 , to any position ($*$), to the leftmost position (l), and to the rightmost position (r), as explained in the sequel:

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

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

According to this definition, applying a rule to a string may result in a finite number of strings. This implies that in our setting each string may appear in an arbitrarily large number of copies.

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

For every rule σ, $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite and non-empty set of rules M, a string w and a language L, we define the followings:

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

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

For two disjoint subsets P (permitting symbols) and F (forbidding symbols) of an alphabet V and a string z over V, we define the predicates:

$$\begin{array}{llll} \varphi^{(s)}(z;P,F) \equiv & P \subseteq alph(z) & \wedge & F \cap alph(z) = \emptyset \\ \varphi^{(w)}(z;P,F) \equiv & (P \neq \emptyset) \rightarrow (alph(z) \cap P \neq \emptyset) & \wedge & F \cap alph(z) = \emptyset. \end{array}$$

For every language $L \subseteq V^*$ and $\beta \in \{(s),(w)\}$, we define:

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

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

- M is a set of either substitution, or 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;
- $PI, FI \subseteq V$ are the *input* permitting/forbidding symbols of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding symbols of the processor.

We denote the set of evolutionary processors over V by EP_V. A *network of evolutionary processors* (NEP for short) is a seven-tuple $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, \underline{In}, \underline{Out})$, where:

- V and U are the input and network alphabets, respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph without loops, 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 strings existing in that node;
- $\beta : X_G \longrightarrow \{(s),(w)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

  $$\begin{array}{r} \text{input filter: } \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x), \\ \text{output filter: } \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x). \end{array}$$

  That is, $\rho_x(z)$ (resp. $\tau_x(z)$) indicates whether or not the string z 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 NEP.

A *configuration* of a NEP Γ as above is a function $C : X_G \longrightarrow 2^{U^*}$ which associates a multiset of strings $C(x)$ with every node x of Γ. As each string appears in an arbitrarily large number of copies, we work with the support of this multiset. For a string $w \in V^*$, we define the initial configuration of Γ on w by $C_0^{(w)}(\underline{In}) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G \setminus \{\underline{In}\}$.

A configuration is followed by another configuration either by an *evolutionary step* or by a *communication step*. A configuration C' follows a configuration C by an evolutionary step if each component $C'(x)$, for some node x, is the result of applying all the evolutionary rules in the set M_x that can be applied to the strings in the set $C(x)$. Formally, configuration C' follows the configuration C by a an evolutionary step, written as $C \Longrightarrow C'$, if

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

In a communication step of a NEP the following actions take place simultaneously for every node x:

(i) All the strings that can pass the output filter of a node are sent out of that node;

(ii) All the strings that left their nodes enter all the nodes connected to their original ones, provided that they can pass the input filter of the receiving nodes.

Note that, according to this definition, those strings that are sent out of a node and cannot pass the input filter of any node are lost.

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

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

Let Γ be a NEP, 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 $C_{2i+1}^{(w)} \models C_{2i+2}^{(w)}$, by a 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 NEP Γ and an input string w, we say that Γ accepts w if the computation of Γ on w halts. Consequently, we define the *language accepted* by Γ by

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

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

3. Simulating Any NEP with a Complete NEP

Theorem 1. *Given an arbitrary NEP Γ, there exists a complete NEP Γ' such that the following two conditions are satisfied:*

1. $L(\Gamma) = L(\Gamma')$;
2. $Time_{\Gamma'}(n) \in \mathcal{O}(Time_\Gamma(n))$.

Proof. Let $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, x_1, x_n)$ be a NEP with the underlying graph $G = (X_G, E_G)$ and $X_G = \{x_1, x_2, \ldots, x_n\}$ for some $n \geq 1$; $x_1 \equiv \underline{In}$ and $x_n \equiv \underline{Halt}$. We construct the NEP $\Gamma = (V', U', G', \mathcal{N}', \alpha', \beta', x_{start}, x_n^s)$; $x_{start} \equiv \underline{In}$ and $x_n^s \equiv \underline{Halt}$, where

$$\begin{array}{ll} V' = V, & U' = U \cup T, \\ T = \{t_i^l, t_i^r, t_i^{l'}, t_i^{r'}, t_i^{l''}, t_i^{r''} \mid 1 \leq i \leq n\} & \end{array}$$

Note that the underlying graph G' is a complete graph. First, we add the following nodes to G':

- $\underline{\text{node } x_{start}}$: $$\begin{array}{ll} M = \left\{ \begin{array}{l} \{\varepsilon \to t_1^{l''}\}, \text{ if } \alpha(x_1) \neq l \\ \{\varepsilon \to t_1^{r''}\}, \text{ if } \alpha(x_1) = l \end{array} \right. , & \\ PI = \emptyset, & FI = T, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = \left\{ \begin{array}{l} l, \text{ if } \alpha(x_1) \neq l \\ r, \text{ if } \alpha(x_1) = l \end{array} \right. , & \beta = (w). \end{array}$$

- nodes x_i^s, $1 \le i \le n$ (they actually simulate the work of x_i in Γ):

$$\begin{array}{ll} M = M(x_i), & \\ PI = PI(x_i), & FI = FI(x_i) \cup T \setminus \{t_i^l, t_i^r\}, \\ PO = PO(x_i), & FO = FO(x_i), \\ \alpha = \alpha(x_i), & \beta = \beta(x_i). \end{array}$$

For each node x_i, $1 \le i \le n$ in Γ we add a subnetwork to Γ' according to the subsequent cases:

Case 1. If $\alpha(x_i) = l$, the subnetwork is defined as follows (these nodes are used for preparing the string in the aim of processing them in the nodes x_i^s):

- nodes x_i^{Ins}, $1 \le i \le n$:

$$\begin{array}{ll} M = \{\varepsilon \to t_i^{r''}\}, & \\ PI = \{t_i^{l'}\}, & FI = \emptyset, \\ PO = \{t_i^{r''}\}, & FO = \emptyset, \\ \alpha = r, & \beta = (w). \end{array}$$

- nodes x_i^{Del}, $1 \le i \le n$:

$$\begin{array}{ll} M = \{t_i^{l'} \to \varepsilon\}, & \\ PI = \{t_i^{r''}\}, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = l, & \beta = (w). \end{array}$$

- nodes x_i^{Sub}, $1 \le i \le n$:

$$\begin{array}{ll} M = \{t_i^r \to t_j^{r'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^{r'} \to t_i^r\} \cup \{t_i^{r''} \to t_i^r\}, & \\ PI = \{t_i^r, t_i^{r'}, t_i^{r''}\}, & FI = \{t_i^{l'}\}, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$$

Case 2. If $\alpha(x_i) = r$, the subnetwork is analogous to the *Case 1* with the characters l and r interchanged.

Case 3. If $\alpha(x_i) = *$, the subnetwork is defined as follows (the role of these nodes is the same as above, namely to prepare the strings for being processed in the nodes x_i^s):

- nodes x_i^{Sub}, $1 \le i \le n$:

$$\begin{array}{ll} M = \{t_i^r \to t_j^{r'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^l \to t_j^{l'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^{r'} \to t_i^r\} \cup \{t_i^{l'} \to t_i^l\} \cup \{t_i^{l''} \to t_i^l\}, & \\ PI = \{t_i^l, t_i^r, t_i^{l'}, t_i^{r'}, t_i^{l''}\}, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$$

Let w be the input string in Γ. In the input node x_{start}, the character $t_1^{l''}$ is inserted at the beginning of the string if $\alpha(x_1) \in \{r, *\}$, or the character $t_1^{r''}$ is inserted at the end of the string, provided that $\alpha(x_1) = l$. Next, the string enters x_1^{Sub} where the character is replaced with t_1^l and t_1^r, respectively. Then, the string can only enter x_1^s and the simulation starts. Note that the same evolutionary rules applicable in $x_1 \in \Gamma$ are also possible in x_1^s since the special character $t_1^{l''}$ or $t_1^{r''}$ is set up in a way that it does not block the computation of nodes with $\alpha = r$ and $\alpha = l$, respectively. Inductively, we may assume that a string of the form $t_i^l w$ or $w t_i^r$ lies in the node $x_i^s \in \Gamma'$ if and only if the string w lies in the node $x_i \in \Gamma$.

Let w be transformed into w' in the node x_i and sent to the connected nodes to x_i in Γ. Then, a string $t_i^l w'$ or a string $w t_i^r$ is produced in the node x_i^s and sent to the node x_i^{Sub}. Let us analyze the case of a string $t_i^l w'$. The process is analogous for the other string. In x_i^{Sub}, the character t_i^l is replaced with the symbol $t_j^{l'}$, assuming that $\{x_i, x_j\} \in \Gamma$, which ensures the new string can only be accepted by subnetworks j corresponding to nodes x_j connected to x_i in the original network Γ. From here, the process differs in accordance with the value α of the connected node x_j.

- If $\alpha(x_j) = l$, the string can only enter x_j^{Ins} where the symbol $t_j^{r''}$ is appended to it. The new string, $t_j^{l'} w' t_j^{r''}$, continues through x_j^{Del} where $t_j^{l'}$ is removed and x_j^{Sub} where $t_j^{r''}$ is replaced with t_j^r, allowing it to enter the node x_j^s. Since the character t_j^r is at the end of the string, it does not interfere with the application of evolutionary rules at the left of the string;
- If $\alpha(x_j) = r$ or $\alpha(x_j) = *$, the string directly enters x_j^{Sub} and the symbol $t_j^{l'}$ is replaced with t_j^l. Then, the string enters x_j^s. As one can see, the communication step in Γ has been simulated by a constant number of (evolution and communication) steps in Γ'. A new evolutionary step in Γ is now simulated. It follows that $L(\Gamma) = L(\Gamma')$. Furthermore, the number of steps in Γ' for simulating an evolutionary step followed by a communication one in Γ is constant; hence, $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$ holds.

□

4. Simulating Any NEP with a Star NEP

Theorem 2. *Given an arbitrary NEP Γ, there exists a star NEP Γ' such that the following two conditions are satisfied:*

1. $L(\Gamma) = L(\Gamma')$;
2. $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$.

Proof. Let $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, x_1, x_n)$ be a NEP with the underlying graph $G = (X_G, E_G)$ and $X_G = \{x_1, x_2, \ldots, x_n\}$ for some $n \geq 1$; $x_1 \equiv \underline{In}$ and $x_n \equiv \underline{Halt}$. We construct the NEP $\Gamma = (V', U', G', \mathcal{N}', \alpha', \beta', x_{start}, x_n^s)$; $x_{start} \equiv \underline{In}$ and $x_n^s \equiv \underline{Halt}$, where

$$V' = V, \qquad U' = U \cup T,$$
$$T = \{t_i^l, t_i^r, t_i^{l'}, t_i^{r'}, t_i^{l''}, t_i^{r''}, t_i^{l'''}, t_i^{r'''} \mid 1 \leq i \leq n\}$$

The *star* network uses the definitions illustrated above for the *complete* network, with the following modifications:

We add a new node *Star* to the subnetwork which acts as the center of the *star* network.

- $\underline{\text{node } Star}$:
$$\begin{array}{ll} M = \{t_i^l \to t_j^{l'} \mid \{x_i, x_j\} \in \Gamma\} \cup \{t_i^r \to t_j^{r'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^{l'''} \to t_i^l\} \cup \{t_i^{r'''} \to t_i^r\}, & \\ PI = \emptyset, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$$

The nodes x_i^{Sub}, $1 \leq i \leq n$ are modified as follows:

Case 1. If $\alpha(x_i) = l$:

- $\underline{\text{nodes } x_i^{Sub},\ 1 \leq i \leq n}$:
$$\begin{array}{ll} M = \{t_i^{r'} \to t_i^{r'''}\} \cup \{t_i^{r''} \to t_i^{r'''}\}, & \\ PI = \{t_i^{r'}, t_i^{r''}\}, & FI = \{t_i^{l'}\}, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$$

Case 2. If $\alpha(x_i) = r$, the nodes x_i^{Sub} are analogous to the *case 1* with the characters l and r interchanged.

Case 3. If $\alpha(x_i) = *$, the nodes x_i^{Sub}, $1 \leq i \leq n$ are defined in the following way:

- $\underline{\text{nodes } x_i^{Sub},\ 1 \leq i \leq n}$:
$$\begin{array}{ll} M = \{t_i^{r'} \to t_i^{r'''}\} \cup \{t_i^{l'} \to t_i^{l'''}\} \cup & \\ \{t_i^{l''} \to t_i^{l'''}\}, & \\ PI = \{t_i^{l'}, t_i^{r'}, t_i^{l''}\}, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$$

Let w be the input string in Γ. In the input node x_{start}, the character $t_1^{l''}$ is inserted in the left-hand side of the string if $\alpha(x_1) \in \{r, *\}$, or the character $t_1^{r''}$ is inserted at the end of the string provided that $\alpha(x_1) = l$. Next, the string enters $Star$ where no rule can be applied. From $Star$, it can only enter x_1^{Sub} where the character is replaced with $t_1^{l\,'''}$ and $t_1^{r\,'''}$, respectively. The new string returns to $Star$ where $t_1^{l\,'''}$ and $t_1^{r\,'''}$ are changed to t_1^l and t_1^r. Then, the string can only enter x_1^s and the simulation starts. Note that the same evolutionary rules applicable in $x_1 \in \Gamma$ are also possible in x_1^s since the special character $t_1^{l\,''}$ or $t_1^{r\,''}$ is set up in a way that it does not block the computation of nodes with $\alpha = r$ and $\alpha = l$, respectively. Inductively, we may assume that a string of the form $t_i^l w$ or wt_i^r lies in the node $x_i^s \in \Gamma'$ if and only if the string w lies in the node $x_i \in \Gamma$.

Let w be transformed into w' in the node x_i and sent to the connected nodes to x_i in Γ. Then, a string $t_i^l w'$ or a string $w' t_i^r$ is produced in the node x_i^s and sent to the node $Star$. Let us analyze the case of a string $t_i^l w'$. The process is analogous for the other string. In $Star$, the character t_i^l is replaced with the symbol $t_j^{l\,'}$, granted that $\{x_i, x_j\} \in \Gamma$, which ensures the new string can only be accepted by subnetworks j corresponding to nodes x_j connected to x_i in the original network Γ. From here, the process is similar to the one described in the previous proof.

- If $\alpha(x_j) = l$, the string can only enter x_j^{Ins} where the symbol $t_j^{r\,''}$ is attached at the end of it. The new string, $t_j^{l\,'} w' t_j^{r\,''}$, continues through x_j^{Del} where $t_j^{l\,'}$ is removed and x_j^{Sub} where $t_j^{r\,''}$ is replaced with $t_j^{r\,'''}$. Then, $t_j^{r\,'''}$ is switched with t_j^r in $Star$, allowing it to enter the node x_j^s. Since the character t_j^r is at the end of the string, it does not interfere with the application of evolutionary rules at the left of the string;
- If $\alpha(x_j) = r$ or $\alpha(x_j) = *$, the string directly enters x_j^{Sub} and the symbol $t_j^{l\,'}$ is replaced with $t_j^{l\,'''}$. Then, the string enters x_j^s after having $t_j^{l\,'''}$ changed to t_j^l in $Star$. As in the previous construction, the communication step in Γ has been simulated by a constant number of (evolution and communication) steps in Γ', and a new evolutionary step in Γ is going to be simulated. We conclude that the two networks accept the same language.

The explanations above allow us to infer that any step in Γ is simulated by a constant number of steps in Γ'; hence, $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$ holds. $\square$

5. Simulating Any NEP with a Grid NEP

Theorem 3. *Given an arbitrary NEP Γ there exists a grid NEP Γ' such that the following two conditions are satisfied:*

1. $L(\Gamma) = L(\Gamma')$;
2. $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$.

Proof. Let $\Gamma = (V, U, G, \mathcal{N}, \alpha, \beta, x_1, x_n)$ be a NEP with the underlying graph $G = (X_G, E_G)$ and $X_G = \{x_1, x_2, \ldots, x_n\}$ for some $n \geq 1$; $x_1 \equiv \underline{In}$ and $x_n \equiv \underline{Halt}$. We construct the NEP $\Gamma = (V', U', G', \mathcal{N}', \alpha', \beta', x_{start}, x_n^s)$; $x_{start} \equiv \underline{In}$ and $x_n^s \equiv \underline{Halt}$, where

$$V' = V, \qquad U' = U \cup T,$$
$$T = \{t_i^l, t_i^r, t_i^{l\,'}, t_i^{r\,'} \mid 1 \leq i \leq n\}$$

First, we add the following nodes to Γ':

- $\underline{\text{node } x_{start}}$:
$$M = \begin{cases} \{\varepsilon \to t_1^l\}, \text{ if } \alpha(x_1) \neq l \\ \{\varepsilon \to t_1^r\}, \text{ if } \alpha(x_1) = l \end{cases},$$
$$PI = \emptyset, \qquad FI = T,$$
$$PO = \emptyset, \qquad FO = \emptyset,$$
$$\alpha = \begin{cases} l, \text{ if } \alpha(x_1) \neq l \\ r, \text{ if } \alpha(x_1) = l \end{cases}, \qquad \beta = (w).$$

- $\underline{\text{nodes } x_i^s,\ 1 \leq i \leq n}:$ $\begin{array}{ll} M = M(x_i), & \\ PI = PI(x_i), & FI = FI(x_i) \cup T \setminus \{t_i^l, t_i^r\}, \\ PO = PO(x_i), & FO = FO(x_i), \\ \alpha = \alpha(x_i), & \beta = \beta(x_i). \end{array}$

For each node $x_i,\ 1 \leq i \leq n$ in Γ we add a subnetwork to Γ' according to the subsequent cases:

Case 1. If $\alpha(x_i) = l$, the subnetwork is defined as follows:

- $\underline{\text{nodes } x_i^{Ins},\ 1 \leq i \leq n}:$ $\begin{array}{ll} M = \{\varepsilon \to t_i^{r'}\}, & \\ PI = \emptyset, & FI = T, \\ PO = \{t_i^{r'}\}, & FO = \emptyset, \\ \alpha = r, & \beta = (w). \end{array}$

- $\underline{\text{nodes } x_i^{Del},\ 1 \leq i \leq n}:$ $\begin{array}{ll} M = \{t_i^{l'} \to \varepsilon\}, & \\ PI = \{t_i^{l'}\}, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = l, & \beta = (w). \end{array}$

- $\underline{\text{nodes } x_i^{Sub},\ 1 \leq i \leq n}:$ $\begin{array}{ll} M = \{t_i^r \to t_j^{r'} \mid \{x_i, x_j\} \in \Gamma\} \cup \{t_i^{r'} \to t_i^r\} \cup & \\ \{t_i^{r''} \to t_i^r\}, & \\ PI = T, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$

Case 2. If $\alpha(x_i) = r$, the subnetwork is analogous to the *case 1* with the symbols l and r interchanged.

Case 3. If $\alpha(x_i) = *$, the subnetwork is defined as follows:

- $\underline{\text{nodes } x_i^{Sub},\ 1 \leq i \leq n}:$ $\begin{array}{ll} M = \{t_i^r \to t_j^{r'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^l \to t_j^{l'} \mid \{x_i, x_j\} \in \Gamma\} \cup & \\ \{t_i^{l'} \to t_i^l\} \cup \{t_i^{r'} \to t_i^r\}, & \\ PI = T, & FI = \emptyset, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$

Lastly, we add a set of dummy nodes to complete the grid topology with the specifications below:

- $\underline{\text{nodes } D_i,\ 1 \leq i \leq 2n \wedge \alpha(x_i) = *}:$ $\begin{array}{ll} M = \emptyset, & \\ PI = \emptyset, & FI = U', \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$

- $\underline{\text{nodes } D}:$ $\begin{array}{ll} M = \emptyset, & \\ PI = \emptyset, & FI = \{t_i^l, t_i^r \mid 1 \leq i \leq n\}, \\ PO = \emptyset, & FO = \emptyset, \\ \alpha = *, & \beta = (w). \end{array}$

The grid network is set up in the following way.

- The node x_{start} is in the top left corner. The first column is composed by it followed by the node x_1^s corresponding to the input node $x_1 \in \Gamma$ and the remaining nodes x_i^s arranged in any order;
- The second column is composed by a dummy node D and the nodes x_i^{Sub}. Each node x_i^{Sub} is connected to the node x_i^s through the left edge;

- The third column is composed by a dummy node D and the nodes x_i^{Del}. Each node x_i^{Del} is connected to the node x_i^{Sub} through the left edge. In the case of $\alpha = *$, a node D_i is used instead of a node x_i^{Del};
- The fourth column is composed by a dummy node D and the nodes x_i^{Ins}. Each node x_i^{Ins} is connected to the node x_i^{Del} through the left edge. In the case of $\alpha = *$, a node D_i is used instead of a node x_i^{Ins};
- The fifth column is composed by nodes D.

Let w be the input string in Γ. In the input node x_{start}, the character t_1^l is inserted in the beginning of the string if $\alpha(x_1) \in \{r, *\}$, or the character t_1^r is inserted at the end of the string, if $\alpha(x_1 \in \Gamma) = l$. Then, the string can only enter x_1^s and the simulation starts. Note that the same evolutionary rules applicable in $x_1 \in \Gamma$ are also possible in x_1^s since the special character t_1^l or t_1^r is set up in a way that it does not block the computation of nodes with $\alpha = r$ and $\alpha = l$, respectively. Inductively, we may assume that a string of the form $t_i^l w$ or wt_i^r lies in the node $x_i^s \in \Gamma'$ if and only if the string w lies in the node $x_i \in \Gamma$.

Let w be transformed into w' in the node x_i and sent to the connected nodes to x_i in Γ. Then, a string $t_i^l w'$ or a string wt_i^r is produced in the node x_i^s and sent to the connected node x_i^{Sub}. In this node, the symbols t_i^l and t_i^r are replaced with $t_j^{l\prime}$ and $t_j^{r\prime}$, respectively, granted that $\{x_i, x_j\} \in \Gamma$. Then, the string continues through the second column of x_i^{Sub} nodes until it ultimately enters the node x_j^{Sub}. Note that even if the string passes through the other nodes $x_k^{Sub} \mid k \neq j$, no rule can applied so the string remains unchanged until it gets to the desired node. Next, the computation can be continued in one of the following ways:

- If $\alpha(x_j) = l$, no rule can be applied in x_j^{Sub} and the string enters x_j^{Del}. In that node, the symbol $t_j^{l\prime}$ is removed. Next, since it does not contain any character $t \in T$, the string can only enter the node x_j^{Ins} where a character $t_j^{r\prime}$ is attached to the end. Then, the string continues through the fifth column of dummy nodes D and it ultimately returns to x_j^{Sub} where $t_j^{r\prime}$ is replaced with t_j^r, allowing it to enter the node x_j^s;
- If $\alpha(x_j) = r$ or $\alpha(x_j) = *$, the string directly enters x_j^{Sub} and the symbol $t_j^{l\prime}$ is replaced with t_j^l. Then, the word enters x_j^s. As in the previous proofs, we conclude that $L(\Gamma) = L(\Gamma')$, as well as $Time_{\Gamma'}(n) \in \mathcal{O}(Time_{\Gamma}(n))$.

□

6. Conclusions and Further Work

We have proposed three constructions for simulating an arbitrary NEP by a NEP having an underlying structure that is a complete graph, a star graph, and a two-dimensional grid, respectively. All these simulations are time efficient in the sense that every computational step in the given network is simulated by a constant number of computational steps in the constructed network.

In our view, it would be of interest whether or not similar results are valid for other variants of NEPs, such as polarized NEPs or NEPs with filtered connections as well as for variants of networks of splicing processors.

Author Contributions: Conceptualization, V.M.; methodology, V.M. and J.Á.S.M.; validation, V.M. and J.Á.S.M.; formal analysis, J.Á.S.M.; investigation, V.M. and J.Á.S.M.; writing—original draft preparation, J.Á.S.M.; writing—review and editing, V.M. and J.Á.S.M.; supervision, V.M.; funding acquisition, V.M. Both authors have read and agreed to the published version of the manuscript.

Funding: This work was partially 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.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Castellanos, J.; Martín-Vide, C.; Mitrana, V.; Sempere, J.M. Networks of evolutionary processors. *Acta Inform.* **2003**, *39*, 517–529. [CrossRef]
2. Csuhaj-Varjú, E.; Martín-Vide, C.; Mitrana, V. Hybrid networks of evolutionary processors are computationally complete. *Acta Inform.* **2005**, *41*, 257–272. [CrossRef]
3. 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.* **2010**, *46*, 174–192. [CrossRef]
4. 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.* **2007**, *103*, 112–118. [CrossRef]
5. 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.* **2011**, *10*, 429–445. [CrossRef]
6. Alarcón, P.; Arroyo, F.; Mitrana, V. Networks of polarized evolutionary processors. *Inform. Sci.* **2014**, *265*, 189–197. [CrossRef]
7. Drăgoi, C.; Manea, F.; Mitrana, V. Accepting networks of evolutionary processors with filtered connections. *J. Univ. Comput. Sci.* **2007**, *13*, 1598–1614.
8. Navarrete, C.; Cruz, M.; Rey, E.; Ortega, A.; Rojas, J. Parallel simulation of NEPs on clusters. In Proceedings of the International Conferences on Web Intelligence and Intelligent Agent Technology, Lyon, France, 22–27 August 2011; Volume 3, pp. 171–174.
9. Gómez, S.; Ortega, A.; Orgaz, P. Distributed simulation of NEPs based nn-demand cloud elastic computation. *Adv. Comput. Intell.* **2015**, *9094*, 40–54.
10. Gómez, S.; Ordozgoiti, B.; Mozo, A. NPEPE: Massive natural computing engine for optimally solving NP-complete problems in Big Data scenarios. *New Trends Databases Inf. Syst.* **2015**, *539*, 207–217.
11. Gómez, S.; Mitrana, V.; Păun, M.; Vararuk, S. High performance and scalable simulations of a bio-inspired computational model. In Proceedings of the International Conference on High Performance Computing & Simulation, Dublin, Ireland, 15–19 July 2019; pp. 543–550.
12. Rozenberg, G.; Salomaa, A. (Eds.) *Handbook of Formal Languages*; Springer: Berlin, Germany, 1998.
13. Manea, F.; Martín-Vide, C.; Mitrana, V. On the size complexity of universal accepting hybrid networks of evolutionary processors. *Math. Struct. Comput. Sci.* **2007**, *17*, 753–771. [CrossRef]
14. Loos, R.; Manea, F.; Mitrana, V. Small universal accepting hybrid networks of evolutionary processors. *Acta Inform.* **2010**, *47*, 133–146. [CrossRef]