Networks of Evolutionary Processors: A Survey

Carlos MARTÍN-VIDE, Victor MITRANA

ABSTRACT: The goal of this paper is to survey, in a uniform and systematic way, the main results regarding networks of evolutionary processors reported so far. First, we recall the results concerning the computational power of these networks viewed as language generating devices. Then, we briefly present a few NP-complete problems and recall how they were solved in linear time by networks of evolutionary processors with linearly bounded resources (nodes, rules, symbols).

Keywords: Networks, evolution, grammar, formal language, NP-complete problems, generative capacity, theory of computing.

1 Introduction

A basic architecture for parallel and distributed symbolic processing, related to the Connection Machine (Hillis 1985) as well as to the Logic Flow paradigm (Errico & Jesshope 1994), 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., Fahlman & Hinton 1983, Hillis 1985).

Starting from the premise that data can be given in the form of strings, (Csuhaj-Varjú 1997) introduces a concept called *network of parallel language processors* with 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 (Csuhaj-Varjú et al. 1994). 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, and then the strings are communicated to the other nodes. Strings can be successfully communicated if they pass some output and input filters. More recently, (Csuhaj-Varjú & Salomaa to appear) introduces networks whose nodes are (standard) Watson-Crick D0L systems which communicate each other either the correct words or the corrected words.

In (Castellanos et al. 2001), 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 a 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 such evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of strings, each copy being processed in parallel in such a way that all the possible evolutionary events that can take place do actually take place.

Our mechanisms introduced in (Castellanos et al. 2001) are further considered in (Castellanos et al. submitted) as language generating devices and their computational power in this respect is investigated. Furthermore, filters, based on the membership condition, used in (Csuhaj-Varjú 1997) are simplified in some versions defined in (Castellanos et al. 2001, submitted). More precisely, the new filters are based on random-context conditions.

In spite of these simplifications, these mechanisms are still powerful. In (Castellanos et al. submitted) one proves that 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 is not a surprise, since similar characterizations have been reported in the literature (see, e.g., [Kari 1991, Kari et al. 1997, Kari & Thierrin 1997, Martín-Vide & Păun 1998). Then, we consider 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. In (Martín-Vide et al. submitted) one considers a natural extension of networks of evolutionary processors, namely 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 nodes in which deletion is done at any position and nodes in which deletion is done in the right end only may co-exist. Also the definition of the filters of two nodes, though both are random-context ones, may differ. These networks turned out to generate all recursively enumerable languages (Csuhaj-Varjú et al. submitted).

Networks of evolutionary processors (NEP) may be used for solving problems in the following way. For any instance of the problem, the computation in the associated NEP is 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.

Despite of their simplicity, these mechanisms are able to solve hard problems in polynomial time. In (Castellanos et al. 2001) 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 (Castellanos et al. submitted), but the new variants were still able to solve another NP-complete problem, namely the "3-colorability problem", in linear time.

In (Martín-Vide et al. submitted), following the descriptive format for three NP-complete problems presented in (Head et al. 1999), one presents a solution to the Common Algorithmic Problem.

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 multisets of strings, each copy being processed in parallel such that all the possible evolutionary 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. However, 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 (Sanko et al. 1992). Furthermore, we were not concerned here with a possible biological implementation, though a matter of great importance.

A similar concept is that introduced in (Csuhaj-Varjú & Mitrana 2000), inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactic level. Cells are represented by strings which describe their DNA sequences.

Informally, at any moment of time, the evolutionary system is described by a collection of strings, where each string represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on strings. Only those cells are accepted as surviving (correct) ones which are represented by a string in a given set of strings, called the genotype space of the species. This feature parallels with the natural process of evolution. It is worth mentioning that any recursively enumerable language is a language of a species of an evolutionary system with point mutations of restricted forms. In the aforementioned paper, a connection between Lindenmayer systems (language theoretical models of developmental systems) and evolutionary systems is established, namely the growth function of any deterministic 0*L* system can be obtained from the population growth relation of some (deterministic) evolutionary system.

2 Basic Definitions

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 a *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 $a, b \in V \cup \{e\}$, is a *substitution rule* if both a and b are not e; it is a *deletion rule* if $a \neq e$ and $b = e$; it is an *insertion rule* if $a = e$ and $b \neq e$. 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:

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

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

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

$$s^r(w) = \begin{cases} \{u : w = ua)\}, \\ \{w\}, \text{ otherwise.} \end{cases}$$

$$s^l(w) = \begin{cases} \{v : w = av)\}, \\ \{w\}, \text{ otherwise.} \end{cases}$$

- If $\sigma \equiv e \to a \in Ins_V$, 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:

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

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

$$M^\alpha(w) = \bigcup_{s \in M} s^\alpha(w),\ M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),\ \text{respectively.}$$

In what follows, we shall refer to the rewriting operations defined above as evolutionary operations, since they may be viewed as linguistic 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:

$$\begin{aligned}
\varphi^a(w;P,F) &\equiv P \subseteq alph(w) \wedge F \cap alph(w) = \emptyset \\
\varphi^b(w;P,F) &\equiv alph(w) \subseteq P \wedge F \cap alph(w) = \emptyset \\
\varphi^c(w;P,F) &\equiv P \subseteq alph(w) \wedge F \not\subseteq alph(w) \\
\varphi^d(w;P,F) &\equiv alph(w) \subseteq P \wedge F \not\subseteq alph(w) \\
\varphi^e(w;P,F) &\equiv alph(w) \cap P \neq \emptyset \wedge F \cap alph(w) = \emptyset.
\end{aligned}$$

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 $\beta \in \{a,b,c,d,e\}$, we define:

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

An *evolutionary processor over V with random-context filters* 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 contexts of the processor.

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

An *evolutionary processor over V with membership filters* is a tuple (M,I,O), where M has the same meaning as above and I,O are subsets of V^*.

We denote by EP_V the set of evolutionary processors, no matter their filters, over V.

A *hybrid network of evolutionary processors with random-context filters* (HNEP(RCF) 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 with the set of vertices X_G and the set of edges E_G, each edge being given in the form of a set of two nodes. G is called the *underlying graph* of the network.
- $N: X_G \to EP_V$ is a mapping that associates with each node $x \in X_G$ the evolutionary processor with random-context filters $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $C_0: X_G \to V^*$ is a mapping that identifies the initial configuration of the network. It associates a finite set of words with each node of the graph G.
- $\alpha: X_G \to \{*,l,r\}$: $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $\beta: X_G \to \{a,b,c,d,e\}$ defines the type of the *input/output* filters of a node. More precisely, for every node $x \in X_G$ and a language $L \subseteq V^*$, the following filters are defined:

  input filter: $\rho_x(L) = \rho^{\beta(x)}(L;PI_x,FI_x)$,

  output filter: $\tau_x(L) = \rho^{\beta(x)}(L;PO_x,FO_x)$.

  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*. If the set of rules existing in any node contains at most one rule, then the network is said to be *elementary*. If all the sets defining the filters are empty (each node can be left and entered by any string), then the network is said to be *free*.

A *hybrid network of evolutionary processors with membership filters* (HNEP(MF) for short) is a 6-tuple $\Gamma=(V,G,N,C_0,\alpha,i_0)$, where:

- V and G are defined as above.
- $N:X_G\rightarrow EP_V$ is a mapping which associates with each node $x\in X_G$ the evolutionary processor with membership filters $N(x)=(M_x,I_x,O_x)$.
- C_0,α,i_0 are defined as above. For every node $x\in X_G$ and a language $L\subseteq V^*$, the following filters are defined:

  input filter: $\rho_x(L)=\{w\in L\cap I_x\}$,

  output filter: $\tau_x(L)=\{w\in L\cap O_x\}$.

In a similar way, we define the homogeneous and elementary HNEP(MF)s, respectively. We use the notation HNEP if the filter type does not matter.

In the theory of networks some types of underlying graphs are common, e.g., rings, stars, grids, etc. We shall discuss 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, 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 $C:X_G\rightarrow V^*$ which associates a set of strings with every node of the graph. A configuration may be understood as the set 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 $C\Rightarrow C'$, iff:

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

When changing by a communication step, each node processor $x\in X_G$ sends one copy of each string it has that 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:

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

Let Γ be an HNEP. A computation in Γ is a sequence of configurations $C_0,C_1,C_2,...$, where C_0 is the initial configuration of Γ, $C_{2i}\Rightarrow C_{2i+1}$ 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 $C_0,C_1,...$, 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

Clearly, HNEPs with membership filters defined by finite languages can generate regular languages only. If one allows membership filters defined by regular languages, we have:

Theorem 1 (Castellanos et al. submitted)

1. Each recursively enumerable language can be generated by a complete homogeneous HNEP(MF) of size 5.
2. Each recursively enumerable language can be generated by a star homogeneous HNEP(MF) of size 5.
3. Each recursively enumerable language can be generated by a ring homogeneous HNEP(MF) of size 6.

These results are not surprising, since similar characterizations have been obtained for other computing devices based on insertion and deletion operations (see, e.g., Csuhaj-Varjú & Mitrana 2000, Kari 1991, Kari et al. 1997, Kari & Thierrin 1996, Martín-Vide et al. 1998).

It is worth mentioning here that, unlike other parallel language generating devices, the above HNEP(MF)s generate 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 HNEP(MF) in at most $10n$ steps.

What is the generative power of smaller HNEP(MF)s? We do not have a complete answer. However, the following statements follow from the proof of Theorem 1 in (Castellanos et al. submitted).

Theorem 2

1. Each context-sensitive language can be generated by a complete or star homogeneous HNEP(MF) of size 4.
2. Each context-sensitive language can be generated by a ring homogeneous HNEP(MF) of size 5.
3. Complete and star homogeneous HNEP(MF)s of size 4 can generate non-recursive languages.
4. Ring homogeneous HNEP(MF)s of size 5 can generate non-recursive languages.

In what follows we discuss the generative power of HNEP(RCF)s. Let us denote by $|x|_a$ the number of all occurrences of a letter a in a string x. In [1] one proves that the language $L=\{x\in\{a,b,c\}^* \mid |x|_a=|x|_b=|x|_c\geq 1\}$ can be generated by a complete homogeneous HNEP(RCF) of size 6. The construction can be carried over star homogeneous HNEP(RCF)s. Therefore, we get:

Theorem 3 (Castellanos et al. submitted)

The families of regular and context-free languages are incomparable with the family of languages generated by complete or star homogeneous HNEP(RCF)s.

The next result obtained in (Martín-Vide et al. submitted) is rather surprising since the size of the HNEP(RCF), 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.

Theorem 4 (Martín-Vide et al. submitted)

Any regular language L over an alphabet with n symbols can be generated by a complete HNEP(RCF) of size $2n+3$.

Obviously, the HNEP(RCF) constructed accordingly to the proof of the above theorem in (Martín-Vide et al. submitted) which generate a given regular language L depends on the number of states of the finite automaton defining L as well, but the underlying graph remains the same for all regular languages over the alphabet of L. For instance, if the number of states of an automaton accepting L is m, then the total number of symbols in the alphabet of the constructed HNEP(RCF) is $2n+2nm+m$, while the total number of evolutionary rules is $2nm+2n$.

Since each linear grammar can be transformed into an equivalent linear grammar with rules of the form $A \to aB, A \to Ba, A \to \boldsymbol{e}$ only, the proof of the theorem from (Martín-Vide et al. submitted) can be adapted for linear grammars as well. Moreover, the statement remains valid for HNEP(RCF)s with other types of underlying structure.

Theorem 5

Any regular and linear language L over an alphabet with n symbols can be generated by a complete/star/ring HNEP(RCF) whose size depends linearly on n, only.

A natural problem arises: is it possible a similar characterization of recursively enumerable languages? Surprisingly enough, the answer is affirmative.

Theorem 6 (Csuhaj-Varjú et al. submitted)

Any recursively enumerable language over an alphabet V can be generated by a complete or star HNEP(RCF) of size $26+3 \cdot card(V)$.

This last result suggests the possibility of constructing a "universal" HNEP(RCF) with a fixed underlying structure for all recursively enumerable languages over a given alphabet.

The minimal size of a complete or star HNEP(RCF) generating an arbitrary recursively enumerable language over a fixed alphabet remains to be further investigated.

However, we can state:

Theorem 7 (Csuhaj-Varjú et al. submitted)

1. The language generated by any HNEP(RCF) of size one is regular.

2. There exist context-free non-regular (even non-linear) languages which can be generated by complete, free, homogeneous HNEP(RCF)s of size 2.
3. There exist context-sensitive non-context-free languages which can be generated by complete or ring homogeneous HNEP(RCF)s of size 4.
4. There exist non-recursive languages which can be generated by complete or star HNEP(RCF)s of size 28.
5. The family of languages generated by complete or star HNEP(RCF)s having no deletion node coincides with the family of context-sensitive languages.

This theorem raises a series of open problems:

1. Are there HNEP(RCF)s of size smaller than 4 able to generate non-context-free languages?
2. Is it true that HNEP(RCF)s of size two generate context-free languages only?
3. Which is the smallest HNEP(RCF) able to generate a non-context-sensitive language? And a non-recursive language?

Is it possible to generate all recursively enumerable languages with elementary HNEP(RCF)s? Again, the next result, proved in (Csuhaj-Varjú et al. submitted), seems interesting.

Theorem 8 (Csuhaj-Varjú et al. submitted)

Any recursively enumerable language can be generated by an elementary, complete or star HNEP(RCF).

4 Solving NP-Complete Problems

We survey in this section the NP-complete problems solved so far by HNEP(RCF)s. We start with one problem known to be NP-complete, namely the Bounded Post Correspondence Problem (BPCP) (Constable et al. 1974, Garey & Johnson 1979), which is a variant of a much celebrated computer science problem, the Post Correspondence Problem (PCP), known to be unsolvable Garey & Johnson 1979) in the unbounded case.

An instance of the PCP consists of an alphabet V and two lists of strings over V:

$u=(u_1,u_2,...,u_n)$, $v=(v_1,v_2,...,v_n)$.

The problem asks whether or not a sequence $i_1,i_2,...,i_k$ of positive integers exists, each between 1 and n, such that:

$u_{i1}u_{i2}...u_{ik}=v_{i1}v_{i2}...v_{ik}$.

The problem is undecidable when no upper bound is given for k, and NP-complete when k is bounded by a constant $K\leq n$. A DNA-based solution to the bounded PCP is proposed in (Kari et al. 2000).

Theorem 9 (Castellanos et al. 2001)

The bounded PCP can be solved by a complete HNEP in size and time linearly bounded by the product of K and the length of the longest string of the two Post lists.

The next problem is the so called 3-Colorability Problem. This problem consists of deciding 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 that are connected by an edge have the same color.

Theorem 10 (Castellanos et al. submitted)

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 in the input graph.

It is rather interesting that the underlying graph of the HNEP(RCF) proposed in (Castellanos et al. submitted) for solving this problem 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. Again, as in the case of language generating, the other parameters of the network depend on both numbers, of nodes and edges, but they still remain linearly bounded by such numbers. For instance, the total number of symbols is $7n+m+1$, while the total number of rules is $16m+3n+1$.

In the sequel, following (Head et al. 1999), we discuss the common descriptive format for three NP-complete problems called the Common Algorithmic Problem in the aforementioned work. 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 in U?

For detailed formulations and discussions about their solutions, the reader is referred to (Garey & Johnson 1979).

These problems can be viewed as special cases of the following algorithmic problem, called the Common Algorithmic Problem (CAP) in (Head et al. 1999). 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 containing 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.

Theorem 11 (Martín-Vide et al. submitted)

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

The price paid for homogeneousness is the following: the total number of symbols is $2n+4n+4$, but the total number of rules is rather high, that is $mn+6n+2+\sum_{i=1}^{m} card(F_i)$. The same problem can be solved in a more economic way with HNEP(RCF)s, namely:

Theorem 12

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

Now, the needed resources are: $m+3n+3$ symbols and $m+3n+1$ rules.

5 Conclusions and Further Work

We have surveyed a computational model whose underlying architecture is a complete graph having evolutionary processors placed in its nodes. Being a bio-inspired system, a natural question arises: how far is this model from the biological reality and engineering possibilities? More precisely, is it possible to exchange biological material between nodes? Can the input/output filter conditions of the node processors be biologically implemented? What about a technological implementation? We hope that at least some answers to these questions are affirmative.

Furthermore, one can take different directions of research. A deeper study of the free HNEP(RCF)s appears to be of interest since it is likely that these networks have better chances to get implemented. The description of the dynamics of evolving cell populations is an intriguing question which is in the focus of interest in current computer science. Results on the dynamics of cell (string) population in HNEPs might be a fruitful further direction of research. Last but not least, HNEPs in which other objects than strings are the mobile data navigating in the network seem to be promising. A first step in this direction was done in (Mitrana & Subramanian in progress), where these objects are pictures (two-dimensional strings).

BIBLIOGRAPHY

Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J. M. (2001) "Solving NPcomplete problems with networks of evolutionary processors" in J. Mira, A. Prieto (eds), *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence* (Proceedings of the Sixth International Work-Conference on Artificial and Natural Neural Networks (IWANN 2001), vol. I, Lecture Notes in Computer Science 2084), Berlin: Springer, 621–628.

Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J. M. (in press) "Networks of evolutionary processors". *Acta Informatica.*

Constable, R., Hunt, H., Sahni, S. (1974) "On the computational complexity of scheme equivalence", Technical Report No. 74-201, Department of Computer Science, Cornell University, Ithaca NY, 1974.

Csuhaj-Varj´u, E. (1997) "Networks of parallel language processors" in Gh. Pãun, A. Salomaa, (eds.) *New Trends in Formal Languages: Control, Cooperation, and Combinatorics*, Lecture Notes in Computer Science 1218, Berlin: Springer, 299–318.

Csuhaj-Varj´u, E., Dassow, J., Kelemen, J., Pãun, Gh. (1994) *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, London: Gordon and Breach.

Csuhaj-Varj´u, E., Martín-Vide, C., Mitrana, V. (submitted) "Hybrid networks of evolutionary processors: completeness results".

Csuhaj-Varj´u, E., Mitrana, V. (2000) "Evolutionary systems: a language generating device inspired by evolving communities of cells", *Acta Informatica* 36, 913–926.

Csuhaj-Varj´u, E., Salomaa, A. (to appear) "Networks of Watson-Crick D0L systems" in M. Ito, (ed.) *Proceedings of the 3rd International Colloquium on Words, Languages and Combinatorics*, Singapore: World Scientific.

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

Fahlman, S. E., Hinton, G. E., Seijnowski, T. J. (1983) "Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines" in *Proceedings of the AAAI National Conference on AI*, Los Altos CA: William Kaufman, 109–113.

Garey, M., Johnson, D. (1979) *Computers and Intractability. A Guide to the Theory of NP-Completeness*, San Francisco CA: W.H. Freeman.

Head, T., Yamamura, M., Gal, S. (1999) "Aqueous computing: writing on molecules" in *Proceedings of the Congress on Evolutionary Computation 1999*, Piscataway NJ: IEEE Service Center, 1006–1010.

Hillis, W. D. (1985) *The Connection Machine*, Cambridge MA: MIT Press.

Kari, L. (1991) *On Insertion and Deletion in Formal Languages*, Ph.D thesis, University of Turku, 1991.

Kari, L., Gloor, G., Yu, S. (2000) "Using DNA to solve the Bounded Correspondence Problem", *Theoretical Computer Science* 231, 193–203.

Kari, L., Pãun, Gh., Thierrin, G., Yu, S. (1997) "At the crossroads of DNA computing and formal languages: characterizing RE using insertion-deletion systems" in *Proceedings of the 3rd DIMACS Workshop on DNA Based Computing*, Philadelphia PA, 318–333.

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

Martín-Vide, C., Mitrana, V., Pérez-Jiménez, M. J., Sancho-Caparrini, F. (submitted) "Hybrid networks of evolutionary processors".

Martín-Vide, C., Pãun, Gh., Salomaa, A. (1998) "Characterizations of recursively enumerable languages by means of insertion grammars", *Theoretical Computer Science* 205/1-2, 195–205.

Mitrana, V., Subramanian, K. G., Ţâtãrˆam, M. (in press) "Networks of pictural processors". *Romanian Journal o Information Science and Technology*.

Sanko, D. [et al.] (1992) "Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome" in *Proceedings of the National Academy of Sciences of USA* 89, 6575–6579.

Carlos MARTÍN VIDE C. Martin-Vide is professor at Rovira i Virgili University (Tarragona), where he manages the Group of Research in Mathematical Linguistics. He works in theory of formal languages. He has published more than 200 scientific articles. He is Editor in Chief of *Grammars* (Kluwer) and Director of the International PhD School in Formal Languages and Applications.

Address: Research Group on Mathematical Linguistics, Rovira i Virgili University, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain. E-mail: cmv@correu.urv.es

Victor MITRANA is professor in Faculty of Mathematics at University of Bucharest and Ramón y Cajal Researcher at Rovira I Virgili University (Tarragona). He works in theory of formal languages, recombination and computation with DNA, bioinformatic and combinatory. He has published around 100 scientific articles.

Address: Faculty of Mathematics, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania. E-mail: mitrana@funinf.cs.unibuc.ro

C. MARTÍN VIDE and **V. MITRANA** are editors of the following books: *Where Mathematics, Computer Science, Linguistics and Biology Meet* (Kluwer, 2001), *Grammars and Automata for String Processing: From Mathematics and Computer Science to Biology, and Back* (Taylor and Francis, 2003) and *Formal Languages and Applications* (Physica, 2003, with G. Pãun)