Acta Informatica 39, 517–529 (2003)
Digital Object Identifier (DOI) 10.1007/s00236-003-0114-y

© Springer-Verlag 2003

Networks of evolutionary processors

Juan Castellanos[1], **Carlos Martín-Vide**[2], **Victor Mitrana**[3,⋆], **José M. Sempere**[4]

[1] Department of Artificial Intelligence, Polytechnical University of Madrid, 28660 Boadilla del Monte, Madrid, Spain (e-mail: jcastellanos@fi.upm.es)
[2] Research Group in Mathematical Linguistics, Rovira i Virgili University, Pça. Imperial Tàrraco 1, 43005 Tarragona, Spain (e-mail: cmv@correu.urv.es)
[3] Faculty of Mathematics, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania (e-mail: mitrana@funinf.math.unibuc.ro)
[4] Department of Information Systems and Computation, Polytechnical University of Valencia, Valencia 46071, Spain (e-mail: jsempere@dsic.upv.es)

Received: 26 September 2002 / 22 January 2003

Abstract. In this paper we consider networks of evolutionary processors as language generating and computational devices. When the filters are regular languages one gets the computational power of Turing machines with networks of size at most six, depending on the underlying graph. When the filters are defined by random context conditions, we obtain an incomparability result with the families of regular and context-free languages. Despite their simplicity, we show how the latter networks might be used for solving an NP-complete problem, namely the "3-colorability problem", in linear time and linear resources (nodes, symbols, rules).

1 Introduction

A basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [7] as well as the Logic Flow paradigm [5], consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules. Afterwards, local data becomes a mobile agent which can navigate in the network following a given protocol. Only such data can be communicated which can pass a filtering process. This filtering

⋆ Work supported by the Generalitat de Catalunya, Direcció General de Recerca (PIV2001-50).

process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g., [6,7].

Starting from the premise that data can be given in the form of strings, [3] introduces a concept called network of parallel language processors in the aim of investigating this concept in terms of formal grammars and languages. Networks of language processors are closely related to grammar systems, more specifically to parallel communicating grammar systems [2]. The main idea is that one can place a language generating device (grammar, Lindenmayer system, etc.) in any node of an underlying graph which rewrite the strings existing in the node, then the strings are communicated to the other nodes. Strings can be successfully communicated if they pass some output and input filters.

In [1], we modify this concept in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a 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 its 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 node is organized in the form of multisets of strings, each copy being processed in parallel such that all the possible evolution events that can take place do actually take place. These networks may be used as language (macroset) generating devices or as computational ones like in [1]. Here we shall consider these networks as language generating devices and study their generative power. Moreover, we discuss how very simple variants of these networks might be used for solving another NP-complete problem. The variants we use here for solving the "3-colorability problem" are simpler than those used in [1] for solving the bounded Post Correspondence Problem. It is worth mentioning here the similarity, in a certain sense, of this model to that of a P system, a new computing model inspired by the hierarchical and modularized cell structure recently proposed in [11]. However, the model discussed here has several proper features like the underlying structure which is a graph, the string communication mode, the derivation mode, the node filters, etc.

2 Preliminaries

The set of all strings over V is denoted by V^* and the empty string is denoted by ε. A *multiset* over a set X is a mapping $M : X \longrightarrow \mathbf{N} \cup \{\infty\}$.

The number $M(x)$ expresses the number of copies of $x \in X$ in the multiset M. When $M(x) = \infty$, then x appears arbitrarily many times in M. The set $supp(M)$ is the support of M, i.e., $supp(M) = \{x \in X \mid M(x) > 0\}$.

A *network of evolutionary processors* (NEP for short) of size n is a construct

$$\Gamma = (V, N_1, N_2, \ldots, N_n, G),$$

where V is an alphabet and for each $1 \leq i \leq n$, $N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

• M_i is a finite set of evolution rules of one of the following forms only

– $a \to b$, $a, b \in V$ (substitution rules),
– $a \to \varepsilon$, $a \in V$ (deletion rules),
– $\varepsilon \to a$, $a \in V$ (insertion rules),

More clearly, the set of evolution rules of any processor contains either substitution or deletion or insertion rules.

• A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i-th node. Actually, in what follows, we consider that each string appearing in any node at any step has an arbitrarily large number of copies in that node, so that we shall identify multisets by their supports.

• PI_i and PO_i are subsets of V^* representing the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i$ ($w \in PO_i$).

Finally, $G = (\{N_1, N_2, \ldots, N_n\}, E)$ is an undirected graph called the *underlying graph* of the network. The edges of G, that is the elements of E, are given in the form of sets of two nodes. The complete graph with n vertices is denoted by K_n.

By a configuration (state) of an NEP as above we mean an n-tuple $C = (L_1, L_2, \ldots, L_n)$, with $L_i \subseteq V^*$ for all $1 \leq i \leq n$. A configuration represents the sets of strings (remember that each string appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$. A configuration can change either by an evolutionary step or by a communicating step. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i.

Formally, we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes into the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by an evolutionary step, written as $C_1 \Longrightarrow C_2$ if L'_i is the set of strings obtained by applying the rules of R_i to the strings in L_i as follows:

(i) If the same substitution or deletion rule may replace different occurrences of the same symbol within a string, all these occurrences must be replaced within different copies of that string. The result is a multiset in which every string that can be obtained appears in an arbitrarily large number of copies.

(ii) An insertion rule is applied at any position in a string. Again, the result is a multiset in which every string, that can be obtained by application of an insertion rule to an arbitrary position in an existing string, appears in an arbitrarily large number of copies.

(iii) If more than one rule, no matter its type, applies to a string, all of them must be used for different copies of that string.

In other words, since an arbitrarily large number of copies of each string is available in every node, after an evolutionary step in each node one gets an arbitrarily large number of copies of any string which can be obtained by using any rule in the set of evolution rules associated with that node. By definition, if L_i is empty for some $1 \leq i \leq n$, then L'_i is empty as well.

Note the main difference in the way of applying the deletion and insertion rules in these variants in comparison with the variants considered in [1]: in the variants from [1] these rules are applied in the ends of the strings only, whereas here they can be applied to any position in the strings. We would like to mention that these extremely simple operations might be interpreted as point mutations. Even the nondeterministic way of applying them, as well as the lack of lexical contexts, suggest this parallelism to local DNA mutations.

When changing by a communication step, each node processor N_i sends all copies of the strings it has which are able to pass its output filter to all the node processors connected to N_i and receives all copies of the strings sent by any node processor connected with N_i providing that they can pass its input filter.

Formally, we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes into the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by a communication step, written as $C_1 \vdash C_2$ if

$$L'_i = L_i \setminus \{w \mid w \in L_i \cap PO_i\} \cup \bigcup_{\{N_i,N_j\}\in E} \{x \mid x \in L_j \cap PO_j \cap PI_i\}$$

for every $1 \leq i \leq n$.

Let $\Gamma = (V, N_1, N_2, \ldots, N_n)$ be an NEP. By a computation in Γ we mean a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$.

If the sequence is finite, we have a finite computation. The result of any finite or infinite computation is a language which is collected in a designated node called the output (master) node of the network. If one considers the

output node of the network as being the node k, and if $C_0, C_1, \ldots$ is a computation, then all strings existing in the node k at some step t - the k-th component of C_t - belong to the language generated by the network. Let us denote this language by $L_k(\Gamma)$.

The time complexity of computing a finite set of strings Z is the minimal number of steps t in a computation $C_0, C_1, \ldots, C_t \ldots$ such that Z is a subset of the k-th component of C_t.

In the theory of networks some types of underlying graphs are common, e.g., rings, stars, grids, etc. We shall investigate here networks of evolutionary processors with their underlying graphs having these special forms. Thus a NEP is said to be a *star, ring, grid, or complete* NEP if its underlying graph is a star, ring, grid, or complete graph, respectively.

3 Computational completeness

Since NEPs with finite filters can generate regular languages only, we shall consider in the sequel NEPs having infinite regular languages as filters.

Theorem 1 *Each recursively enumerable language can be generated by a complete NEP of size 5.*

Proof. Let $G = (N, T, S, P)$ be an arbitrary phrase-structure grammar in the Kuroda normal form, namely P contains only rules of the following forms:

$$A \to a,\ A \to BC,\ AB \to CD,\ A \to \varepsilon,$$

where A, B, C, D are nonterminals and a is a terminal. We assume that the rules $A \to BC$ and $AB \to CD$ of P are labelled in a one-to-one manner by the labels $r_1, r_2, \ldots, r_n$. We shall refer to the rules of the form $A \to a$, $A \to \varepsilon$, $A \to BC$, and $AB \to CD$ as rules of type 0, 1, 2, and 3, respectively. We construct the following NEP of size 5 having a complete underlying graph:

$$\Gamma = (N \cup T \cup V \cup \{X\}, N_0, N_1, N_2, N_3, N_4, K_5)$$

where $V = \{r_i, p_i, q_i, s_i, t_i \mid 1 \le i \le n\}$ and

$$N_0 = (\emptyset, \emptyset, T^*, (N \cup T \cup V \cup \{X\})^*(N \cup V \cup \{X\})(N \cup T \cup V \cup \{X\})^*)$$

$$N_1 = (M_1, \{S\}, (N \cup T)^* \cup (N \cup T)^*\{r_i q_i \mid 1 \le i \le n\}(N \cup T)^*, (N \cup T)^*(\{r_i, s_i p_i, t_i q_i \mid 1 \le i \le n\} \cup \{X\})(N \cup T)^* \cup T^*)$$

with

$$
\begin{aligned}
M_1 &= \{A \to X \mid A \to \varepsilon \in P\} \cup \{A \to a \mid A \to a \in P\} \cup \{A \to r_i, \\
&\quad r_i \to t_i \mid r_i : A \to BC \in P\} \cup \{A \to s_i, B \to p_i \mid r_i : AB \to CD \in P\} \\
N_2 &= (\{\varepsilon \to q_i \mid r_i : A \to BC\}, \emptyset, (N \cup T)^* \{r_i \mid 1 \le i \le n\} (N \cup T)^*, \\
&\quad (N \cup T \cup V)^*) \\
N_3 &= (M_3, \emptyset, (N \cup T)^* \{s_i p_i, t_i q_i \mid 1 \le i \le n\} (N \cup T)^*, (N \cup T)^*) \\
&\quad \text{with} \\
M_3 &= \{t_i \to B, q_i \to C \mid r_i : A \to BC \in P\} \\
&\quad \cup \{s_i \to C, p_i \to D \mid r_i : AB \to CD \in P\}, \\
N_4 &= (\{X \to \varepsilon\}, \emptyset, (N \cup T)^* \{X\} (N \cup T)^*, (N \cup T)^*).
\end{aligned}
$$

Here are some explanations on the working mode of this network. Initially, there are arbitrarily many copies of the string S in the node N_1. By extrapolation, we may assume that a multiset of strings containing at least one nonterminal and no symbol from $V \cup \{X\}$, each string appearing in an unbounded number of copies, is in N_1 at a given moment, before an evolutionary step. If there is one string x in this multiset having at least one occurrence of a nonterminal A and $A \to Y$ is an evolution rule in M_1, then the first occurrence of A in x is replaced by Y in an infinite number of copies of x, the second occurrence of A in x is replaced by Y in an infinite number of copies of x, and so on for all the occurrences of A in x. This process applies to all strings in N_1 and to all evolution (substitution) rules in M_1.

Those strings obtained by an application of a rule $A \to Y, Y \in T$ cannot leave N_1 in the next step, which is a communication step, since they cannot pass its output filter. In this way, the network simulates all the possible one step applications of the rules of type 0 in two steps.

Those strings obtained by an application of a rule $A \to r_i$, for some i, (this means that there is a rule $r_i : A \to BC$ in P) are sent out and received by either N_2 or N_4 which are the only nodes able to receive them. More precisely, those strings containing X are received by N_4 and the others by N_2. In N_4, the symbol X is removed and the obtained strings are sent back to either N_1, provided that they still contain nonterminals, or N_0, if they are terminal strings. Thus, all possible one step applications of the rules of type 1 are done in four steps. In N_2, q_j is inserted to any position in the existing strings in the same way as that discussed above for the rule $A \to Y$, for all j such that r_j is a rule of type 2. Only those strings having an occurrence of r_i which received an adjacent symbol q_i in the righthand side of r_i can leave N_2, all the others remaining in N_2 forever. The strings leaving N_2 cannot be received by any node other than N_1 where the only useful evolution rules which can be applied to them are those of the form $r_i \to t_i$. After applying these rules, the new strings are sent out again. All the other rules applied to the strings just received by N_1 lead to strings obliged to remain forever in

N_0. The strings sent out are received by N_3 where t_i and q_i are replaced by B and C, respectively, provided that the righthand side of the rule r_i is BC. In this way, the network simulates all the possible one step applications of the rules of type 2 in ten steps.

In a similar way, the network simulates all the possible one step applications of the rules of type 3 in eight steps. More precisely, if one wants to apply the rule $r_i : AB \to CD$, then in N_1, in an evolutionary step, an occurrence of A is replaced by s_i, these new strings remain in N_1 during the next communication step since they cannot pass the output filter of N_1, then in the next evolutionary step an occurrence of B is replaced by p_i, but only those strings containing the subword s_ip_i (this means that a subword AB was replaced by s_ip_i) can pass the output filter of N_1, the others remaining in N_1 forever.

As one can see, the strings over the alphabet $N \cup T$ always return to N_1, where the aforementioned process resumes for all of them still containing nonterminals while the terminal strings are sent to N_0 where they remain forever.

By these explanations, we infer that the language generated by Γ in the output node N_0 is exactly $L(G)$. □

It is worth mentioning here that, unlike other parallel language generating devices, a NEP generates a language in a very efficient way, namely all strings that can be generated by a grammar, each of them in n steps, are generated altogether by a NEP in at most $10n$ steps.

If one looks deeper into the proof above, one may see that it can be easily modified for a star NEP having N_1 as the central node. Therefore, we may state:

Theorem 2 *Each recursively enumerable language can be generated by a star NEP of size 5.*

A similar situation holds for ring NEPs, namely

Theorem 3 *Each recursively enumerable language can be generated by a ring NEP of size 6.*

Proof. The construction is rather similar to that from the proof of Theorem 1. For the same grammar from the previous proof we consider the NEP of size 6:

$$\Gamma = (N \cup T \cup U \cup \{X\}, N_0, N_1, N_2, N_3, N_4, N_5, G)$$

where $U = \{r_i, p_i, q_i, s_i \mid 1 \neq i \leq n\}$ and

$$N_0 = (\emptyset, \emptyset, T^*, (N \cup T \cup U)^*(N \cup U)$$

$$(N \cup T \cup U)^*),$$
$$N_1 = (M_1, \{S\}, (N \cup T)^*, (N \cup T)^*\{r_i, s_i p_i | 1 \leq i \leq n\}(N \cup T)^* \cup T^*),$$
with
$$M_1 = \{A \to X \mid A \to \varepsilon \in P\} \cup \{A \to a \mid A \to a \in P\} \cup$$
$$\{A \to r_i | r_i : A \to BC \in P\} \cup \{A \to s_i, B \to p_i | r_i : AB \to CD \in P\}$$

$$N_2 = (\{\varepsilon \to q_i | r_i : A \to BC\}, \emptyset, (N \cup T)^*\{r_i, s_i p_i | 1 \leq i \leq n\}(N \cup T)^*,$$
$$(N \cup T)^*\{s_i p_i, r_i q_i \mid 1 \leq i \leq n\}(N \cup T)^*)$$
$$N_3 = (M_3, \emptyset, (N \cup T)^*\{s_i p_i, r_i q_i \mid 1 \leq i \leq n\}(N \cup T)^*,$$
$$(N \cup T)^*\{r_i q_i \mid 1 \leq i \leq n\}(N \cup T)^*),$$
with
$$M_3 = \{s_i \to C, p_i \to D \mid r_i : AB \to CD \in P\}$$
$$N_4 = (M_4, \emptyset, (N \cup T)^*\{r_i q_i \mid 1 \leq i \leq n\}(N \cup T)^*, (N \cup T)^*),$$
with
$$M_4 = \{r_i \to B, q_i \to C \mid r_i : A \to BC \in P\},$$
$$N_5 = (\{X \to \varepsilon\}, \emptyset, (N \cup T)^*(\{X\} \cup \{r_i, s_i p_i \mid 1 \leq i \leq n\})(N \cup T)^*,$$
$$(N \cup T \cup U)^*).$$

The underlying graph is

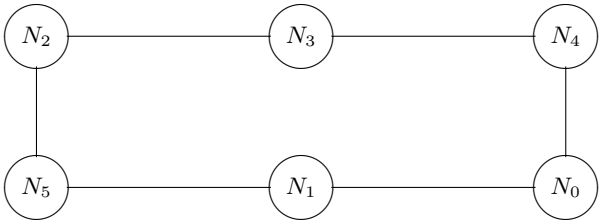

Fig. 1. The ring underlying graph of Γ

By similar considerations to those from the previous proof we conclude that $L(G) = L_0(\Gamma)$. □

It is easy to note that in all cases above, the node N_0 is used as a squeezing node; its unique role is to collect the terminal strings. If one removes it, one obtains smaller NEPs (of size 4 and 5, respectively) which we do not know whether or not can generate all recursively enumerable languages (this is left as an *open problem*) but they can generate non-recursive languages, besides all context-sensitive languages (note that the node N_4 in the proof

of Theorem 1 and N_5 in the proof of Theorem 3 are used for simulating the erasing rules of the given grammar only).

4 Simple NEPs

A *simple NEP* of size n is a construct $\Gamma = (V, N_1, N_2, \ldots, N_n, G)$, where, V and G have the same interpretation as for NEPs, and for each $1 \leq i \leq n$, $N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i)$ is the i-th evolutionary node processor of the network. M_i and A_i from above have the same interpretation as for an evolutionary node in a NEP, but

• PI_i and FI_i are subsets of V representing the input filter. This filter, as well as the output filter, is defined by random context conditions, PI_i forms the permitting context condition and FI_i forms the forbidding context condition. A string $w \in V^*$ can pass the input filter of the node processor i, if w contains each element of PI_i but no element of FI_i. Note that any of the random context conditions may be empty, in this case the corresponding context check is omitted. We write $\rho_i(w) = \underline{true}$, if w can pass the input filter of the node processor i and $\rho_i(w) = \underline{false}$, otherwise.

• PO_i and FO_i are subsets of V representing the output filter. Analogously, a string can pass the output filter of a node processor if it satisfies the random context conditions associated with that node. Similarly, we write $\tau_i(w) = \underline{true}$, if w can pass the input filter of the node processor i and $\tau_i(w) = \underline{false}$, otherwise.

The working mode of such a simple NEP differs only in the communication step, namely we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes into the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by a communication step, written as $C_1 \vdash C_2$ if for every $1 \leq i \leq n$,

$$L'_i = L_i \setminus \{w \in L_i \mid \tau_i(w) = \underline{true}\} \cup \bigcup_{\{N_i, N_j\} \in E} \{x \in L_j \mid (\tau_j(x) \wedge \rho_i(x)) = \underline{true}\}.$$

As far as the computational power of the simple NEPs is concerned we do not have a complete answer. However, these devices can generate non-context-free languages.

Theorem 4 *The families of regular and context-free languages are incomparable with the family of languages generated by simple NEPs.*

Proof. The number of all occurrences of a letter a in a string x is denoted by $|x|_a$. First we show how the language $L = \{x \in \{a, b, c\}^* \mid |x|_a = |x|_b = |x|_c \geq 1\}$ can be generated by a simple complete NEP of size 6. To this aim,

we take the language $V = \{X, a, b, c, a', b'\}$ and construct the simple NEP $\Gamma = (V, N_0, N_1, N_2, N_3, N_4, N_5)$, where

$$N_0 = (\{\varepsilon \to a', \varepsilon \to X\}, \{\varepsilon\}, \emptyset, \{a, b, c, X\}, \{X, a'\}, \emptyset),$$
$$N_1 = (\{\varepsilon \to b'\}, \emptyset, \{a'\}, \emptyset, \emptyset, \emptyset), \quad N_2 = (\{a' \to a\}, \emptyset, \{a', b'\}, \emptyset, \emptyset, \emptyset),$$
$$N_3 = (\{\varepsilon \to c\}, \emptyset, \{b'\}, \{a'\}, \emptyset, \emptyset), \quad N_4 = (\{b' \to b\}, \emptyset, \{b', c\}, \emptyset, \emptyset, \emptyset),$$
$$N_5 = (\{X \to \varepsilon\}, \emptyset, \{a, b, c\}, \{b'\}, \emptyset, \{a, b, c\}).$$

A few words about the way in which the strings are generated in this NEP. The computation starts in N_0 where after $n + 1$ evolutionary steps all strings containing n occurrences of a' and one occurrence of X are sent out. Only N_1 is entitled to receive them; in N_1 the symbol b' is inserted to any possible position resulting in strings which contain n occurrences of a' and one occurrence of X and b', respectively. After a communicating step, all these strings arrive in N_2, where an occurrence of a' (no matter which of them) is replaced by a. Now, the strings are returned to N_1 and the process is resumed. This "ping-pong" process continues until all occurrences of a' were transformed into a. During this process the same number of occurrences, that is n, of b' were inserted such that at the end of this process, only those strings having n occurrences of a and the same number of occurrences of b', together with one occurrence of X are in N_2.

The aforementioned process applies again for the occurrences of b' and c by means of the nodes N_3 and N_4. When all strings having the same number, n, of occurrences of each letter a, b, and c, and still one occurrence of X, were produced in N_4, they are collected in N_5, where X is removed and the resulting strings will remain here forever. Hence $L = L_5(\Gamma)$.

On the other hand, it is obvious that the regular language $\{a^n b^m \mid n, m \geq 1\}$ cannot be generated by any simple NEP. □

We shall discuss now how these simple NEPs, despite their simplicity, might be used for solving a well known NP-complete problem, namely the "3-colorability problem". This problem is to decide whether each vertex in an undirected graph can be colored by using three colors (say red, blue, and green) in such a way that after coloring, no two vertices which are connected by an edge have the same color.

Theorem 5 *The "3-colorability problem" can be solved in $O(m + n)$ time by a complete simple NEP of size $7m + 2$, where n is the number of vertices and m is the number of edges of the input graph.*

Proof. Let $G = (\{1, 2, \ldots, n\}, \{e_1, e_2, \ldots, e_m\})$ a graph and assume that $e_t = \{i_t, j_t\}$, $1 \leq i_t < j_t \leq n$, $1 \leq t \leq m$. We consider the alphabet $U = V \cup V' \cup T \cup \{X_1, X_2, \ldots, X_{m+1}\}$, where $V = \{b_1, r_1, g_1, \ldots, b_n, r_n, g_n\}$, $T = \{a_1, a_2, \ldots, a_n\}$. Here V' is the primed copy of V, that is the set formed

by the primed copies of all letters in V. We construct the complete simple NEP of size $7m + 2$ having the nodes:

$$N_0 = (\{a_i \to b_i, a_i \to r_i, a_i \to g_i \mid 1 \le i \le n\}, \{a_1 a_2 \dots a_n X_1\}, \\ T \cup \{X_1\}, \emptyset, \emptyset, T).$$

The further nodes are:

$$\begin{aligned}
N_{e_t}^{(Z)} &= (\{Z_{i_t} \to Z'_{i_t}\}, \emptyset, \{X_t\}, U \setminus V, \{Z'_{i_t}\}, \emptyset), Z \in \{b, r, g\}, \\
\bar{N}_{e_t}^{(b)} &= (\{r_{j_t} \to r'_{j_t}, g_{j_t} \to g'_{j_t}\}, \emptyset, \{X_t, b'_{i_t}\}, \emptyset, \{r'_{j_t}, g'_{j_t}\}, \emptyset), \\
\bar{N}_{e_t}^{(r)} &= (\{b_{j_t} \to b'_{j_t}, g_{j_t} \to g'_{j_t}\}, \emptyset, \{X_t, r'_{i_t}\}, \emptyset, \{b'_{j_t}, g'_{j_t}\}, \emptyset), \\
\bar{N}_{e_t}^{(g)} &= (\{r_{j_t} \to r'_{j_t}, b_{j_t} \to b'_{j_t}\}, \emptyset, \{X_t, g'_{i_t}\}, \emptyset, \{r'_{j_t}, b'_{j_t}\}, \emptyset), \\
N_{e_t} &= (\{r'_{i_t} \to r_{i_t}, b'_{i_t} \to b_{i_t}, g'_{i_t} \to g_{i_t}, r'_{j_t} \to r_{j_t}, b'_{j_t} \to b_{j_t}, g'_{j_t} \to g_{j_t}, \\
&\quad X_t \to X_{t+1}\}, \emptyset, \{X_t\}, \{r_{i_t}, b_{i_t}, g_{i_t}, r_{j_t}, b_{j_t}, g_{j_t}\}, \{X_{t+1}\}, U \setminus V),
\end{aligned}$$

for all $1 \le t \le m$, and $N_1 = (\{X_{m+1} \to \varepsilon\}, \emptyset, \{X_{m+1}\}, U \setminus V, \emptyset, V)$.

For the first $2n$ steps, out of which n steps are communication ones when nothing is actually communicated, the strings will remain in N_0 until no letter in T appears in them anymore. When this process is finished, the obtained strings encode all possible ways of coloring the vertices, satisfying or not the requirements of the problem.

Now, for each edge e_t, our NEP keeps only those strings which encodes a colorability satisfying the condition for the two vertices of e_t. This is done by means of the nodes $N_{e_t}^{(b)}$, $N_{e_t}^{(r)}$, $N_{e_t}^{(g)}$, $\bar{N}_{e_t}^{(b)}$, $\bar{N}_{e_t}^{(r)}$, $\bar{N}_{e_t}^{(g)}$, and finally N_{e_t} in 8 steps. It is clear that the given instance has a solution if the language of the node N_1 is non-empty. As one can see, the overall time of a computation is $8m + 2n$.

We finish the proof by mentioning that the total number of rules is $16m + 3n + 1$. In conclusion, all parameters of the network are of $O(m + n)$ size. □

It is rather interesting that the underlying graph of the NEP above does not depend on the number of nodes of the given instance of the problem. In other words, the same underlying structure may be used for solving any instance of the 3-colorability problem having the same number of edges but no matter the number of nodes.

The construction above can be modified in the aim of starting from an infinite number of copies of the empty string only. To this end, we need an alphabet T' with n extra symbols $a'_1, a'_2, \dots, a'_n$, and $n + 1$ extra nodes

$$\begin{aligned}
N'_0 &= (\{\varepsilon \to a'_i \mid 1 \le i \le n\}, \{X_1\}, \emptyset, U \setminus T', T' \cup \{X_1\}, \emptyset), \\
N'_i &= (\{a'_i \to a_i\}, \emptyset, T_i \cup \{X_1\}, T' \setminus T_i, \emptyset, \emptyset),
\end{aligned}$$

for all $1 \leq i \leq n$. For each such i, we denoted by $T_i = \{a_1, a_2, \ldots, a_{i-1}, a'_i, \ldots, a'_n\}$.

Now, the computation starts by forming strings containing all letters from T' and one occurrence of X_1 in the node N'_0. This takes $2n$ steps. All these strings are sent out but N'_1 is the only node which can receive them. In the next $2n$ steps those strings which initially contained exactly one occurrence of each symbol in T' will be received, in turn, by the nodes $N'_1, N'_2, \ldots, N'_n$, and finally N_0. From now on, the computation follows the above description.

5 Conclusions

We have considered a mechanism inspired from cell biology, namely networks of evolutionary processors, that is networks whose nodes are very simple processors able to perform just one type of point mutation (insertion, deletion or substitution of a symbol). These nodes are endowed with a filter which is defined by some membership or random context condition. Networks with at most six nodes having filters defined by the membership to a regular language condition are able to generate all recursively enumerable languages no matter the underlying structure. This result does not surprise since similar characterizations have been reported in the literature, see, e.g., [4, 8–10]. Then we considered networks with nodes having filters defined by random context conditions which seem to be closer to the possibilities of biological implementation. Even in this case, rather complex languages like non-context-free ones, can be generated. A more exact characterization of the generative power of these devices remains to be done.

However, these very simple mechanisms are able to solve hard problems in polynomial time. We presented a linear solution for an NP-complete problem, namely the "3-colorability problem". What other complicated problems could be solved in this framework is another point of interest which might be further investigated.

A few remarks about the term of "evolutionary processor" used in this paper. Definitely, the computational process described here is not exactly an evolutionary process in the Darwinian sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process 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 into considerations local mutations only [12]. Furthermore, we were not concerned here with a possible biological implementation, though a matter of great importance.

Acknowledgements. The authors express their thanks to Mario J. Pérez-Jiménez and Fernando J. Martin Mateos for their valuable suggestions on an earlier version of this paper.

References

1. Castellanos, J., Martin-Vide, C., Mitrana, V., Sempere, J.: Solving NP-complete problems with networks of evolutionary processors. Proceedings of IWANN 2001, LNCS 2084, Springer-Verlag, 2001, 621–628
2. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Paun, G.: Grammar Systems. Gordon and Breach, 1993
3. Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In New Trends in Formal Languages, LNCS 1218, Springer Verlag, 1997, 299–318
4. Csuhaj-Varjú, E., Mitrana, V.: Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* 36(2000), 913–926
5. Errico, L., Jesshope, C.: Towards a new architecture for symbolic processing. In Artificial Intelligence and Information-Control Systems of Robots '94, World Sci. Publ., Singapore, 1994, 31–40
6. Fahlman, S.E., Hinton, G.E., Seijnowski, T.J.: Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In Proc. AAAI National Conf. on AI, William Kaufman, Los Altos, 1983, 109–113
7. Hillis, W.D.: The Connection Machine, MIT Press, Cambridge, 1985
8. Kari, L., Păun, Gh., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. *Proc. 3rd DIMACS Workshop on DNA Based Computing*, Philadelphia, 1997, 318–333
9. Kari, L., Thierrin, G.: Contextual insertion/deletion and computability. *Information and Computation* 131, 1(1996), 47–61
10. Martin-Vide, C., Păun, Gh., Salomaa, A.: Characetrizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science* 205, 1-2(1998), 195–205
11. Păun, G.: Computing with membranes, J. Comput. Syst. Sci. **61**(2000) 108–143
12. Sankoff, D., et al.: Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, **89**(1992) 6575–6579