Descriptional and Computational Aspects of Generating and Accepting Hybrid Networks of Evolutionary Processors

Carlos Martín-Vide[1] **Victor Mitrana**[1,2]

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

[2]Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 010014, Bucharest, Romania.

Abstract

The goal of this paper is to survey in a systematic and uniform way the main results regarding some descriptional and computational aspects of hybrid networks of evolutionary processors viewed both as generating and accepting devices, as well as solving problems with these mechanisms. We start by surveying some results regarding the size complexity of generating hybrid networks of evolutionary processors. Then, we define a computational complexity class of accepting hybrid networks of evolutionary processors and prove that this class equals the classical class **NP**. In another section, we present a few NP-complete problems and recall how they can be solved in linear time by accepting networks of evolutionary processors with linearly bounded resources (nodes, rules, symbols). We also recall a possible implementation of a solution for the 3-CNF-SAT using WWW. Finally we discuss some possible directions for further research.

1 Introduction

The origin of networks of evolutionary processors (NEPs for short) is twofold. In [7] we consider a computing model inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactical level. Cells are represented by words which encode their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of words, where each word represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on words. Only those cells are accepted as surviving (correct) ones which are represented by a word in a given set of words, called the genotype space of the species. This feature parallels with the natural process of evolution.

On the other hand, a well-known architecture for parallel and distributed symbolic processing, related to the Connection Machine [17] as well as the Logic Flow paradigm [10], 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 that data which can pass a filtering process can be communicated among the processors. 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., [11, 17].

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. Networks of language processors are closely related to grammar systems, more specifically to parallel communicating grammar systems [5]. 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. More recently, [8] introduces networks whose nodes are (standard) Watson-Crick D0L systems which communicate each other either the correct words or the corrected words.

In [1], we modify this concept in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, 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. Actually, many fitness functions on words may also be defined by random-context conditions. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [22]. Consequently, hybrid networks of evolutionary processors might be viewed as bio-inspired computing models.

Our mechanisms introduced in [1] are further considered in [2] as language generating devices and their computational power is investigated. Furthermore, filters, based on the membership and random-context conditions, used in [6] are generalized in some versions defined in [1, 2, 20]. More precisely, the new filters are based on different types of random-context conditions. In the aforementioned papers, the filters of all nodes are defined by the same random-context condition type. Moreover, the rules are applied in the same manner in all the nodes. These restrictions are discarded in [20] and [19]. By this reason, these networks were called *hybrid*.

In [19], we consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that **NP** equals the class of languages accepted by AHNEPs in polynomial time.

In a series of papers, we present *linear* time solutions to some NP-complete problems using generating hybrid networks of evolutionary processors (GHNEP). Such solutions are presented for the Bounded Post Correspondence Problem in [1], for the "3-colorability problem" in [2] (with simplified networks), and for the Common Algorithmic Problem in [20].

In [18] we propose two linear time solutions to two much celebrated NP-complete problems, namely the 3CNF-SAT and the HPP, based on AHNEPs having all resources (size, number of rules and symbols) linearly bounded by the size of the given instance. This work presents for the first time such solutions based on AHNEPs and not GHNEPs, and more important, by the definition of AHNEPs, one can evaluate the descriptional (number of nodes, rules, symbols) and computational (time) complexity of these AHNEPs with respect to their input word which is actually the given instance of the problem.

The reader can easily extend this approach to the solutions based on GHNEPs proposed in the aforementioned works.

We want to stress from the very beginning that we were not concerned with a possible biological implementation, though a matter of great importance. However, in the last section of [18] we discuss an (im)possible and a bit funny implementation, not of biological inspiration as one may expected according to the above considerations, but using WWW.

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 $\mathrm{card}(A)$. Any sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. The length of a word x is denoted by $|x|$ while the number of occurrences of a letter a in a word x is denoted by $|x|_a$. Furthermore, for each nonempty word x we denote by $alph(x)$ the minimal alphabet W such that $x \in W^*$. We denote by w^R the mirror image of the word w and by L^R the language of mirror images of all words in L. A morphism from $(V \cup U)^*$ to V^* which erases all symbols from U and leaves unchanged all symbols from V is called *projection* and it is denoted by pr_V.

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 is denoted by Sub_V, Del_V, and Ins_V, respectively.

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

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : w = ua\}, \\ \{w\}, \text{ otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : w = av\}, \\ \{w\}, \text{ otherwise} \end{cases}$$

- 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 a deletion or insertion 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 linguistic formulations of local DNA mutations.

For two disjoint subsets P and F of an alphabet V and a word over V, we define the predicates

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

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, the first condition requires that all permitting symbols are and no forbidding symbol is present in w, the second one requires that all symbols of w are permitting ones, while the last two conditions are weaker variants of the first one such that some forbidding symbols may appear in w but not all of them, and at least one permitting symbol appears in w, respectively.

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 contexts of the processor.

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

A *generating hybrid network of evolutionary processors* (a GHNEP, for short) is a 7-tuple $\Gamma = (V, G, \mathcal{N}, C_0, \alpha, \beta, x_O)$, where the following conditions hold:

- 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 is given in the form of a set of two nodes. G is called the *underlying graph* of the network.
- $\mathcal{N} : X_G \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $C_0 : X_G \longrightarrow V^*$ 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 occurring 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, $x \in X_G$, we define the following filters: the input filter is given as $\rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x)$, and the output filter is defined as $\tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x)$. That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.
- $x_O \in X_G$ is the *output node* of the GHNEP.

An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

- V and U are the input and network alphabet, respectively, $V \subseteq U$.
- G, N, α, β, x_O are defined as above, and x_I is the *input node* of the AHNEP.

In the above definitions, we say that card(X_G) is the size of Γ, denoted by $size(\Gamma)$. 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 at every node consists of at most one rule, then the network is said to be *elementary.*

Further, a network having all filters empty sets is said to be *free*. In the theory of networks some types of underlying graphs are common, e.g., rings, stars, grids, etc. In some of the aforementioned papers ([2, 9, 20, 19]), there were investigated networks of evolutionary processors having underlying graphs of these special forms, but with a special attention to complete graphs. Thus a GHNEP (AHNEP) is said to be a *star, ring, grid, or complete* GHNEP (AHNEP) if its underlying graph is a star, ring, grid, or complete graph, respectively. The star, ring, and complete graph with n nodes is denoted by S_n, R_n, and K_n, respectively. Most of the results presented in the sequel concern complete GHNEPs (AHNEPs).

A *configuration* of a GHNEP (AHNEP) Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words 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 word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words 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

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

Note that words which leave a node are eliminated from that node. If they cannot pass the input filter of any node, they are lost.

Let Γ be a GHNEP, a *computation* in Γ is a sequence of configurations C_0, C_1, C_2, ..., where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geq 0$. Note that the two steps, evolutionary and communication, are synchronized and they happen alternatively one after another. 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. 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 words existing in the output node at some step belong to the language generated by the network. Formally, the *language* generated by Γ is $L_{gen}(\Gamma) = \bigcup_{s \geq 0} C_s(x_O)$.

Let Γ be an AHNEP, the *computation of* Γ *on the input word* $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G$, $x \neq x_I$, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Again, the two steps, evolutionary and communication, are synchronized and they happen alternatively one after another. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. In other terms, each computation in an AHNEP is deterministic. A computation as above immediately halts if one of the following two conditions holds:

(i) There exists a configuration in which the set of words existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation*.

(ii) There exist two consecutive identical configurations.

In the aforementioned cases the computation is said to be finite. The language accepted by Γ is

$$L_{acc}(\Gamma) = \{w \in V^* \mid \text{ the computation of } \Gamma \text{ on } w \text{ is an accepting one}\}.$$

3 Descriptional Complexity: Size of Complete GHNEPs

Let L be a language generated by a complete GHNEP. We define

$$size(L) = \min\{size(\Gamma) \mid L = L_{gen}(\Gamma)\}.$$

For a regular language L we denote by $state(L)$ the minimal number of states of a finite automaton recognizing L. The first natural problem concerns the existence of a constant upper bound for the size of any language generated by a GHNEP. The next theorem shows that this is not the case.

Theorem 1 [3]. *The measure size is connected, that is for any $n \geq 1$ there exists a language L_n such that $size(L_n) = n$.*

Note that the languages L_n used in the above proof are defined over alphabets depending on n. Does the statement hold anymore for alphabets of a fixed size? If yes, which is this size?

The results presented in the first two lines of the next diagram are rather surprising since the size of the GHNEP generating a regular language L, hence its underlying structure, does not depend on $state(L)$. 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 words of the same length are generated simultaneously.

In the next diagram, L is a regular language, Γ is a GHNEP generating L, $symb(\Gamma)$ delivers the number of symbols used by Γ, and $rule(\Gamma)$ denotes the number of all rules in the nodes of Γ. The last column indicates the work where the result was proved.

$card(alph(L))$	$state(L)$	$size(\Gamma)$	$symb(\Gamma)$	$rule(\Gamma)$	Work
n	m	$2n+3$	$2n+2nm+m$	$\leq 2nm+2n+1$	[20]
n	m	$n+5$	$3n+(n+1)m$	$\leq 3n+(2n+1)m$	[3]
n	m	$2m+2$	$2m+n+m^2$	$\leq 2m^2+(n+1)m$	[21]

Table 1: Size complexity of complete GHNEPs generating regular languages

A natural problem arises: Given a regular language L, is $size(L)$ algorithmically computable? We are not able to give a complete answer to this problem. However, we can state:

Theorem 2 [3]. *Given a regular language L one can algorithmically decide whether or not $size(L) = 1$.*

However, a complete answer to the question: "Is the size of a regular language computable?" remains an open problem. The same problem is completely solved for context-free languages by showing that it is not decidable for context-free languages.

Theorem 3 [3]. *One cannot algorithmically decide whether the size of a context-free language equals 1.*

As an immediate consequence, we state

Corollary 1 [3]. *The measure size is not computable for the family of context-free languages.*

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, we can state:

Theorem 4 [3]. *Any regular and linear language L over an alphabet with n symbols can be generated by a complete/star/ring GHNEP whose size depends linearly on n, only.*

Another natural problem arises here: Is it possible to give a similar characterization of other families of languages in the Chomsky hierarchy? Surprisingly enough, the answer is affirmative even for the class of recursively enumerable languages.

Theorem 5 [9]. *Any recursively enumerable language L over an alphabet V can be generated by a complete or star GHNEP of size $28 + 3 \cdot \mathrm{card}(V)$. Hence, $size(L) \leq 28 + 3 \cdot \mathrm{card}(alph(L))$.*

This last result suggests the possibility of constructing a "universal" GHNEP with a fixed underlying structure for all recursively enumerable languages over a given alphabet. Furthermore, a similar result based on the proof of Theorem 5 from [9] can be proved for context-free languages too.

The minimal size of a complete or star GHNEP generating an arbitrary recursively enumerable language over a fixed alphabet remains to be further investigated. However, we can state:

Theorem 6 [9]. *1. The language generated by any GHNEP of size one is regular.*
2. There exist non-context-free languages which can be generated by complete, homogeneous GHNEPs of size 2.
3. There exist non-recursive languages which can be generated by complete or star GHNEPs of size 28.
4. The family of languages generated by complete or star GHNEPs having no deletion node coincides with the family of context-sensitive languages.

The smallest size of a non-context-free language is 2. Which is the smallest size of a non-context-sensitive language? The same question for non-recursive language remains open.

The next result, proved in [9], seems to be interesting. It raises a new series of open problems: which is the smallest size of an elementary GHNEP generating a non-recursive or non-context-free language? Is still the size of such a GHNEP generating a regular or recursively enumerable language linearly bounded by the alphabet size?

Theorem 7 [9]. *Any recursively enumerable language can be generated by an elementary, complete or star GHNEP.*

4 Computational Complexity of Complete AHNEPs

The reader is referred to [13, 14] for the classical time and space complexity classes defined on the standard computing model of Turing machine.

We define some computational complexity measures by using AHNEP as the computing model. To this aim we consider a AHNEP Γ and the language L accepted by Γ. The *time complexity* of the accepting computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the partial function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in L(\Gamma), |x| = n\}.$$

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Time}_{AHNEP}(f(n)) = \{L \mid \text{ there exists an AHNEP } \Gamma \text{ and } n_0 \text{ such that } L = L(\Gamma) \text{ and } \forall n \geq n_0 (Time_\Gamma(n) \leq f(n))\}$$

Moreover, we write $\mathbf{PTime}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Time}_{AHNEP}(n^k)$.

Now we prove a result which establishes a strong connection between the complexity classes defined on Turing machines and those defined on AHNEPs.

Proposition 1 [19]. *For any nondeterministic Turing machine, M, recognizing a language L there exists an AHNEP, Γ, accepting the same language L. Moreover, if M works within time $f(n)$ then $Time_\Gamma(n) \in O(f(n))$.*

It is worth mentioning that the underlying graph in the proof from [19] of this proposition is the complete graph K_p, with $p = 15+7(\text{card}(V_2)-1)+\text{card}(Q)+2(\text{card}(V_2)-1)^2+2\text{card}(Q)(\text{card}(V_2)-1)$. That is, the number of nodes is bounded by a quadratic function depending on the number of states and symbols of the simulated Turing machine. Also, the total number of symbols in this above simulation is bounded by a cubic function depending on the number of states and symbols of the Turing machine. More precisely, it is exactly

$$4\text{card}(Q)(\text{card}(V_2)-1)^2 + 2(\text{card}(V_2)-1)^2 + \text{Card}(V_2) + \text{card}(Q)(\text{card}(V_2)-1) + \\ 9(\text{card}(V_2)-1) + \text{card}(Q).$$

Based on this result, proved the main result of [19]:

Theorem 8 [19]. $\mathbf{NP} = \mathbf{PTime}_{AHNEP}$.

We can define another time complexity class by

$$\mathbf{ExpTime}_{AHNEP} = \bigcup_{k\geq 0} \mathbf{Time}_{AHNEP}(2^{n^k}).$$

Proposition 1 can now be extended, by the same proof, to

Proposition 2. $\mathbf{NEXPTIME} \subseteq \mathbf{ExpTime}_{AHNEP}$.

hence

Theorem 9. $\mathbf{NEXPTIME} = \mathbf{ExpTime}_{AHNEP}$.

5 Solving NP-complete Problems in Linear Time With Complete AHNEPs With Linearly Bounded Resources

We discuss very briefly and informally how AHNEPs could be used as problem solvers. A possible correspondence between decision problems and languages can be done via an encoding function which transforms an instance of a given decision problem into a word, see, e.g., [12]. We say that a decision problem is solved in linear time by AHNEPs if the following conditions are satisfied:

1. The encoding function can be computed by a deterministic Turing machine in linear time. Therefore each instance of the problem is linearly related to its associated word.

2. For each instance of the problem one can effectively construct an AHNEP which decides in linear time the word encoding the given instance. This means that the word is accepted if and only if the solution to the given instance of the problem is "YES". This effective construction is called a linear time solution to the considered problem.

As we shall see in the sequel, some well-known NP-complete problems can be solved in linear time.

- The *Bounded Post Correspondence Problem* (BPCP)[4, 12]: is a variant of a much celebrated computer science problem, the Post Correspondence Problem (PCP) known to be unsolvable [12] in the unbounded case. An instance of the BPCP consists of an alphabet V, two lists of words over V $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$, and $K \leq n$. The problem asks whether or not a sequence $i_1, i_2, \ldots, i_k$, $k \leq K$, of positive integers exists, each between 1 and n, such that $u_{i_1}u_{i_2}\ldots u_{i_k} = v_{i_1}v_{i_2}\ldots v_{i_k}$.

- The *3-colorability problem* is to decide whether each vertex in an undirected graph can be colored by using three colors (say red, blue, and green) in such a way that after coloring, no two vertices which are connected by an edge have the same color.
- The *Common Algorithmic Problem* (CAP): 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. Let n and m be the cardinality of S and F, respectively. This problem might be viewed as a descriptive format for three NP-complete problems (maximum independence set problem, vertex cover problem, satisfiability problem) as shown in [16].
- The *Hamiltonian Path Problem* (HPP) is to decide whether or not a given directed graph has a Hamiltonian path. A Hamiltonian path in a directed graph is a path which contains all vertices exactly once.

The next table presents the complexity of AHNEPs solving the above NP-complete problems in linear time.

Problem	Time	Size	Other resources	Work
$BPCP$	$O(K)$	$O(K)$	$O(K)$	[1]
3-colorability	$O(n+m)$	$O(m)$	$O(m+n)$	[2]
CAP	$O(m+n)$	$O(m+n)$	$O(m+n)$	[20]
HPP	$O(n)$	$O(n)$	$O(n)$	[18]

Table 2: Complexity of AHNEPs solving NP-complete problems

We also can say that the networks solving the aforementioned problems, excepting the input and output nodes, may be viewed as "programs".

6 Directions for Further Research

We list here several possible directions for further research; clearly, the reader may identify others some of them being more interesting. However, these appear natural and attractive to us and we consider that they deserve a deep investigation.

– The first direction we suggest is to consider similar questions (size and time complexity) with respect to free GHNEPs/AHNEPs. On the other hand, the computational power of these extremely simple mechanisms seems difficult to be settled.

– Almost all results surveyed here concern complete GHNEPs/AHNEPs. Which of them remain valid for networks with another underlying structure?

– A natural idea is to replace the evolutionary operations by another very common operation in the area of DNA computing, that of splicing. In 1987, T. Head introduced the *splicing* operation as a language theoretical approach of the recombinant behavior of DNA under the influence of restriction enzymes and ligases [15]. Roughly speaking, the main idea of the splicing operation is that two sequences are cut at specified sites, and the first substring of one sequence is pasted to the second segment of the other and vice versa. The whole research program accomplished for GHNEPs/AHNEPs could be initiated for GHNEPs/AHNEPs with the evolutionary operations replaced by splicing.

– Another natural and possible fruitful idea is to consider *space* complexity classes defined on AHNEPs. At least two definitions might be considered: the length of the longest word navigating in the network during a computation, the greatest number of different words existing in the networks at some step during a computation.

– The problem of finding the class of all NP problems that can be solved in linear time by AHNEPs with linearly bounded resources seems to be quite attractive.

References

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

[2] Castellanos, J., Martín-Vide, C., Mitrana, V., & Sempere, J., Networks of evolutionary processors, *Acta Informatica*, **39** (2003), 517-529.

[3] Castellanos, J., Leupold, P., & Mitrana, V., On the size complexity of hybrid networks of evolutionary processors, *Theoret. Comput. Sci.*, in press.

[4] Constable, R., Hunt, H., & Sahni, S., On the computational complexity of scheme equivalence, *Technical Report* No. 74-201, Dept. of Computer Science, Cornell University, Ithaca, NY, 1974.

[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* (Gh. Păun, A. Salomaa, eds.), LNCS 1218, Springer Verlag, 1997, 299–318.

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

[8] Csuhaj-Varjú, E., & Salomaa, A., Networks of Watson-Crick D0L systems. In: *Proc. International Conference Words, Languages & Combinatorics III* (M. Ito, T. Imaoka, eds.), World Scientific, Singapore, 2003, 134–150.

[9] Csuhaj-Varjú, E., Martín-Vide, C., & Mitrana, V., Hybrid networks of evolutionary processors: Completeness results, submitted.

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

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

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

[13] Hartmanis, J., Lewis II, P.M., & Stearns, R.E., Hierarchies of memory limited computations. In: *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, 1965, 179–190.

[14] Hartmanis, J., & Stearns, R.E., On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 533–546.

[15] Head, T., Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, *Bull. Math. Biology* **49** (1987), 737–759.

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

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

[18] Manea, F., Martín-Vide, V., & Mitrana, V., Solving 3CNF-SAT and HPP in linear time using WWW, *Proc. of MCU 2004*, LNCS, in press.

[19] Margenstern, M., Mitrana, V., & Perez-Jimenez, M., Accepting hybrid networks of evolutionary processors. In: *Proc. of DNA 10*, LNCS, in press.

[20] Martín-Vide, C., Mitrana, V., Perez-Jimenez, M., & Sancho-Caparrini, F., Hybrid networks of evolutionary processors. In: *Proc. of GECCO 2003*, LNCS 2723, Springer Verlag, Berlin, 2003, 401–412.

[21] Martín-Vide, C., Mitrana, V., Networks of evolutionary processors: Results and perspectives. In: *Molecular Computational Models: Unconventional Approaches*, Idea Group Publishing, Hershey, 2004, 41 pp.

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