On the Number of Nodes in Universal Networks of Evolutionary Processors

Artiom Alhazov[1,2], Carlos Martín-Vide[3], Yurii Rogozhin[2]

[1]Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

[2]Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
E-mail: {artiom,rogozhin}@math.md
[3]Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
E-mail: cmv@correu.urs.es

Abstract

The purpose of this article is to improve the existing universality results for the networks of evolutionary processors as devices that are computing or generating languages to four nodes for both the ring and complete graphs (with or without loops) when the filters are regular languages. Three nodes are already enough to generate any recursively enumerable language modulo the terminal alphabet.

If operations of different kinds (symbol insertion, symbol substitution and symbol deletion) are allowed in the same node, then two nodes are already enough for universality, while one node generates any recursively enumerable language modulo the terminal alphabet.

1 Introduction

The networks of evolutionary processors (NEPs for short) are a (biologically inspired) distributed word rewriting model introduced recently (see [1, 2]). The inspiration comes from looking at a network of biological cells performing the point operations on DNA and communicating according to the input/output filters.

A NEP consists of a graph of nodes having operations and filters. The simplest operations considered, such as insertion, deletion and substitution of one symbol. More powerful variants are not very interesting from the computational viewpoint: substitution of up to two symbols by up to two symbols are already universal (type-0 grammars) (see, for example, [4]), while insertion/deletion systems are universal with weights 2/3 and 3/2 (see [3]).

Since there is no interaction between different strings, one could view a deterministic system parallel at the level of different strings as a non-deterministic distributed sequential rewriting of one string. Let us point out that the output does not depend on whether the system is synchronized.

Finally, the complete graph (with loops) is the one that best corresponds to the idea of cells communicating with the environment only (instead of communicating via channels designed in a specific way), allowing the output to be interpreted as a result of action of independent agents (emergence).

2 Preliminaries

Definition 1 *A NEP of size n is a tuple $\Gamma = (V, N_1, \cdots, N_n, G)$, where V is the alphabet and $N_i = (M_i, A_i, I_i, O_i)$ is the i-th node, $1 \le i \le n$:*

- *M_i is a finite set of evolutionary rules of the same type ($M_i \in 2^{\{a\to b | a,b\in V\}} \cup 2^{\{a\to\lambda | a\in V\}} \cup 2^{\{\lambda\to b | b\in V\}}$).*
- *A_i is a finite set of strings over V (initial strings).*
- *I_i and O_i are languages over V specifying conditions for a string to enter and to exit a node, respectively (input and output filters).*

Finally, $G = (\{N_1, \cdots, N_n\}, E)$ is an undirected graph specifying the underlying network. Let us denote the complete graph without loops by K_n and the complete graph with loops by K'_n.

The configuration $C = (C[1], \cdots, C[n])$ of the system consists of the sets of strings appearing in each node (each string appears in an arbitrary large number of copies).

- Evolution step $C = (C[1], \cdots, C[n]) \Rightarrow C' = (C'[1], \cdots, C'[n])$:
  $C'[i] = M_i(C[i]) = \cup_{op\in M_i, w\in C[i]} r(w)$, where $r(w)$ is the set of string that can be obtained by one application of rule r to a string w, if r is applicable to w, or $\{w\}$ otherwise.
- Communication step $C = (C[1], \cdots, C[n]) \vdash C' = (C'[1], \cdots, C'[n])$:
  $C'[i] = C[i] \setminus O_i \cup \bigcup_{(N_i,N_j)\in E} C[j] \cap O_j \cap I_i$.

A computation consists of a sequence of configurations C_i, where $C_0 = (A_1, \cdots, A_n)$, $C_{2i} \Rightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$for $i \ge 0$. Notice since an arbitrary number of copies of each string is available in every node, after an evolutionary step in each node one gets an arbitrary 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 $C[i]$ is empty for some $1 \le i \le n$, then $C'[i]$ is empty as well. The result of a (possibly infinite) computation is a language collected in a designated node N_k called the output node. Thus, $L_k(\Gamma) = \cup_{k\ge 0} C_t[k]$.

3 Mixed nodes

In this section only, we consider a variant of NEPs where operations of different kinds are allowed in the same node ($M_i \subseteq \{a \to b \mid a, b \in V\} \cup \{a \to \lambda \mid a \in V\} \cup \{\lambda \to b \mid b \in V\}$) (we denote this variant NEPs as mNEPs). We will present results concerning mNEPs with one and two nodes.

Theorem 1 *Each recursively enumerable language can be generated by a mNEP with one node modulo to terminal alphabet.*

Consider an arbitrary recursively enumerable language L. Then there exists a type-0 grammar $G = (N, T, P, S)$ generating L, and any rule of $p \in P$ can be written as $p : vw \to xy$, where $v, w, x, y \in N \cup T \cup \{\lambda\}$ and symbol $v \in N$.

Construct the following mNEP with one node: $\Gamma_1 = (V, N_1, H)$, where $V = N \cup T \cup P_1 \cup P_2 \cup P_3 \cup P_4$, $P_i = \{p_i \mid p \in P\}$, $1 \le i \le 4$.

Let us suppose $H = K_1$ (graph with one node and no edges) and $N_1 = (M, \{S\}, \emptyset, O)$ with $M = \{v \to p_1, w \to p_2, p_1 \to p_3, p_2 \to p_4, p_3 \to x, p_4 \to y \mid p : vw \to xy \in P,\ v, w, x, y \in N \cup T \cup \{\lambda\}\}$ and $O = V^* \setminus W(\{\lambda\} \cup P_1 \cup P_{1,2} \cup P_{3,2} \cup P_{3,4} \cup P_4)W$, where $W = (N \cup T)^*$ and $P_{i,j} = \{p_i p_j \mid p \in P\}$, $1 \le i, j \le 4$.

Claim: $L_1(\Gamma) \cap T^* = L$. We start with a string in $S \in W$. Every production of G is simulated in six steps (the elements of M corresponding to that production, in that order). All terminal strings of L can thus be obtained at some time in node N_1. Now we only need to prove that $z \in L_1(\Gamma) \cap T^* \Rightarrow z \in L$.

Let us think of symbols p_i as markers, telling us the current step of the simulation. If simulation of some production starts before the previous one is finished, then the string will have markers corresponding to both productions and will hence leave the node.

The "correct" simulation is the following:
$WvwW \Rightarrow^{v \to p_1} Wp_1wW \Rightarrow^{w \to p_2} Wp_1p_2W \Rightarrow^{p_1 \to p_3} Wp_3p_2W \Rightarrow^{p_2 \to p_4} Wp_3p_4W \Rightarrow^{p_3 \to x} Wxp_4W \Rightarrow^{p_4 \to y} WxyW$.

If at some step a different production is applied, or it is applied at a different position, the corresponding string will leave the node through the output filter and be discarded, as is shown by the following table.

	Shape	$v \to p_1$	$w \to p_2$	$p_1 \to p_3$	$p_2 \to p_4$	$p_3 \to x$	$p_4 \to y$
(0)	W	(1)	out	n/a	n/a	n/a	n/a
(1)	WP_1W	out	(2)	out	n/a	n/a	n/a
(2)	$WP_{1,2}W$	out	out	(3)	out	n/a	n/a
(3)	$WP_{3,2}W$	out	out	n/a	(4)	out	n/a
(4)	$WP_{3,4}W$	out	out	n/a	n/a	(5)	out
(5)	WP_4W	out	out	n/a	n/a	n/a	(0)

Here "out" means that the resulting string is discarded (sent out, never brought in), and "n/a" means that the operation is not applicable to the string.

Thus, the only way to obtain a terminal string with all sentential forms not belonging to the output filter is to do a correct simulation of G, so all terminal strings generated by Γ belong to L, q.e.d.

Remark 1 *We obtained the optimal result for mNEPs with one node modulo to terminal alphabet.*

Remark 2 *The construction is also valid for both possible graphs H.*

If the graph has a loop ($H = K_1'$), the system behaves exactly the same because the input filter is empty.

Corollary 1 *Nonrecursive languages can be generated by mNEPs with one node.*

Let us take $L \notin REC$. Since the family of recursive languages is closed under intersection with regular languages, $L_1(\Gamma) \cap T^* = L$ implies $L_1(\Gamma) \notin REC$.

Corollary 2 *mNEPs with two nodes can generate any RE language.*

Take $\Gamma_1 = (V, N_1, H)$ from the construction above. Consider $\Gamma_2 = (V, N_1', N_2, H')$ where $N_2 = (\emptyset, \emptyset, T^*, \emptyset)$ and H' is any graph such that N_1 is connected to N_2 Finally, N_1' is obtained from N_1 by adding T^* to its output filter. The reasoning is the same, except the terminal strings will immediately leave node N_1, enter node N_2 and stay there.

Remark 3 *The result above is optimal for mNEPs with 2 nodes. Indeed, suppose it is not optimal. Then there exists a mNEP with one node that generates the language $(aa)^*$. So as we can insert only one symbol a this system will generate an odd length string too, thus this is a contradiction.*

4 Classical variant

Now let us return to the definition of NEPs where each node is specialized in at most one type of operations. We will present the results concerning NEPs with three and four nodes.

Theorem 2 *Each recursively enumerable language can be generated by a NEP with three nodes modulo to terminal alphabet.*

Consider an arbitrary recursively enumerable language L. Then there exists a grammar $G = (N, T, P, S)$ having rules of types $A \to BC$, $AB \to C$ generating $L \setminus \{\lambda\}$. (Indeed, take a grammar $G' = (N', T, P', S)$ generating L with rules of type $p : u \to v$, where $1 \le |u| \le 2$, $|v| \le 2$. Simulating rules of grammar G for each rule $p : u \to v \in P'$ of grammar G' are as follows:
$|u| = 1$, $|v| = 2$: directly,

$|u| = 1$, $|v| = 1$: $u \to p'p''$, $p'p'' \to v$,
$|u| = 1$, $|v| = 0$: $uA \to A$, $Au \to A$ for all $A \in N' \cup T$,
$|u| = 2$, $|v| = 2$: $u \to p'$, $p' \to v$,
$|u| = 2$, $|v| = 1$: directly,
$|u| = 2$, $|v| = 0$: $u \to p'$, $p'A \to A$, $Ap' \to A$ for all $A \in N' \cup T$, where p, p', p'' are new nonterminals of grammar G.)

Let us denote $W = (N \cup T)^*$, $P_i = \{p_i \mid p \in P, 1 \le i \le 7\}$, $P_{i,j} = \{p_i p_j \mid p \in P, 1 \le i, j \le 7\}$, $P_{i,j,k} = \{p_i p_j p_k \mid p \in P, 1 \le i, j, k \le 7\}$.

Construct the following NEP with three nodes: $\Gamma_2 = (V, N_1, N_2, N_3, H)$, where $V = \bigcup_{1 \le i \le 7} P_i \cup N \cup T$, $H = K_3$, $N_i = (M_i, A_i, I_i, O_i)$, where $1 \le i \le 3$, $A_1 = \{S\} \cup (L \cap \{\lambda\})$, $A_2 = A_3 = \emptyset$. The operations are defined as follows. For each rule of G:

- $p : A \to BC$:
  N_1 has rules $\lambda \to p_1$, $\lambda \to p_2$;
  N_2 has rules $A \to p_3$, $p_1 \to B$, $p_2 \to C$;
  N_3 has rules $p_3 \to \lambda$.
- $p : AB \to C$:
  N_1 has rules $\lambda \to p_5$;
  N_2 has rules $A \to p_4$, $B \to p_6$, $p_6 \to p_7$, $p_4 \to C$;
  N_3 has rules $p_5 \to \lambda$, $p_7 \to \lambda$.

The input filters are:

$$\begin{aligned} I_1 &= W, \\ I_2 &= W(P_{1,2} \cup P_5)W, \\ I_3 &= W(P_3 \cup P_{5,7})W. \end{aligned}$$

The output filters are defined in order to remove the garbage and communicate the strings that should change the type of operation, keep only the strings that should continue to evolve by operations of the same type:

$$\begin{aligned} O_1 &= V^* \setminus W P_1 W, \\ O_2 &= V^* \setminus W(P_{1,2,3} \cup P_{1,3} \cup P_{2,3} \cup P_{4,5} \cup P_{4,5,6} \cup P_{4,5,7})W, \\ O_3 &= V^* \setminus W P_7 W. \end{aligned}$$

Claim: $L_1(\Gamma_2) \cap T^* = L$. If $\lambda \in L$, then $\lambda \in A_1$ is in $L_1(\Gamma_2) \cap T^*$. Moreover, as it will be clear from the reasons below, no other terminal strings can be obtained from λ. Also λ cannot be obtained from S, so if $\lambda \notin A_1$, then $\lambda \notin L_1(\Gamma_2)$. We will now proceed with proving $L_1(\Gamma_2) \cap T^+ = L \setminus \{\lambda\}$, considering that we start with a string $S \in W$.

Every rule of G is simulated in maximum seven evolution steps, in N_1, N_2, N_3, in that order. All terminal strings of L can thus be obtained at some time in node N_3 (and brought to N_1). Now we only need to prove that $z \in L_1(\Gamma) \cap T^+ \Rightarrow z \in L \setminus \{\lambda\}$.

Let us think of symbols p_i as markers, telling us the current step of the simulation. If simulation of some production starts before the previous one is finished, then the string will have markers corresponding to both productions and will hence leave the current node without entering any node.

The "correct" simulation of one production is the following.
Rule $A \to BC$:

- In N_1: $W\lambda\lambda \cdot AW \Rightarrow^{\lambda\to p_1} Wp_1 \cdot \lambda \cdot AW \Rightarrow^{\lambda\to p_2} Wp_1p_2AW$,
- In N_2: $Wp_1p_2AW \Rightarrow^{A\to p_3} Wp_1p_2p_3W \Rightarrow^{p_1\to B} WBp_2p_3W \Rightarrow^{p_2\to C} WBCp_3W$, or
  $Wp_1p_2AW \Rightarrow^{A\to p_3} Wp_1p_2p_3W \Rightarrow^{p_2\to C} Wp_1Cp_3W \Rightarrow^{p_1\to B} WBCp_3W$,
- In N_3: $WBCp_3W \Rightarrow^{p_3\to\lambda} WBC \cdot \lambda W$;

Rule $AB \to C$:

- In N_1: $WA \cdot \lambda \cdot BW \Rightarrow^{\lambda\to p_5} WAp_5BW$,
- In N_2: $WAp_5BW \Rightarrow^{A\to p_4} Wp_4p_5BW \Rightarrow^{B\to p_6} Wp_4p_5p_6W \Rightarrow^{p_6\to p_7} Wp_4p_5p_7W \Rightarrow^{p_4\to C} WCp_5p_7W$,
- In N_3: $WCp_5p_7W \Rightarrow^{p_5\to\lambda} WC \cdot \lambda \cdot p_7W \Rightarrow^{p_7\to\lambda} WC \cdot \lambda\lambda W$.

If at some step a different production is applied, or it is applied at a different position, the corresponding string will leave the node through the output filter and be discarded, as is shown by the following tables.

N_1,Shape	$\lambda \to p_1$	$\lambda \to p_2$	$\lambda \to p_5$
(0) W	(1)	out	$N_2(3)$
(1) WP_1W	out	$N_2(0)$	out

N_2,Shape	$A \to p_3$	$p_1 \to B$	$p_2 \to C$	$A \to p_4$	$B \to p_6$	$p_4 \to C$	$p_6 \to p_7$
(0) $WP_{1,2}W$	(1)	out	out	out	out	n/a	n/a
(1) $WP_{1,2,3}W$	out	(2)	(2′)	out	out	n/a	n/a
(2) $WP_{2,3}W$	out	n/a	$N_3(0)$	out	out	n/a	n/a
(2′) $WP_{1,3}W$	out	$N_3(0)$	n/a	out	out	n/a	n/a
(3) WP_5W	out	n/a	n/a	(4)	out	n/a	n/a
(4) $WP_{4,5}W$	out	n/a	n/a	out	(5)	special	n/a
(5) $WP_{4,5,6}W$	out	n/a	n/a	out	out	out	(6)
(6) $WP_{4,5,7}W$	out	n/a	n/a	out	out	$N_3(1)$	n/a

N_3,Shape	$p_3 \to \lambda$	$p_5 \to \lambda$	$p_7 \to \lambda$
(0) WP_3W	$N_1(0)$	n/a	n/a
(1) $WP_{5,7}W$	n/a	(2)	out
(2) WP_7W	n/a	n/a	$N_1(0)$

Again here "out" means that the resulting string is discarded (sent out, never brought in), and "n/a" means that the operation is not applicable to the string. There is one special case: in $N_2(4)$, instead of applying $A \to p_4$, $B \to p_6$, $p_4 \to C$ in that order, we apply $A \to p_4$ and then $p_4 \to C$, then the string is again in shape WP_5W. If $A = C$, then it was making one step back, and it does not generate anything new. If $A \neq C$, then there is no A before p_5 anymore, so any string produced in the next step will be discarded.

Thus, the only way to obtain a terminal string with all sentential forms not belonging to the output filter is to do a correct simulation of G, so all terminal strings generated by Γ belong to L, q.e.d.

Remark 4 *The construction is also valid for any graph H where N_1, N_2, N_3 are pairwise connected.*

If the graph has loops (for example, for $H = K'_3$), the system behaves exactly the same because the input filter and the output filter are disjoint for any node.

Corollary 3 *Nonrecursive languages can be generated by NEPs with three nodes.*

Let us take $L \notin REC$. Since the family of recursive languages is closed under intersection with regular languages, $L_1(\Gamma_2) \cap T^* = L$ implies $L_1(\Gamma_2) \notin REC$.

Corollary 4 *NEPs with four nodes can generate any RE language.*

Take $\Gamma_2 = (V, N_1, N_2, N_3, H)$ from the construction above. Consider $\Gamma'_2 = (V, N_1, N_2, N_3, N_4, H')$ where $N_4 = (\emptyset, \emptyset, T^*, \emptyset)$ and H' is any graph such that N_1 is connected to N_2, N_2 is connected to N_3, N_3 is connected to N_1 and N_4. The reasoning is the same, except the terminal strings will immediately leave node N_3, enter node N_4 and stay there. (It is not necessary to exclude T^* from the input filter of N_1 because it is not possible to obtain one terminal string from another one).

5 Conclusions

We improved the conditions for universality of NEPs in terms of nodes in the classical case, where only operations of one kind are allowed in each node. As well, we considered the case when more than one kind of operations is allowed in the same node and got some universality results for it too. Namely, we proved that NEPs with 3 nodes can generate any recursively enumerable language (modulo to terminal alphabet), and more we obtained the optimal results for mNEPS with one node (modulo to terminal alphabet). The question about computing power of NEPs with 2 nodes is open.

It is interesting to consider from this point of view the extension of NEPs, Hybrid Networks of Evolutionary Processors (HNEPs) ([7, 5, 8, 6]).

Acknowledgements The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first and the third authors also acknowledge the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034, and the third author is supported by the NATO project PST.CLG.976912

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] Castellanos, J., Martin-Vide, C., Mitrana, V., Sempere, J.: Networks of evolutionary processors. Acta Informatica **39**, Springer-Verlag (2003) 517-529.

[3] Margenstern, M., Paun, G., Rogozhin, Yu., Verlan, S.: Context-free insertion-deletion systems. *Theoretical Computer Science*, Elsevier, vol.**330**, issue 2 (2005) 339 - 348.

[4] Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages, vol.1-3. Springer, 1997.

[5] Martin-Vide, C., Mitrana, V., Perez-Jimenez, M., Sancho-Caparrini. F., Hybrid networks of evolutionary processors. In: Proc. of GECCO 2003, 2003, 401-412.

[6] Margenstern, M., Mitrana, V.. Perez-Jimenez, M., Hibrid Networks of Evolutionary Processors as Accepting Devices, in progress.

[7] Csuhaj-Varju, E., Martin-Vide. C., Mitrana, V., Hybrid networks of evolutionary processors are Computationaly Complete, (submitted).

[8] Castellanos, J., Leupold, P., Mitrana, V., Descriptional and Computational Complexity Aspects of Hybrid Networks of Evolutionary Processors, submitted.