Artiom Alhazov | Carlos Martín-Vide | Yurii Rogozhin

Networks of Evolutionary Processors with Two Nodes Are Unpredictable

Turku Centre *for* Computer Science

TUCS Technical Report
No 818, April 2007

Networks of Evolutionary Processors with Two Nodes Are Unpredictable

Artiom Alhazov
Åbo Akademi University, Department of Information Technologies,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
aalhazov@abo.fi
Carlos Martín-Vide
Rovira i Virgili University, Research Group on Mathematical Linguistics,
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
{artiome.alhazov@estudiants., carlos.martin@}urv.cat
Yurii Rogozhin
Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Str. Academiei 5, Chişinău, MD-2012 Moldova
{artiom,rogozhin}@math.md

TUCS Technical Report
No 818, April 2007

Abstract

Networks of evolutionary processors are distributed communicating computational devices motivated by cell biology. It is known that every recursively enumerable language can be generated by some network of evolutionary processors with three nodes modulo a terminal alphabet. In this paper we present an unexpected result that networks of evolutionary processors with two nodes using only insertion and deletion operations are still powerful and can generate non-recursive languages.

Keywords: Networks of evolutionary processors, Distributed parallel computing, Point mutations, Filters, Universality

TUCS Laboratory
Computational Biomodelling Laboratory

1 Introduction

Starting from the premise that data can be given in the form of words, [6] introduces a concept called network of parallel language processors in the aim of investigating this concept in terms of formal grammars and languages. The main idea is that one can place a language generating device (grammar, Lindenmayer system, etc.) in any node of an underlying graph which rewrites the words existing in the node, then the words are communicated to the other nodes. Words can be successfully communicated if they pass some output and input filter.

In [3, 4], this concept was modified 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 genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations.

More formally, a network of evolutionary processors (NEP shortly) consists of a graph of nodes having operations and filters. A language is associated with a node at any moment of time. 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, [14]), while insertion/deletion systems are universal with weights 2/3 and 3/2 (see [12]).

A computation consists of two kinds of steps: a derivation step and a communication step. In a derivation step any node generate new language from language associated with it. In a communication step a node sends generated words to other nodes in the case if outgoing words can pass an output filter of the node and takes words sent by the other nodes if the words can pass an input filter of the node (input and output filters are regular sets). The language generated by a network of evolutionary processors consist of all (terminal) words which occur in the language associated with a given (output) node.

In our previous work we improved the existing universality results for NEPs with five and six nodes (see [4]) down to four nodes (three nodes, if we consider output words only in the terminal alphabet) and considered a variant of NEP (called mNEP) where operations of different kinds (symbol insertion, symbol substitution and symbol deletion) are allowed in the same node. We proved that mNEPs with two nodes are already enough for universality. This is not true for mNEPs with one node (thus we obtain the optimal result for mNEPs), while mNEPs with one node generate any recursively enumerable

language modulo a terminal alphabet [1].

In this paper we present an unexpected result that we can avoid symbol substitution and use only symbol deletion and symbol insertion operations and obtain still powerful NEPs with two nodes (they can generate non-recursive languages). Finally, we formulate some open problems.

2 Preliminaries

Consider a finite alphabet V. Given a word $u \in V^*$, we define the following sets (partial prefixes, partial suffixes and non-empty suffixes of u, respectively).

$$\begin{aligned} PPref(u) &= \{x \mid u = xy,\ |y| \geq 1\}, \\ PSuf(u) &= \{y \mid u = xy,\ |x| \geq 1\}, \\ NSuf(u) &= \{y \mid u = xy,\ |y| \geq 1\}. \end{aligned}$$

We will use the following notations for some known language families. REG, CS, REC, RE stand for regular, context-sensitive, recursive and recursively enumerable languages, respectively.

2.1 Networks of Evolutionary Processors

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, $1 \leq i \leq n$:*

- *M_i is a finite set of evolutionary rules of a certain type, i.e.,*
  $M_i \in 2^{\{a \to b \mid a,b \in V\}}$ (substitution rules) or
  $M_i \in 2^{\{a \to \lambda \mid a \in V\}}$ (deletion rules) or
  $M_i \in 2^{\{\lambda \to b \mid b \in V\}}$ (insertion rules).

- *A_i is a finite set of strings over V (the initial strings).*

- *I_i and O_i are regular languages over V specifying conditions for a string to enter and to exit a node, respectively (the input and the 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.

- Evolution step $C = (C[1], \cdots, C[n]) \Rightarrow C' = (C'[1], \cdots, C'[n])$: $C'[i] = M_i(C[i]) = \cup_{r \in M_i, w \in C[i]} r(w)$, where $r(w)$ is the set of strings 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 $i \geq 0$. The result of a (possibly infinite) computation is a language collected in a designated node N_k called the output node. Thus, $L_k(\Gamma) = \cup_{t \geq 0} C_t[k]$.

3 Main Result

Theorem 1 *There exists a morphism μ such that for any $L \in RE$, $L \subseteq T^*$, where T is a finite alphabet, there is a NEP Γ with two nodes such that*

$$\mu^{-1}(L_2(\Gamma) \cap T^*) = L.$$

Proof. Consider a type-0 grammar $G = (N, T, P, S)$, where N is a non-terminal alphabet, T is a terminal alphabet, $N \cap T = \emptyset$, P is a finite set of rewriting rules $u \to v$, $u \in (N \cup T)^* N (N \cup T)^*$, $v \in (N \cup T)^*$, $S \in N$ and $L(G) = L$.

We define a morphism μ by $\mu(a) = aa$ for $a \in N \cup T$ and $\mu(aw) = \mu(a)\mu(w)$, $a \in N \cup T$, $w \in (N \cup T)^*$. Let us denote $P' = \{p_i \mid p \in P,\ 1 \leq i \leq 4\}$, $W = \{\mu(w) \mid w \in (N \cup T)^*\}$. We construct the following NEP with two nodes:

$$\begin{aligned}
\Gamma \ = \ & (V, (M_1, A_1, I_1, O_1,), (M_2, A_2, I_2, O_2), H), \text{where} \\
& V = P' \cup N \cup T, \ H = K_2', \\
& M_1 = \{\lambda \to p_i \mid p_i \in P', 1 \leq i \leq 4\} \cup \{\lambda \to a \mid a \in N \cup T\}, \\
& A_1 = \{\mu(S)\}, \\
& I_1 = W, \\
& O_1 = V^* \setminus W R_{1,1} W, \\
& M_2 = \{p_i \to \lambda \mid p_i \in P', 1 \leq i \leq 4\} \cup \{a \to \lambda \mid a \in N \cup T\}, \\
& A_2 = \emptyset, \\
& I_2 = W R_{1,2} W, \\
& O_2 = V^* \setminus W R_{2,2} W.
\end{aligned}$$

We define $R_{1,1}, R_{1,2}$ and $R_{2,2}$ as follows:

$$
\begin{aligned}
R_{1,1} &= \bigcup_{p:u\to v\in P} (\{p_1\mu(u), p_1\mu(u)p_3, p_1p_2\mu(u)p_3, p_1p_2\mu(u)p_3p_4\} \\
&\qquad \cup \{p_1p_2\mu(u)\}PPref(\mu(v))\{p_3p_4\}) \\
&\quad \backslash\{p_1p_2\mu(u)p_3p_4 \mid p : u \to \lambda \in P\}, \\
R_{1,2} &= \{p_1p_2\mu(uv)p_3p_4 \mid p : u \to v \in P\}, \\
R_{2,2} &= \bigcup_{p:u\to v\in P} (\{p_1p_2\}PSuf(\mu(u))\{\mu(v)p_3p_4\} \\
&\qquad \cup \{p_2\mu(v)p_3p_4, p_2\mu(v)p_4, \mu(v)p_4\}).
\end{aligned}
$$

The output and input 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. Since morphism $\mu(a) = aa$ is introduced, the strings obtained by applying rules to the left or to the right of the place of application of the current rule no longer satisfy the parity (recall that $W = \{aa \mid a \in N \cup T\}^*$), so they leave the system.

Claim: $\mu^{-1}(L_2(\Gamma) \cap T^*) = L$.

The "correct" simulation of one production is the following.
Consider application of a production $p : a_1 \cdots a_s \to b_1 \cdots b_t$ to a sentential form $\alpha a_1 \cdots a_s \beta$; let $x = \mu(\alpha)$, $y = \mu(\beta)$.

- In N_1: $xa_1a_1 \cdots a_sa_sy \Rightarrow^{\lambda\to p_1} xp_1a_1a_1 \cdots a_sa_sy$
  $\Rightarrow^{\lambda\to p_3} xp_1a_1a_1 \cdots a_sa_sp_3y \Rightarrow^{\lambda\to p_2} xp_1p_2a_1a_1 \cdots a_sa_sp_3y$
  $\Rightarrow^{\lambda\to p_4} xp_1p_2a_1a_1 \cdots a_sa_sp_3p_4y \Rightarrow^{\lambda\to b_1} xp_1p_2a_1a_1 \cdots a_sa_sb_1p_3p_4y$
  $\Rightarrow^{\lambda\to b_1} xp_1p_2a_1a_1 \cdots a_sa_sb_1b_1p_3p_4y \Rightarrow^* xp_1p_2a_1a_1 \cdots a_sa_sb_1b_1 \cdots b_sp_3p_4y$
  $\Rightarrow^{\lambda\to b_s} xp_1p_2a_1a_1 \cdots a_sa_sb_1b_1 \cdots b_sb_sp_3p_4y$

- In N_2: $xp_1p_2a_1a_1 \cdots a_sa_sb_1b_1 \cdots b_sb_sp_3p_4y$
  $\Rightarrow^{a_1\to\lambda} xp_1p_2a_1 \cdots a_sa_sb_1b_1 \cdots b_sb_sp_3p_4y \Rightarrow^* xp_1p_2a_sa_sb_1b_1 \cdots b_sb_sp_3p_4y$
  $\Rightarrow^{a_s\to\lambda} xp_1p_2a_sb_1b_1 \cdots b_sb_sp_3p_4y \Rightarrow^{a_s\to\lambda} xp_1p_2b_1b_1 \cdots b_sb_sp_3p_4y$
  $\Rightarrow^{p_1\to\lambda} xp_2b_1b_1 \cdots b_sb_sp_3p_4y \Rightarrow^{p_3\to\lambda} xp_2b_1b_1 \cdots b_sb_sp_4y$
  $\Rightarrow^{p_2\to\lambda} xb_1b_1 \cdots b_sb_sp_4y \Rightarrow^{p_4\to\lambda} xb_1b_1 \cdots b_sb_sy$

Notice that if production can be applied to the same sentential form in different ways (multiple productions and/or multiple places to apply them), then the corresponding number of strings is produced in the first step (inserting marker p_1 associated to the production, to the left of the application place). The rest of the simulation is "deterministic": starting from $xp_1a_1a_1 \cdots a_sa_sy$, the result $xb_1b_1 \cdots b_sb_sy$ is obtained according to the the derivations above, while all other strings are discarded.

The strings that leave one node and enter another one are: $O_1 \cap I_2 = I_2$ and $O_2 \cap I_1 = I_1$. All other strings that leave any node do not enter anywhere.

Case 1: "incorrect" insertions in node N_1. Case 2: "incorrect" deletions in node N_2. The only strings that remain in the system are listed in the tables below, 5 situations in either case.

The following tables illustrate the behaviour of a string.

N	Shape in N_1	$\lambda \to p_1$	$\lambda \to p_3$	$\lambda \to p_2$	$\lambda \to p_4$	$\lambda \to A$
1	W	2	out	out	out	out
2	$Wp_1\mu(u)W$	out	3	out	out	out
3	$Wp_1\mu(u)p_3W$	out	out	4	out	out
4	$Wp_1p_2\mu(u)p_3W$	out	out	out	5	out
5	$Wp_1p_2\mu(u)$ $\cdot PPref(\mu(v))p_3p_4W$	out	out	out	out	5,$N_2(1)$

N	Shape in N_2	$p_1 \to \lambda$	$p_3 \to \lambda$	$p_2 \to \lambda$	$p_4 \to \lambda$	$A \to \lambda$
1	$Wp_1p_2NSuf(\mu(u))$ $\cdot\mu(v)p_3p_4W$	out	out	out	out	1,2
2	$Wp_1p_2\mu(v)p_3p_4W$	3	out	out	out	out
3	$Wp_2\mu(v)p_3p_4W$	n/a	4	out	out	out
4	$Wp_2\mu(v)p_4W$	n/a	n/a	5	out	out
5	$W\mu(v)p_4W$	n/a	n/a	n/a	$N_1(1)$	out

These tables illustrate the fact that if a symbol is inserted or deleted in a way that does not follow the "correct" simulation, than the string leaves the system.

Finally, consider $L_2(G) \cup T$. It is the set of all strings obtained in N_2 without non-terminal symbols and without markers. Hence, all of them are obtained from shape 5 of N_2 by deleting the marker p_4. This exactly corresponds to the set of terminal strings produced by the underlying grammar G, all letters being represented by a double repetition, i.e., encoded by μ. □

Corollary 1 *There exists a NEP Γ with two nodes such that $L_2(\Gamma) \notin REC$.*

Let us take $L \notin REC$. Since the family of recursive languages is closed under intersection with regular languages, $L_2(\Gamma) \cap T^* = L$ implies $L_2(\Gamma) \notin REC$.

4 Discussion

The computational power of networks of evolutionary processors presents a considerable interest, so we try to characterize it, depending on the number of nodes and operations allowed.

Below is a short summary of the best known results ([0] means this article) and open questions we consider interesting.

All $\not\supseteq REG$ statements are obtained from the impossibility to generate a language $(aa)^*$, see [1]. All $\not\subseteq REC$ statements are obtained by arguments similar to that in Corollary 1. The statements $\subseteq CS$ are valid for networks generating words by only insertion and verifying a regular condition: it can easily be done with linear workspace.

Clearly, considering a more general system we can transfer the results on the lower bounds, while considering a particular case we can transfer the results on the upper bounds.

n	types of rules	$\cap T^*$	encoded	power	ref
1	all, mixed	yes	no	RE	[1]
1	all, mixed	no	no	$\not\subseteq REC$, $\not\supseteq REG$	[1]
1	all, mixed	no	yes	$\not\subseteq REC$, ?	[1]
2	all, mixed	no	no	RE	[1]
3	all, one	yes	no	RE	[1]
3	all, one	no	no	$\not\subseteq REC$, ?	[1]
3	all, one	no	yes	$\not\subseteq REC$, ?	[1]
4	all, one	no	no	RE	[1]

n	types of rules	$\cap T^*$	encoded	power	ref
1	insdel, mixed	no	no	$\not\supseteq REG$, ?	[0]
1	insdel, mixed	yes	no	$\not\supseteq REG$, ?	[0]
1	insdel, mixed	no	yes	$\not\supseteq REG$, ?	[0]
1	insdel, mixed	yes	yes	$\not\supseteq REG$, ?	[0]
2	insdel, mixed	no	no	$\not\subseteq REC$, ?	[0]
2	insdel, mixed	yes	no	$\not\subseteq REC$, ?	[0]
2	insdel, mixed	no	yes	$\not\subseteq REC$, ?	[0]
1	ins, one	yes	yes	$\not\supseteq REG$, ?	[0]
1	ins, one	no	no	$\not\supseteq REG$, $\subseteq CS$	[0]
1	ins, one	yes	no	$\not\supseteq REG$, $\subseteq CS$	[0]
1	ins, one	no	yes	$\not\supseteq REG$, ?	[0]
2	insdel, one	yes	yes	RE	[0]
2	insdel, one	no	no	$\not\subseteq REC$, ?	[0]
2	insdel, one	yes	no	$\not\subseteq REC$, ?	[0]
2	insdel, one	no	yes	$\not\subseteq REC$, ?	[0]
3	insdel, one	no	no	$\not\subseteq REC$, ?	[0]

Acknowledgements

The first author gratefully acknowledges the support by Academy of Finland, project 203667. The first and the third authors acknowledge the project

06.411.03.04P from the Supreme Council for Science and Technological Development of the Academy of Sciences of Moldova.

References

[1] Alhazov, A., Martín-Vide, C., Rogozhin, Yu.: On the Number of Nodes in Universal Networks of Evolutionary Processors. Acta Informatica **43**(5), Springer, 331–339 (2006)

[2] Castellanos, J., Leupold, P., Mitrana, V.: On the size complexity of hybrid networks of evolutionary processors. Theoretical Computer Science **330**(2), Elsevier, 205–220 (2005)

[3] Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.: Solving NP-complete problems with networks of evolutionary processors. In: Proceedings of IWANN 2001. Lecture Notes in Computer Science **2084**, Springer, 621–628 (2001)

[4] Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.: Networks of evolutionary processors. Acta Informatica **39**(6-7) Springer, 517–529 (2003)

[5] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar systems, Gordon and Breach (1993)

[6] Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In: New Trends in Formal Languages, ed. by Gh. Paun, A. Salomaa. Lecture Notes in Computer Science **1218**, Springer, 299–318 (1997)

[7] Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V., Hybrid networks of evolutionary processors are computationaly complete. Acta Informatica **41**(4-5), Springer, 257–272 (2005)

[8] Errico, L., Jesshope, C.: Towards a new architecture for symbolic processing. In: Artificial Intelligence and Information-Control Systems of Robots '94, ed. by I. Plander, World Scientific, Singapore, 31–40 (1994)

[9] 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, 109–113 (1983)

[10] Hillis, W.D.: The Connection Machine. MIT Press, Cambridge (1985)

[11] Margenstern, M., Mitrana, V., Pérez-Jiménez, M.: Accepting hybrid networks of evolutionary processors. In: C.Ferretti, G.Mauri, C.Zandron (Eds.), Pre-proc. of 10th International Workshop on DNA Computing, DNA 10, Milan, Italy, June 7–10, 2004, 107–117. Revised Selected Papers, Lecture Notes in Computer Science **3384**, Springer, 235–246 (2005)

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

[13] Martín-Vide, C., Mitrana, V., Pérez-Jiménez, M., Sancho-Caparrini. F.: Hybrid networks of evolutionary processors. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO 2003), Lecture Notes in Computer Science **2723**, Springer, 401–412 (2003).

[14] Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol.1-3. Springer, Berlin Heidelberg New York (1997)

[15] Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B.F., Cedergren, R.: Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. In: Proc. Nat. Acad. Sci. USA, **89**, 6575–6579 (1992)

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-1879-8
ISSN 1239-1891