Hybrid Networks of Evolutionary Processors

Carlos Martín-Vide[1] **Victor Mitrana**[2*]
Mario J. Pérez-Jiménez[3] **Fernando Sancho-Caparrini**[3]

[1]Research Group in Mathematical Linguistics
Rovira i Virgili University
Pça. Imperial Tàrraco 1, 43005 Tarragona, Spain
cmv@correu.urv.es

[2]Faculty of Mathematics, University of Bucharest
Str. Academiei 14, 70109 Bucharest, Romania
mitrana@funinf.math.unibuc.ro

[3]Department of Computer Science and Artificial Intelligence
University of Seville
{Mario.Perez,Fernando.Sancho}@cs.us.es

Abstract

A hybrid network of evolutionary processors consists of several processors which are placed in a node of a virtual graph and can perform one simple operation only on the words existing in that node in accordance with some strategies. Then the words which can pass the output filter of each node navigate simultaneously through the network and enter those nodes whose input filter was passed. We prove that these networks with filters defined by simple random-context conditions, used as language generating devices, are able to generate all linear languages in a very efficient way, as well as non-context-free languages. Then, when using them as computing devices, we present two linear solutions of the Common Algorithmic Problem.

*This work, done when this author was visiting the Department of Computer Science and Artificial Intelligence of the University of Seville, was supported by the Generalitat de Catalunya, Direcció General de Recerca (PIV2001-50)

1 Introduction

This work is a continuation of the investigation started in [1] and [2] where one has 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.

Another source of inspiration is a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [13] as well as the Logic Flow paradigm [6]. This 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, and, then 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 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., [7, 13].

Starting from the premise that data can be given in the form of strings, [4] 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 [3]. 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 filter.

Our mechanisms introduced in [1] and [2] simplify as much as possible the networks of parallel language processors defined in [4]. Thus, in each node is placed a very simple processor, called evolutionary processor, which is able to perform a simple rewriting operation only, namely either insertion of a symbol or substitution of a symbol by another, or deletion of a symbol. Furthermore, filters used in [4] are simplified in some versions defined in [1, 2].

In spite of these simplifications, these mechanisms are still powerful. In [2] networks with at most six nodes having filters defined by the membership to a regular language condition are able to generated 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., [5, 11, 10, 12, 14]. Then we

have considered networks with nodes having filters defined by random context conditions which seem to be closer to the biological possibilities of implementation. Even in this case, rather complex languages like non-context-free ones, can be generated.

However, these very simple mechanisms are able to solve hard problems in polynomial time. In [1] it is presented a linear solution for an NP-complete problem, namely the Bounded Post Correspondence Problem, based on networks of evolutionary processors able to substitute a letter at any position in the string but insert or delete a letter in the right end only.

This restriction was discarded in [2], but the new variants were still able to solve in linear time another NP-complete problem, namely the "3-colorability problem".

In the present paper, we consider hybrid networks of evolutionary processors in which each deletion or insertion node has its own working mode (at any position, in the left end, or in the right end) and its own way of defining the input and output filter. Thus, in the same network one may co-exist nodes in which deletion is done at any position and nodes in which deletion is done in the right end only. Also the definition of the filters of two nodes, though both are random context ones, may differ.

This model may be viewed as a biological computing model in the following way: each node is a cell having a genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations (insertion, deletion or substitution of a pair of nucleotides). Each node is specialized just for one of these evolutionary operations. Furthermore, the biological data in each node is organized in the form of arbitrarily large multisets of strings (each string appears in an arbitrarily large number of copies), each copy being processed in parallel such that all the possible evolutions events that can take place do actually take place. 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 consideration local mutations only [15]. Furthermore, we were not concerned here with a possible biological implementation, though a matter of great importance.

The paper is organized as follows: in the next section we recall the some basic notions from formal language theory and define the hybrid networks of evolutionary processors. Then, we briefly investigate the computational power of these networks as language generating devices. We prove that all regular languages over an n-letter alphabet can be generated in an efficient way by networks having the same underlying structure and show that this result can be extended to linear languages. Furthermore, we provide a non-context-free language which can be generated by such networks. The last section is dedicated to hybrid networks of evolutionary processors

viewed as computing (problem solving) devices; we present two linear solutions of the so-called Common Algorithmic Problem. The latter one needs linearly bounded resources (symbols and rules) as well.

2 Preliminaries

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called *string (word)* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by ε. The length of a string x is denoted by $|x|$ while the number of occurrences of a letter a in a string x is denoted by $|x|_a$. Furthermore, for each nonempty string x we denote by $alph(x)$ the minimal alphabet W such that $x \in W^*$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule as above σ and a string $w \in V^*$, we define the following *actions* of σ on w:

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

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

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

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

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

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

$\alpha \in \{*, l, r\}$ expresses the way of applying an evolution rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word,

respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by:

$$\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$$

Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

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

respectively.

In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistical formulations of local gene mutations. For two disjoint subsets P and F of an alphabet V and a word over V, we define the predicates

$$\begin{array}{lllll} \varphi^{(1)}(w; P, F) \equiv & P \subseteq alph(w) & \wedge & F \cap alph(w) = \emptyset \\ \varphi^{(2)}(w; P, F) \equiv & alph(w) \subseteq P & \wedge & F \cap alph(w) = \emptyset \\ \varphi^{(3)}(w; P, F) \equiv & P \subseteq alph(w) & \wedge & F \not\subseteq alph(w) \\ \varphi^{(4)}(w; P, F) \equiv & alph(w) \subseteq P & \wedge & F \not\subseteq alph(w) \end{array}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*).

For every language $L \subseteq V^*$ and $\beta \in \{(1), (2), (3), (4)\}$, we define:

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

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

- Either $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only.
- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexs of the processor.

We denote the set of evolutionary processors over V by EP_V.

An *hybrid network of evolutionary processors* (HNEP for short) is a 7-tuple $\Gamma = (V, G, N, C_0, \alpha, \beta, i_0)$, where:

- V is an alphabet.

- $G = (X_G, E_G)$ is an undirected graph, without loops, with the set of vertices X_G and the set of edges E_G, each edge is given in the form of a set of two nodes. G is called the *underlying graph* of the network.
- $N : X_G \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $C_0 : X_G \longrightarrow 2^{V^*}$ is a mapping which identifies the initial configuration of the network. It associates a finite set of words with each node of the graph G.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $\beta : X_G \longrightarrow \{(1), (2), (3), (4)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

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

  That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the string w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of strings of L that can pass the input (resp. output) filter of x.
- $i_0 \in X_G$ is the *output node* of the HNEP.

We say that $card(X_G)$ is the size of Γ. If $\alpha(x) = \alpha(y)$ and $\beta(x) = \beta(y)$ for any pair of nodes $x, y \in X_G$, then the network is said to be *homogeneous*. 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 HNEP is said to be a *star, ring, or complete* HNEP if its underlying graph is a star, ring, grid, or complete graph, respectively. The star, ring, and complete graph with n vertices is denoted by S_n, R_n, and K_n, respectively.

A *configuration* of a HNEP Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of strings with every node of the graph. A configuration may be understood as the sets of strings which are present in any node at a given moment. A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

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

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each string it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the strings sent by any node processor connected with x providing that they can pass its input filter.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$ iff

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \text{ for all } x \in X_G.$$

Let Γ an HNEP, a computation in Γ is a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geq 0$. By the previous definitions, each configuration C_i is uniquely determined by the configuration C_{i-1}. If the sequence is finite, we have a finite computation. If one uses HNEPs as language generating devices, then the result of any finite or infinite computation is a language which is collected in the output node of the network.

For any computation $C_0, C_1, \ldots$, all strings existing in the output node at some step belong to the language generated by the network. Formally, the language generated by Γ is $L(\Gamma) = \bigcup_{s \geq 0} C_s(i_0)$.

The time complexity of computing a finite set of strings Z is the minimal number s such that $Z \subseteq \bigcup_{t=0}^{s} C_t(i_0)$.

3 Computational power of HNEP as language generating devices

First, we compare these devices with the simplest generative grammars in the Chomsky hierarchy. In [2], one proves that the families of regular and context-free languages are incomparable with the family of languages generated by homogeneous networks of evolutionary processors. HNEPs are more powerful, namely

Theorem 1 *Any regular language can be generated by any type (star, ring, complete) of HNEP.*

Proof. Let $A = (Q, V, \delta, q_0, F)$ be a deterministic finite automaton; without loss of generality we may assume that $\delta(q, a) \neq q_0$ holds for each $q \in Q$ and each $a \in V$. Furthermore, we assume that $card(V) = n$. We construct the following complete HNEP (the proof for the other underlying structures is left to the reader):

$$\Gamma = (U, K_{2n+3}, N, C_0, \alpha, \beta, f).$$

The alphabet U is defined by

$$U = V \cup V' \cup Q \cup \{s_a \mid s \in Q, a \in V\},$$

where $V' = \{a' \mid a \in V\}$. The set of nodes of the complete underlying graph is $\{x_0, x_1, x_f\} \cup V \cup V'$, and

Node	M	PI	FI	PO	FO	C_0	α	β
x_0	$\{q \to s_b\}_{\delta(s,b)=q}$	$\emptyset$	$\{s_b\}_{s,b} \cup \{b'\}_b$	$\emptyset$	$\emptyset$	F	$*$	(1)
$a \in V$	$\varepsilon \to a'$	$\{s_a\}_s \cup V$	Q	U	$\emptyset$	$\emptyset$	l	(2)
$a' \in V'$	$\{s_a \to s\}_s$	$\{a'\}$	Q	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)
x_1	$\{b' \to b\}_b$	$\emptyset$	$\{s_b\}_{s,b}$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)
x_f	$q_0 \to \varepsilon$	$\{q_0\}$	V'	$\emptyset$	V	$\emptyset$	r	(1)

Table 1.

In Table 1, s and r are generic states from Q and symbols from V, respectively. One can easily prove by induction that

1. $\delta(q, x) \in F$ for some $q \in Q \setminus \{q_0\}$ if and only if $xq \in C_{8|x|}(0)$.

2. x is accepted by A ($x \in L(A)$) if and only if $x \in C_p(f)$ for any $p \geq 8|x| + 1$.

Therefore, $L(A)$ is exactly the language generated by Γ. □

Surprisingly enough, the size of the above HNEP, hence its underlying structure, does not depend on the number of states of the given automaton. In other words, this structure is common to all regular languages over the same alphabet, no matter the state complexity of the automata recognizing them. Furthermore, all strings of the same length are generated simultaneously.

Since each linear grammar can be transformed into an equivalent linear grammar with rules of the form $A \to aB, A \to Ba, A \to \varepsilon$ only, the proof of the above theorem can be adapted for proving the next result.

Theorem 2 *Any linear language can be generated by any type of HNEP.*

We do not know whether these networks are able to generate all context-free languages, but they can generate non-context-free languages as shown below.

Theorem 3 *There are non-context-free languages that can be generated by any type of HNEP.*

Proof. We construct the following complete HNEP which generates the non-context-free language $L = \{wcx \mid x \in \{a,b\}^*, w \text{ is a permutation of } x\}$:

$$\Gamma = (V, K_9, N, C_0, \alpha, \beta, y_2),$$

where

$$V = \{a, b, a', b', X_a, X_b, X\}, \quad X_{K_9} = \{y_0, y_1, y_2, y_a, y_b, \bar{y}_a, \bar{y}_b, \tilde{y}_a, \tilde{y}_b\},$$

and

Node	M	PI	FI	PO	FO	C_0	α	β
y_0	$\{\varepsilon \to X, \varepsilon \to D\}$	$\emptyset$	$\{a', b', a, b, X_a, X_b\}$	$\{D\}$	$\emptyset$	$\{\varepsilon\}$	r	(1)
y_1	$\{\varepsilon \to X_a, \varepsilon \to X_b\}$	$\emptyset$	$\{X_a, X_b, a', b'\}$	$\emptyset$	$\emptyset$	$\emptyset$	r	(1)
y_u	$\{X \to u'\}$	$\{X_u\}$	$\{a', b'\}$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)
$\bar{y}_u$	$\{X_u \to u\}$	$\{u'\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)
$\tilde{y}_u$	$\{u' \to u\}$	$\{u'\}$	$\{X_a, X_b\}$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)
y_2	$\{D \to c\}$	$\emptyset$	$\{X, a', b', X_a, X_b\}$	$\emptyset$	$\{a, b\}$	$\emptyset$	$*$	(1)

Table 2.

In Table 2, u is a generic symbol in $\{a, b\}$. The working mode of this network is rather simple. In the node y_0 there are generated strings of the form X^n for any $n \geq 1$. They can leave this node as soon as they receive a D at their right end, the only node able to receive them being y_1. In y_1, either X_a or X_b is added to their right end. Thus, for a given n, the strings X^nDX_a and X^nDX_b are produced in y_1. Let us follow what happens with the strings X^nDX_a, a similar analysis applies to the strings X^nDX_b as well. So, X^nDX_a goes to y_a where any occurrence of X is replaced by a' in different identical copies of X^nDX_a. In other words, y_a produces each string $X^ka'X^{n-k-1}DX_a$, $0 \leq k \leq n-1$. All these strings are sent out but no node, except $\bar{y}_a$, can receive them. Here, X_a is replaced by a and the obtained strings are sent to $\tilde{y}_a$ where a is substituted to a'. As long as the strings contains occurrences of X, they follow the same itinerary, namely $y_1, y_u, \bar{y}_u, \tilde{y}_u$, $u \in \{a, b\}$, depending on what symbol X_a or X_b is added in y_1.

After a finite number of such cycles, when no occurrence of X is present in the strings, they are received by y_2 where D is replaced by c in all of them, and they remain in this node for ever. By these explanations, the node y_2 collects all strings of L and any string which arrives in this node belongs to L. □

A more precise characterization of the family of languages generated by HNEPs remains to be done.

4 Solving problems with HNEPs

HNEPs may be used for solving problems in the following way. For any instance of the problem the computation in the associated HNEP must be finite. In particular, this means that there is no node processor specialized in insertions. If the problem is a decision problem, then at the end of the computation, the output node provides all solutions of the problem encoded by strings, if any, otherwise this node will never contain any word. If the problem requires a finite set of words, this set will be in the output node at the end of the computation. In other cases, the result is collected by specific methods which will be indicated for each problem.

In [2] one provides a complete homogeneous NEP of size $7m + 2$ which solves in $O(m + n)$ time an instance of the "3-colorability problem" with n vertices and m edges.

In the sequel, following the descriptive format for three NP-complete problems presented in [9] we present a solution to the *Common Algorithmic Problem*. The three problems are:

1. The maximum independent set: Given an undirected graph $G = (X, E)$, where X is the finite set of vertices and E is the set of edges given as a family of sets of two vertices, find the cardinality of a maximal subset (with respect to inclusion) of X which does not contain both vertices connected by any edge in E.

2. The vertex cover problem: Given an undirected graph find the cardinality of a minimal set of vertices such that each edge has at least one of its extremes in this set.

3. Satisfiability problem: For a given set P of Boolean variables and a finite set U of clauses over P, does a truth assignment for the variables of P exist satisfying all the clauses of U?

For detailed formulations and discussions about their solutions, the reader is refered to [8].

These problems can be viewed as special cases of the following algorithmic problem, called the Common Algorithmic Problem (CAP) in [9]: let S be a finite set and F be a family of subsets of S. Find the cardinality of a maximal subset of S which does not include any set belonging to F. The sets in F are called *forbidden sets*.

Let us show how the three problems mentioned above can be obtained as special cases of CAP. For the first problem, we just take $S = X$ and $F = E$.

The second problem is obtained by letting $S = X$ and F contains all sets $o(x) = \{x\} \cup \{y \in X \mid \{x, y\} \in E\}$. The cardinality one looks for is the difference between the cardinality of S and the solution of the CAP.

The third problem is obtained by letting $S = P \cup P'$, where $P' = \{p' \mid p \in P\}$, and $F = \{F(C) \mid C \in U\}$, where each set $F(C)$ associated with the clause C is defined by

$$F(C) = \{p' \mid p \text{ appears in } C\} \cup \{p \mid \neg p \text{ appears in } C\}.$$

From this it follows that the given instance of the satisfiability problem has a solution if and only if the solution of the constructed instance of the CAP is exactly the cardinality of P.

First, we present a solution of the CAP based on homogeneous HNEPs.

Theorem 4 *Any instance of the CAP can be solved by a complete homogeneous HNEP of size $m + 2n + 2$ in O(m+n) time.*

Proof. Let $S = \{a_1, a_2, \ldots, a_n\}$ and $F = \{F_1, F_2, \ldots, F_m\}$, $m \geq 1$, be an instance of the CAP. We construct the complete homogeneous HNEP

$$\Gamma = (U, K_{m+2n+2}, N, C_0, \alpha, \beta).$$

Since the result will be collected in a way which will be specified later, the output node is missing.

The alphabet of the network is

$$\begin{aligned} U &= S \cup \bar{S} \cup S' \cup \{Y, Y_1, Y_2, \ldots, Y_{m+1}\} \cup \{b\} \cup \{Z_0, Z_1, \ldots, Z_n\} \cup \\ &\quad \{Y'_1, Y'_2, \ldots, Y'_{m+1}\} \cup \{X_1, X_2, \ldots, X_n\}, \end{aligned}$$

where $\bar{S}$ and S' are copies of S obtained by taking the barred and primed copies of all letters from S, respectively. The nodes of the underlying graph are:

$$x_0, x_{F_1}, x_{F_2}, \ldots, x_{F_m}, x_{a_1}, x_{a_2}, \ldots, x_{a_n}, y_0, y_1, \ldots, y_n.$$

The mapping N is defined by:

$$\begin{aligned} N(x_0) &= (\{X_i \to a_i, X_i \to \bar{a}_i \mid 1 \leq i \leq n\} \cup \{Y \to Y_1\} \cup \{Y'_i \to Y_{i+1} \mid \\ &\quad 1 \leq i \leq m\}, \{Y'_i \mid 1 \leq i \leq m\}, \emptyset, \emptyset, \{X_i \mid 1 \leq i \leq n\} \cup \{Y\}), \\ N(x_{F_i}) &= (\{\bar{a} \to a' \mid a \in F_i\}, \{Y_i\}, \emptyset, \emptyset, \emptyset), \\ &\quad \text{for all } 1 \leq i \leq m, \\ N(x_{a_j}) &= (\{a'_j \to \bar{a}_j\} \cup \{Y_i \to Y'_i \mid 1 \leq i \leq m\}, \{a'_j\}, \emptyset, \emptyset, \{a'_j\} \cup \{Y_i \mid \\ &\quad 1 \leq i \leq m\}), \text{ for all } 1 \leq j \leq n, \\ N(y_n) &= (\{\bar{a}_i \to b \mid 1 \leq i \leq n\} \cup \{Y_{m+1} \to Z_0\}, \{Y_{m+1}\}, \emptyset, \{Z_0, b\}, \bar{S}), \\ N(y_{n-i}) &= (\{b \to Z_i\}, \{Z_{i-1}\}, \emptyset, \{b, Z_i\}, \emptyset), \\ &\quad \text{for all } 1 \leq i \leq n. \end{aligned}$$

The initial configuration C_0 is defined by

$$C_0(x) = \begin{cases} \{X_1X_2 \dots X_nY\} \text{ if } x = x_0 \\ \emptyset, \text{ otherwise} \end{cases}$$

Finally, $\alpha(x) = *$ and $\beta(x) = (1)$, for any node x.

A few words on how the HNEP above works: in the first $2n$ steps, in the first node one obtains 2^n different words $w = x_1x_2 \dots x_nY$, where each x_i is either a_i or $\bar{a}_i$. Each such string w can be viewed as encoding a subset of S, namely the set containing all symbols of S which appear in w. After replacing Y by Y_1 in all these strings they are sent out and x_{F_1} is the only node which can receive them. After one rewriting step, only those strings encoding subsets of S which do not include F_1 will remain in the network, the others being lost. The strings which remain are easily recognized since they have been obtained by replacing a barred copy of symbol with a primed copy of the same symbol. This means that this symbol is not in the subset encoded by the string but in F_1. In the nodes x_{a_i} the modified barred symbols are restored and the symbol Y_1' is substituted for Y_1. Now, the strings go to the node x_0 where Y_2 is substituted for Y_1' and the whole process above resumes for F_2. This process lasts for $8m$ steps.

The last phase of the computation makes use of the nodes y_j, $0 \leq j \leq n$. The number we are looking for is given by the largest number of symbols from S in the strings from y_n. It is easy to note that the strings which cannot leave y_{n-i} have exactly $n - i$ such symbols, $0 \leq i \leq n$. Indeed, only the strings which contains at least one occurrence of b can leave y_n and reach y_{n-1}. Those strings which do not contain any occurrence of b have exactly n symbols from S. In y_{n-1}, Z_1 is substituted for an occurrence of b and those strings which still contain b leave this node for y_{n-2} and so forth. The strings which remain here contain $n - 1$ symbols from S. Therefore, when the computation is over, the solution of the given instance of the CAP is the largest j such that y_j is nonempty. The last phase is over after at most $4n + 1$ steps. By the aforementioned considerations, the total number of steps is at most $8m + 4n + 3$, hence the time complexity of solving each instance of the CAP of size (n, m) is $O(m + n)$.

As far as the time and memory resources the HNEP above uses, the total number of symbols is $2m + 5n + 4$ and the total number of rules is

$$mn + m + 5n + 2 + \sum_{i=1}^{m} card(F_i).$$

□

The same problem can be solved in a more economic way, regarding especially the number of rules, with HNEPs, namely

Theorem 5 *Any instance of the CAP can be solved by a complete HNEP of size* $m+n+1$ *in O(m+n) time.*

Proof. For the same instance of the CAP as in the previous proof, we construct the complete HNEP

$$\Gamma = (U, K_{m+n+1}, N, C_0, \alpha, \beta)$$

The alphabet of the network is

$$U = S \cup S' \cup \{Y_1, Y_2, \ldots, Y_{m+1}\} \cup \{b\} \cup \{Z_0, Z_1, \ldots, Z_n\}.$$

The other parameters of the network are given in Table 3.

Node	M	PI	FI	PO	FO	C_0	α	β
x_0	$\{a'_i \to a_i\}_i$ $\{a'_i \to T\}_i$	$\{a'_1\}$	$\emptyset$	$\emptyset$	$\{a'_i\}_i$	$\{a'_1 \ldots a'_n Y_1\}$	$*$	(1)
x_{F_j}	$\{Y_j \to Y_{j+1}\}$	$\{Y_j\}$	F_j	$\emptyset$	U	$\emptyset$	$*$	(3)
y_n	$\{T \to Z_0\}$	$\{Y_{m+1}\}$	$\emptyset$	$\{T\}$	$\emptyset$	$\emptyset$	$*$	(1)
y_{n-i}	$\{T \to Z_i\}$	$\{Z_{i-1}\}$	$\emptyset$	$\{T\}$	$\emptyset$	$\emptyset$	$*$	(1)

Table 3.

In the table above, i ranges from 1 to n and j ranges from 1 to m.

The reasoning is rather similar to that from the previous proof. The only notable difference concerns the phase of selecting all strings which do not contain any symbol from any set F_j. This selection is simply accomplished by the way of defining the filters of the nodes x_{F_j}. The time complexity is now $2m+4n+1 \in O(m+n)$, while the needed resources are: $m+3n+3$ symbols and $m+3n+1$ rules. □

References

[1] J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors, *IWANN 2001* (J. Mira, A. Prieto, eds.), LNCS 2084, Springer-Verlag, 2001, 621–628.

[2] J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, submitted 2002.

[3] E. Csuhaj - Varju, J. Dassow, J. Kelemen, Gh. Paun - *Grammar Systems*, Gordon and Breach, 1993.

[4] E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors. In *New Trends in Formal Languages* (Gh. Păun, A. Salomaa, eds.), LNCS 1218, Springer Verlag, 1997, 299–318

[5] E. Csuhaj-Varjú, V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* 36(2000), 913–926.

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

[7] S. E. Fahlman, G. E. Hinton, T. J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.

[8] M. Garey, D. Johnson, *Computers and Intractability. A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, 1979.

[9] T. Head, M. Yamamura, S. Gal, Aqueous computing: writing on molecules, in *Proc. of the Congress on Evolutionary Computation 1999*, IEEE Service Center, Piscataway, NJ, 1999, 1006–1010.

[10] Kari, L.: *On Insertion and Deletion in Formal Languages*, Ph.D. Thesis, University of Turku, 1991.

[11] 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.

[12] Kari, L., Thierrin, G.: Contextual insertion/deletion and computability. *Information and Computation* 131, 1(1996), 47–61.

[13] W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.

[14] 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.

[15] D. Sankoff et al. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, **89**(1992) 6575–6579.