Computing and Informatics, Vol. 34, 2015, 858–876

OPTIMAL PERFORMANCE: UNDERLYING OCTAHEDRON GRAPH OF EVOLUTIONARY PROCESSORS

Luis Fernando de Mingo López, Nuria Gómez Blas

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos
Universidad Politécnica de Madrid
Crta. de Valencia, km. 7
28031 Madrid, Spain
e-mail: {fernando.demingo, nuria.gomez.blas}@upm.es

Alberto Arteta

Computer Science Department
Troy University
112C Wright Hall 36082 Alabama, USA
e-mail: aarteta@troy.edu

Abstract. Networks of evolutionary processors with an underlying octahedron graph consist of 7 language processors which are linked to the vertices of the octahedron graph. Notice that they are located in the 6 facets and the core of a cube graph. Also note that the nodes are only able to perform a type of mutation based on the words found in that node. Each node is associated with an input filter and an output filter, defined by some regular language. Rules are applied to all the words existing in every node. The words, able to pass the output filter of the respective node, are sent out and they navigate through the graph. Such words will enter those nodes provided their input filters are satisfied. The computational power of the network is comparable to Turing machines when the filters are regular languages. We introduce several variants of octahedron networks, depending on rule types and the way of computation plus their computational power. Some known problems are addressed at the end.

Keywords: Networks of evolutionary processors, natural computing, evolutionary models, P-systems

1 INTRODUCTION

Networks of evolutionary processor is a mechanism inspired by cell biology. These networks have nodes which are very simple processors able to perform just one type of point mutation (insertion, deletion or substitution of a symbol) [9]. These nodes are endowed with filters which are defined by some membership or random context condition. This work is a continuation of the investigation started in [7] and [8].

The computational process described here is not exactly an evolutionary process in the Darwinian sense. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we refer to a processor able to perform very simple rewriting operations, such as insertion, deletion or substitution of symbols within a given word. Generally speaking, each node could be considered as a cell that has genetic information encoded in *DNA* sequences. This sequence can evolve by local evolutionary events, which is to say point mutations. Each node is specialized in just one of these evolutionary operations.

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 [20]. Furthermore, we were not concerned here with a possible biological implementation, though a matter of great importance.

These networks may be used as language generating devices or as computational ones. Here, we consider them as computational mechanisms and show how an NP-complete problem can be solved in linear time. The model is similar to P systems, a new computing model inspired by the hierarchical and modularized cell structure proposed in [17, 18].

Networks of evolutionary processors [7] are language generating devices [8], if we look at the strings collected in the output node. We can also look at them as doing some computation. If we consider these networks with nodes having filters defined by random context conditions, which seems to be closer to the recent possibilities of biological implementation, we can use these simple mechanisms to solve NP-complete problems in linear time. Such solutions are presented for the Bounded Post Correspondence Problem in [6], for the 3-Colorability Problem in [7] and for the Common Algorithmic Problem in [15]. As a further step, in [15] the so-called hybrid networks of evolutionary processors are also considered. Here, deletion nodes or insertion nodes could have a different working mode (performs the operation at any position, in the left-hand end or in the right-hand end of the word) and different nodes are allowed to use different ways of filtering. Thus, the same network may have nodes where the deletion operation can be performed at arbitrary position and nodes where the deletion can be done only at right-end of the word.

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 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.

In the paper we present some results regarding a network of evolutionary processor based on an octahedron graph from the point of view of their computational power. This octahedron organization is suitable for a hardware implementation, similar to those carried out with transition *P systems* [4, 5], since such cubes can be connected surface-to-surface to obtain a complex network in a modular way.

Why not use a triangular graph instead of an octahedron graph? The problem yields in a hardware constraint that is – a triangular graph has 5 processors (4 facets and the core) therefore 3 bits must be used to address any processor. But 3 bits can address 8 processors. For this reason an octahedron graph is chosen, this way any processor can be addressed in its hardware implementation (the 8^{th} bit combination is used to address the whole system).

2 PRELIMINARIES

Formally, a network of evolutionary processors [6, 7] of size n is a construct $\Gamma = (V, N_1, N_2, \ldots, N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n$, $N_i = (M_i, A_i, PI_i, PO_i)$ is the i^{th} evolutionary node processor of the network. The parameters of every processor are:

- M_i is a finite set of evolution rules of one of the following forms only
  - $a \to b$, $a, b \in V$ (substitution rules)
  - $a \to \epsilon$, $a \in V$ (deletion rules)
  - $\epsilon \to a$, $a \in V$ (insertion rules)

  More clearly, the set of evolution rules of any processor contains either substitution or deletion or insertion rules.
- A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i^{th} node. Actually, in what follows, we consider that each string appearing in any node at any step has an arbitrarily large number of copies in that node, so that we shall identify multisets by their supports.
- PI_i and PO_i are subsets of V^* representing the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i (w \in PO_i)$.

Finally, $G = (\{N_1, N_2, \ldots, N_n\}, E)$ is an undirected graph called the underlying graph of the network. The edges of G, that is the elements of E, are given in the form of sets of two nodes. The complete graph with n vertices is denoted by K_n. By a configuration (state) of an *NEP* as above we mean an n-tuple $C = (L_1, L_2, \ldots, L_n)$, with $L_i \subseteq V^*$ for all $1 \leq i \leq n$. A configuration represents the sets of strings (remember that each string appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$.

A configuration can change either by an evolutionary step or by a communicating step [6, 7]. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i. When changing by a communication step, each node processor N_i sends all copies of the strings it has which are able to pass its output filter to all the node processors connected to N_i and receives all copies of the strings sent by any node processor connected with N_i providing that they can pass its input filter.

Theorem 1. Each recursively enumerable language can be generated by a complete NEP of size 5. [6]

Theorem 2. Each recursively enumerable language can be generated by a star NEP of size 5. [6]

Theorem 3. The bounded PCP can be solved by an NEP in size and time linearly bounded by the product of K and the length of the longest string of the two Post lists. [7]

2.1 Simple Networks of Evolutionary Processors

A simple *NEP* [6] of size n is a construct $\Gamma = (V, N_1, N_2, \ldots, N_n, G)$, where, V and G have the same interpretation as for *NEPs*, and for each $1 \leq i \leq n, N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i)$ is the i^{th} evolutionary node processor of the network. M_i and A_i from above have the same interpretation as for an evolutionary node in a *NEP*, but

- PI_i and FI_i are subsets of V representing the input filter. This filter, as well as the output filter, is defined by random context conditions, PI_i forms the permitting context condition and FI_i forms the forbidding context condition. A string $w \in V^*$ can pass the input filter of the node processor i, if w contains each element of PI_i but no element of FI_i. Note that any of the random context conditions may be empty, in this case the corresponding context check is omitted. We write $\rho_i(w) = true$, if w can pass the input filter of the node processor i and $\rho_i(w) = false$, otherwise.
- PO_i and FO_i are subsets of V representing the output filter. Analogously, a string can pass the output filter of a node processor if it satisfies the random context conditions associated with that node. Similarly, we write $\tau_i(w) = true$, if w can pass the input filter of the node processor i and $\tau_i(w) = false$, otherwise.

Theorem 4. The families of regular and context-free languages are incomparable with the family of languages generated by simple NEPs. [6]

Theorem 5. 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 of the input graph. [6]

3 OCTAHEDRON NETWORK OF EVOLUTIONARY PROCESSORS

Let V be an alphabet over a set of symbols. A string x of length m over an alphabet V is defined as the sequence of symbols $a_1 a_2 \dots a_m$ where $a_i \in V$ for all $1 \leq i \leq m$. The set of all strings over V is denoted by V^* and the empty string is denoted by ϵ.

The definition of network of evolutionary processors with an underlying octahedron graph is a restricted variant of previously studied hybrid networks of evolutionary processors (see e.g. [13, 15]), with a special graph structure and without forbidden filters.

Formally, a network of evolutionary processors with an underlying octahedron graph (*NEP[OCT]* for short) is a construct $\Sigma = \{V, N_0, N_1, N_2, N_3, N_4, N_5, N_6\}$, where V is an alphabet and processors N_i are connected within a graph in the following way:

- N_0 is connected to all other processors.
- Each processor $\{N_i | 1 \leq i \leq 6\}$ is connected to all processors $\{N_j | j \neq i \wedge 1 \leq j \leq 6\}$ except the opposite processor N_k where $k = i + (2(i \bmod 2) - 1)$. Therefore, each of these processors is also connected to other 5 processors.

The geometrical structure of the underlying graph that connects processors is a cube as shown in Figure 1. Processor N_0 is the cube core that collects desired results of computation. Each processor has connections with another 5 processors and therefore, the underlying graph has 7 vertices and 18 edges.

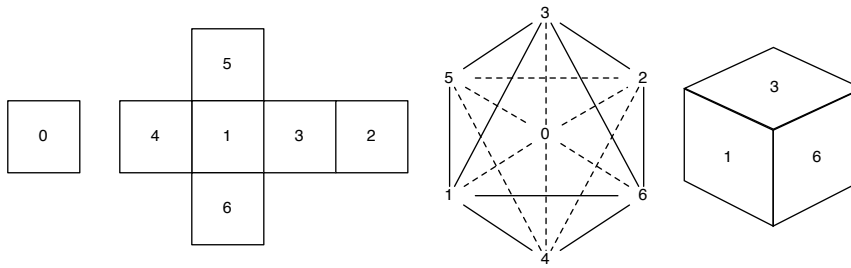


Figure 1. Geometrical structure of an octahedron network of evolutionary processors

A processor $N_i = \{M_i, A_i, PI_i, PO_i\}$ is the i^{th} evolutionary processor of the network. The parameters of every processor are:

- M_i is a finite set of evolution rules of one of the following forms only:
  - $a \to b$, $a, b \in V$ (substitution rules, Sub_V),
  - $a \to \epsilon$, $a \in V$ (deletion rules, Del_V),
  - $\epsilon \to a$, $a \in V$ (insertion rules, Ins_V),

  Given a rule as above σ and a string $w \in V$, [15] defines 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 $\sigma \equiv a \to \epsilon \in Del_V$, 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 $\sigma \equiv \epsilon \to a \in Ins_V$, then

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

Simply put, the set of evolution rules of any processor contains either substitution or deletion or insertion rules. Context information can be controlled by using the operation mode $\alpha \in \{*, l, r\}$, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. This way, context rules [11] and directional context rules [12] are built using the α mode of processors.

- A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i^{th} node. We consider that each string appearing in a node of the net at any step has an arbitrarily large number of copies in that node, so that we shall identify multisets by their supports.
- PI_i and PO_i are subsets of V^* representing the input and output filter respectively. These filters are defined by membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $\forall x \in PI_i, w = axb$ where $a, b \in V^*$ ($\forall x \in PO_i, w = axb$ where $a, b \in V^*$).

  In other words, given a subset P of an alphabet V^* and a word w over V, we define the predicate:

  – $\varphi(w; P) \equiv P \subseteq alph(w)$

  The construction of this predicate is based on random-context conditions defined by the set P (permitting context). For every language $L \subseteq V$ we define $\varphi(L; P) = \{w \in L | \varphi(w; P)\}$.

  For a given processor i, we define:

- $\rho_i(.) = \varphi(.; PI_i)$
- $\tau_i(.) = \varphi(.; PO_i)$

That is, $\rho_i(w)$ (or τ_x) indicates whether or not the string w can pass the input (or output) filter of processor i. Simply put, $\rho_i(L)$ (or $\tau_i(L)$) is the set of strings of L that can pass the input (or output) filter of i.

We write $\rho_i(w) = true$, if w can pass the input filter of the node processor i and $\rho_i(w) = false$, otherwise. We write $\tau_i(w) = true$, if w can pass the output filter of the node processor i and $\tau_i(w) = false$, otherwise.

$PI_i \subseteq V^*$ is the input permitting context of the processor i, while $PO_i \subseteq V^*$ is the output permitting context of the processor i [15].

By a configuration of a NEP[OCT] as seen above we we refer to a n-tuple:

$$C = (L_0, L_1, \ldots, L_n)$$

with $L_i \subseteq V^*$ for all $0 \leq i \leq n$. A configuration represents the sets of strings (please note that each string appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$.

4 DYNAMICS OF OCTAHEDRON NETWORK OF EVOLUTIONARY PROCESSORS

A configuration can change either by an evolutionary step or by a communicating step. Computation steps can be defined in a controlled way, that is, first through an evolutionary step and then a communicating step, or in a parallel nondeterministic way (when evolution and communication take place at the same time).

When a new evolutionary step occurs, each component L_i of the configuration changes in accordance with the evolutionary rules associated with node i. Formally, we say that configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes to configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$. This is caused by the following evolutionary step:

$$C_1 \Rightarrow C_2$$

if L'_i is the set of strings obtained by applying the rules of R_i to the strings in L_i as follows:

- If the same substitution rule replaces different occurrences of the same symbol within a string, all these occurrences are replaced within different copies of the string. The resulting multiset contains that string in an arbitrary large number of copies.
- Unlike their common use, deletion and insertion rules are applied only to the end of the string. Thus, a deletion rule $a \to \epsilon$ can be applied only to a string

which ends by a, (in other words wa), leading to the string w, and an insertion rule $\epsilon \to a$ applied to a string x consists of adding the symbol a to the end of x, obtaining xa. This is controlled using the operation mode $\alpha \in \{*, l, r\}$ of the whole network. Here, we do not consider an operation mode for every rule as [15] does, all rules have the same operation mode.

- If more than one rule (regardless of its type) applies to a string, then all the rules are applied to different copies of such string.

By definition, if L_i is empty for some $0 \leq i \leq n$, then L'_i is empty as well.

When changing by a communication step, each node processor sends all copies of the strings to all connected node processors and receives all copies of the strings sent by any connected node processor, provides that they can pass its input filter.

Formally, we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes to the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by the communication step written below.

$$C_1 \vdash C_2$$

if for every $0 \leq i \leq n$,

$$L'_i = L_i \setminus \{w \in L_i | \tau_i(w) = true\} \cup \bigcup_{j=0, j \neq i}^{n} \{x \in L_j | \tau_j(x) = true \wedge \rho_i(x) = true\}$$

A parallel non-deterministic computation step among two configurations C_1 and C_2, is represented by $C_1 \models C_2$.

$$C_1 \models C_2 = (C_1 \vdash C_2) \wedge (C_1 \Rightarrow C_2)$$

Let $\Gamma = (V, N_1, N_2, \ldots, N_n)$ be an *NEP[OCT]*. By a parallel non-deterministic computation in Γ we refer to a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration and $C_i \models C_{i+1}$ for all $i \geq 0$.

If the sequence is finite, we have a finite computation. The result of any finite computation is collected in a particular node called the output node (the core cube). If one considers the output node of the network as being the node N_0, and if $C_0, C_1, \ldots, C_t$ is a computation, then the set of strings existing in the node N_0 at the last step – the 0^{th} component of C_t – is the result of this computation. The time complexity of the above computation is the number of steps, which is represented by t.

Under the present definition, if the evolution and communication do not necessarily alternate, the system becomes non-deterministic in a random way, and the time complexity considerations become more difficult to prove. That is the reason a controlled computation is needed.

4.1 Controlled Computation

Let $\Gamma = (V, N_1, N_2, \ldots, N_n)$ be an octahedron network of evolutionary processors. By a controlled computation in Γ we refer to a sequence of configurations

$C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration and $C_{2i} \Rightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$.

Controlled computation can be achieved using a *parallel non-deterministic* computation by modifying the rules in R_i and filter PO_i of processor p in the following way:

- each rule $r_k \in R_i$, where $k = 1, \ldots, card(R_i)$, the symbol $X_k^{(p)}$ must be added to the string w
- output filter PO_i' contains all the symbols $X_k^{(p)}$, which is to say, $PO_i' = PO_i \cup \left\{X_1^{(p)}, X_2^{(p)}, \ldots, X_j^{(p)}\right\}$ where $1 \leq j \leq k$

and adding the following rules to processor p:

- $X_k^{(l)} \rightarrow \epsilon$, with $l = 1, \ldots, n$ and $l \neq p$.

In this way a parallel non-deterministic computation becomes a controlled computation as $\tau(w) = false$. This occurs until all rules are applied.

	$\alpha = r$	$\alpha = l$	$\alpha = *$
Controlled computation	$NEP[OCT]_{cr}$	$NEP[OCT]_{cl}$	$NEP[OCT]_{c*}$
Parallel computation	$NEP[OCT]_{pr}$	$NEP[OCT]_{pl}$	$NEP[OCT]_{p*}$

Table 1. Different kinds of octahedron networks depend on the following rules and dynamics: parallel non-deterministic or controlled

In short, many kinds of octahedron networks can be defined depending on the rules operational mode(α) and dynamics (controlled and parallel), see Table 1. The next section shows that all these families are computationally complete. By referring to their computational power it is clear that:

- $NEP[OCT]_{c\alpha} \subseteq NEP[OCT]_{p\alpha}$
- $NEP[OCT]_{Xr} \subseteq NEP[OCT]_{X*}$
- $NEP[OCT]_{Xl} \subseteq NEP[OCT]_{X*}$

where p means parallel non-deterministic computation and c means controlled computation.

Theorem 6. Problems solved using a $NEP[OCT]_{c\alpha}$ can be solved using a network $NEP[OCT]_{p\alpha}$, where $\alpha \in \{*, r, l\}$.

Proof. Given a processor $N_i = \{A_i, R_i, PI_i, PO_i\}$ belonging to a $NEP[OCT]_{c\alpha}$ it is possible to convert it into a processor $N_i' = \{A_i, R_i', PI_i, PO_i'\}$ which behaves in a same way within a $NEP[OCT]_{p\alpha}$.

This is described in the following way:

- given a rule $r_k \in R_i$, where $k = 1, \ldots, card(R_i)$, with the notation $A \rightarrow B$, with $1 \leq k \leq p$, each rule $r_k' \in R_i'$ has the form $A \rightarrow BX_{ik}$

- given the output filter PO_i, $PO'_i = PO_i \bigcup_{k=1}^{p} X_{ip}$
- the following rules must be added to processor N'_i:

$$X_{lm} \to \epsilon, \text{ with } l = 1, \ldots, n, l \neq i \text{ and } m = 1, \ldots, card(R_l).$$

With these new sets, R'_i and PO'_i, the parallel non-deterministic computation of an octahedron network of evolutionary processors behaves in the same way as a controlled computation, since $\tau(w) = false$ until all rules are applied. □

Theorem 7. Problems solved by using a $NEP[OCT]_{Xr}$ can also be solved by using a $NEP[OCT]_{X*}$ and problems solved by using a $NEP[OCT]_{Xl}$ can be solved by using a $NEP[OCT]_{X*}$, where $X \in \{p, c\}$.

Proof. It is obvious that $abc \to adc$ where $a, (c = \epsilon) \in V^*$ is equal to $ab \to ad$ and $ab \to ad$ where $(a = \epsilon) \in V^*$ is equal to $b \to d$. □

5 COMPUTATIONAL COMPLETENESS

As [2, 3] stated, networks of evolutionary processors with two nodes are unpredictable. They are able to accept or generate all recursively enumerable languages. Since the structure of octahedron networks of evolutionary processors contains a complete subgraph on three nodes, the presented finding results are a consequence of results from [2, 3]. It shows that 3 nodes can generate any recursively enumerable language with a simple process of encoding. Nevertheless, 4 nodes are enough to generate all recursively enumerable languages (the graph structure of octahedron networks also contains a complete subgraph with 4 nodes).

As octahedron networks with finite filters can generate regular languages only, we shall consider having finite regular languages as filters within octahedron networks (as filters are subsets of V, they must be finite by definition).

Theorem 8. Each recursively enumerable language can be generated by a network $NEP[OCT]_{cr}$.

Proof. Let $G = (N, T, S, P)$ be an arbitrary phrase-structure grammar in the *Kuroda* normal form [10], namely P contains only rules in the following forms:

$$A \to a, A \to BC, AB \to CD, A \to \epsilon$$

where A, B, C, D are nonterminals and a is a terminal. We assume that the rules $A \to BC$ and $AB \to CD$ of P are labelled in a one-to-one manner by the labels $r_1, r_2, \ldots, r_n$. We shall refer to the rules as $A \to a, A \to \epsilon, A \to BC$, and $AB \to CD$ as rules of type $0, 1, 2$, and 3, respectively. We construct the following $NEP[OCT]_{cr}$:

$$\Gamma = (V, N_0, N_1, N_2, N_3, N_4, N_5, N_6).$$

where

$$
\begin{aligned}
V &= N \cup T \cup V' \cup \{X\},\\
V' &= \{r_i, p_i, q_i, s_i, t_i | 1 \le i \le 5\}
\end{aligned}
$$

and

$$
\begin{aligned}
N_0 &= (\emptyset, \emptyset, T^*, (N \cup T \cup V' \cup \{X\})^*(N \cup V' \cup \{X\})(N \cup T \cup V' \cup \{X\})^*),\\
N_1 &= (M_1, \{S\}, (N \cup T)^* \cup (N \cup T)^*\{r_i q_i | 1 \le i \le 5\}(N \cup T)^*,\\
&\quad (N \cup T)^*(\{r_i, s_i p_i, t_i q_i | 1 \le i \le 5\} \cup \{X\})(N \cup T)^* \cup T^*)
\end{aligned}
$$

with

$$
\begin{aligned}
M_1 &= \{A \to X | A \to \epsilon \in P\} \cup \{A \to a | A \to a \in P\} \cup \{A \to r_i,\\
&\quad r_i \to t_i | r_i : A \to BC \in P\} \cup \{A \to s_i, B \to p_i | r_i : AB \to CD \in P\},
\end{aligned}
$$

$$
\begin{aligned}
N_2 &= (\{\epsilon \to q_i | r_i : A \to BC\}, \emptyset, (N \cup T)^*\{r_i | 1 \le i \le 5\}(N \cup T)^*\\
&\quad (N \cup T \cup V')^*),\\
N_3 &= (M_3, \emptyset, (N \cup T)^*\{s_i p_i, t_i q_i | 1 \le i \le 5\}(N \cup T)^*, (N \cup T)^*)
\end{aligned}
$$

with

$$
\begin{aligned}
M_3 &= \{t_i \to B, q_i \to C | r_i : A \to BC \in P\} \cup\\
&\quad \{s_i \to C, p_i \to D | r_i : AB \to CD \in P\},
\end{aligned}
$$

$$
\begin{aligned}
N_4 &= (\{X \to \epsilon\}, \emptyset, (N \cup T)^*\{X\}(N \cup T)^*, (N \cup T)^*),\\
N_5 &= (\emptyset, \emptyset, \emptyset, \emptyset),\\
N_6 &= (\emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}
$$

Note that filters PI and PO are subsets of V^* by definition.

There is a more thorough explanation in [7]; the only difference is that such proof is based on a complete graph with size 5. We stress that nodes N_5 and N_6 are empty, in which case they only transmit information to other processors. In this way they behave as a bridge among two given processors. Due to this reason the cube can be considered a complete graph with size 5 and therefore same proof as [7] can be applied.

Initially, there are arbitrarily many copies of the string S in the node N_1. By extrapolation, we may assume that a multiset of strings containing at least one nonterminal and no symbol from $V \cup X$, each string appearing in an unbounded number of copies, is in N_1 at a given moment, before an evolutionary step. If there is one string x in this multiset having at least one occurrence of a nonterminal A and $A \to Y$ is an evolution rule in M_1, then the first occurrence of A in x is replaced by Y in an infinite number of copies of x, the second occurrence of A in x is replaced by Y in an infinite number of copies of x, and so on for all the occurrences of A in x. This process applies to all strings in N_1 and to all evolution (substitution) rules in M_1.

Those strings obtained by an application of a rule $A \to Y$, $Y \in T$ cannot leave N_1 in the next step, which is a communication step, since they cannot pass its output filter. In this way, the network simulates all the possible one step applications of the rules of type 0 in two steps.

Those strings obtained by an application of a rule $A \to r_i$, for some i, (this means that there is a rule $r_i : A \to BC$ in P) are sent out and received by either N_2 or N_4 which are the only nodes able to receive them. More precisely, those strings containing X are received by N_4 and the others by N_2. In N_4, the symbol X is removed and the obtained strings are sent back to either N_1, provided that they still contain nonterminals, or N_0, if they are terminal strings. Thus, all possible one step applications of the rules of type 1 are done in four steps. In N_2, q_j is inserted to any position in the existing strings in the same way as that discussed above for the rule $A \to Y$, for all j such that r_j is a rule of type 2. Only those strings having an occurrence of r_i which received an adjacent symbol q_i in the righthand side of r_i can leave N_2, all the others remaining in N_2 forever. The strings leaving N_2 cannot be received by any node other than N_1 where the only useful evolution rules which can be applied to them are those of the form $r_i \to t_i$. After applying these rules, the new strings are sent out again. All the other rules applied to the strings just received by lead to strings obliged to remain forever in N_0. The strings sent out are received by N_3 where t_i and q_i are replaced by B and C, respectively, provided that the righthand side of the rule r_i is BC. In this way, the network simulates all the possible one step applications of the rules of type 2 in ten steps.

In a similar way, the network simulates all the possible one step applications of the rules of type 3 in eight steps. More precisely, if one wants to apply the rule $r_i : AB \to CD$, then in N_1, in an evolutionary step, an occurrence of A is replaced by s_i, these new strings remain in N_1 during the next communication step since they cannot pass the output filter of N_1, then in the next evolutionary step an occurrence of B is replaced by p_i, but only those strings containing the subword $s_i p_i$ (this means that a subword AB was replaced by $s_i p_i$) can pass the output filter of N_1, the others remaining in N forever.

In short, strings created from the alphabet $N \cup T$ always return N_1. The overall process applies to all strings containing nonterminals while the terminal strings are sent to N_0 where they stay forever. According to those results, we can state for certain that the language generated by Γ within the output node N_0 is exactly $L(G)$. □

Theorem 9. Each recursively enumerable language can be generated by an octahedron network of evolutionary processors $NEP[OCT]_{X\alpha}$, $X \in \{c, p\}$ and $\alpha \in \{*, r, l\}$.

Proof. According to Theorems 6 and 7, $NEP[OCT]_{cr}$ is the smallest subset of $NEP[OCT]$ variants. By applying Theorem 8 the proof is obvious. □

Furthermore, unlike other parallel language generating devices, a $NEP[OCT]_{cr}$ generates a language in a very efficient way. Namely all strings that can be generated

by a grammar in n steps are generated together by a $NEP[OCT]_{cr}$ in, at most, $10n$ steps.

6 SOLUTION TO 3-COLORABILITY PROBLEM

Despite their simplicity, networks of evolutionary processors might be used for solving NP-complete problems. In particular, they can solve the "3-colorability problem" in linear time and linear resources (nodes, symbols, rules).

Theorem 10. The "3-colorability problem" can be solved with a computational complexity $O(m+n)$. We use a simple network of evolutionary processors. The network size is $7m+2$; n is the number of vertices and m is the number of edges of the "3-colorability graph" [6].

The main idea consists of building a simple network. First there are $2n$ steps, n represents the communication steps when nothing is being communicated, the strings will remain in N_0 until there are not any letters appearing in T anymore. When this process is finished, the obtained strings encode all possible ways of marking the vertices. Therefore it is possible to know whether the problem requirements are fulfilled or not.

The interesting part is that the underlying graph of the above network does not depend on the number of the instance nodes. 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. This process is not affected by the number of nodes.

Theorem 11. The "3-colorability problem" can be solved in $O(m+n)$ time by k networks of evolutionary processors with an octahedron graph ($NEP[OCT]$). $NEP[OCT]^p$, $p = 1, \ldots, k$ modules must be connected using the core processors. Where n is the number of vertices, m is the number of edges within the "3-colorability graph" and $k = int(\frac{4m+1}{7})$.

If $4m+1 > 7$ then several octahedron networks must be used in order to get the right number of processors. Communication among such octahedron networks depends on the problem to solve. In this case, a complete communication scheme could be used.

Proof. Let $G = (\{1, 2, \ldots, n\}, \{e_1, e_2, \ldots, e_m\})$ be a graph and assume that $e_t = \{k_t, l_t\}, 1 \leq k_t \leq l_t \leq n, 1 \leq t \leq m$. We consider the alphabet $U = V \cup V' \cup T \cup A$, where $V = \{b, r, g\}$, $T = \{a_1, a_2, \ldots, a_n\}$, and $A = \{\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_n\}$.

We construct the following processors of an octahedron network:

- A generator processor:

$$N_0 = \{\{a_1 a_2 \ldots a_n\}, \{a_i \to b\hat{A}_i, a_i \to r\hat{A}_i, a_i \to g\hat{A}_i | 1 \leq i \leq n\}, \emptyset, \{\hat{A}_i | 1 \leq i \leq n\}\}.$$

This processor generates all possible color combinations, solutions or not, to the problem; and it sends those strings to the next processors.

- For each edge in the graph $e_t = \{k_t, l_t\}$, we have 4 filtering processors (where $i \in \{k_t, l_t\}$):

$$\begin{aligned} N_{e_t^1} &= \{\emptyset, \{g\hat{A}_i \to g'a_i, r\hat{A}_i \to r'a_i\}, \{\hat{A}_i\}, \{g', r'\}\}, \\ N_{e_t^2} &= \{\emptyset, \{g\hat{A}_i \to g'a_i, b\hat{A}_i \to b'a_i\}, \{\hat{A}_i\}, \{g', b'\}\}, \\ N_{e_t^3} &= \{\emptyset, \{b\hat{A}_i \to b'a_i, r\hat{A}_i \to r'a_i\}, \{\hat{A}_i\}, \{b', r'\}\}, \\ N_{e_t^4} &= \{\emptyset, \{r'a_i \to r\hat{A}_i, g'a_i \to g\hat{A}_i, b'a_i \to b\hat{A}_i\}, \{a_i\}, \{\hat{A}_i\}\}. \end{aligned}$$

We can build an octahedron network with previous nodes in such a way that N_0 generates all possible colored strings and then applies rules to objects in processors $N_{e_t^1}, N_{e_t^2}, N_{e_t^3}, N_{e_t^4}$ to filter such strings for the edge e_t. Repeating such filtering process with the rest of edges provides a valid solution to the given problem.

If more than one $NEP[OCT]$ is needed, then all processors from the first $NEP[OCT]$ will be linked to the core processor of the second one, and so on.

An octahedron network with the above architecture can solve the *3-colorability problem* of n cities with m edges.

For the first n steps, that are evolution representatives where nothing is being communicated, the strings will remain in N_0 until there are not any letters appearing in T anymore. When this process is finished, the obtained strings encode all possible ways of coloring the vertices.

After this, step 1 is needed to communicate all possible solutions to the next processors. At this point, for each edge e_t, the network only keeps those strings which encode a colorability satisfying the condition for the two vertices of e_t. This is performed by means of the nodes $N_{e_t^1}, N_{e_t^2}, N_{e_t^3}$, and finally $N_{e_t^4}$ in 12 steps. As we can see, the overall time of a computation is $12m + n + 1$. We finish the proof by referencing that the total number of rules as $18m + 3n + 1$. In conclusion, all parameters of the network are of $O(m + n)$ size. □

An ad-hoc simulation has been coded in order to solve the *3-colorability problem*. This software uses the *Java* threaded model to get a massive parallel simulation of NEPs. All concurrent access to objects are safe thread due to the implementation of object locks. All processors, rules and filters run in a separated thread and have been synchronized via software patterns. It is clear that this simulation does not achieve a linear computation time $O(m + n)$ since it has been run on a sequential machine. But it opens up a testing platform of theorems concerning NEP properties.

Here it is the final configuration of the system at the last node of the network after the filtering process done in previous nodes which is the solution to the given problem in Figure 2.

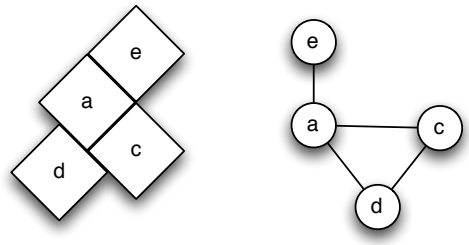


Figure 2. 3-colorability problem that has been solved using a massive paralell NEP

```
Processor: Objects: (12) [
   rAbCgDbE, gAbCrDbE, gAbCrDrE, rAgCbDgE, bAgCrDgE, rAgCbDbE,
   rAbCgDgE, bAgCrDrE, bArCgDrE, gArCbDrE, bArCgDgE, gArCbDbE
]
```

Where $\{XY|X \in \{r(ed), g(reen), b(lue)\}, Y \in \{a, c, d, e\}\}$ codes the color of the cities, that is, X means the color of the city Y in the map. Table below shows all objects in processor N_0 after applying the evolution rules. Such processor has 256 objects, each one is obtained using a given rule. This object set is – theoretically – obtained in $n = 4$ steps and contains all possible combinations, solutions or not, to the given problem.

```
Processor: Objects:(256)[ acde, rAcde, gAcde, bAcde, acrDe, acgDe, acbDe, arCde, rAcrDe, rAcgDe,
rAcbDe, gAcrDe, gAcgDe, gAcbDe, bAcrDe, bAcgDe, bAcbDe, arCrDe, arCgDe, arCbDe, acdrE, acdgE, agCde,
abCde, rArCde, rAgCde, rAbCde, gArCde, gAgCde, gAbCde, bArCde, bAgCde, bAbCde, agCrDe, abCrDe,
agCgDe, abCgDe, agCbDe, abCbDe, rArCrDe, rAgCrDe, rAbCrDe, rArCgDe, rAgCgDe, rAbCgDe, rArCbDe,
rAgCbDe, rAbCbDe, gArCrDe, gAgCrDe, gAbCrDe, gArCgDe, gAgCgDe, gAbCgDe, gArCbDe, gAgCbDe, gAbCbDe,
bArCrDe, bAgCrDe, bAbCrDe, bArCgDe, bAgCgDe, bAbCgDe, bArCbDe, bAgCbDe, bAbCbDe, arCdrE, agCdrE,
abCdrE, arCdgE, agCdgE, abCdgE, acdbE, rAcdrE, rAcdgE, rAcdbE, gAcdrE, gAcdgE, gAcdbE, bAcdrE,
bAcdgE, bAcdbE, acrDrE, acrDgE, acrDbE, acgDrE, acgDgE, acgDbE, acbDrE, acbDgE, acbDbE, arCdbE,
rAcrDrE, rAcrDgE, rAcrDbE, rAcgDrE, rAcgDgE, rAcgDbE, rAcbDrE, rAcbDgE, rAcbDbE, agCdbE,
abCdbE, rArCdrE, rAgCdrE, rAbCdrE, rArCdgE, rAgCdgE, rAbCdgE, rArCdbE, rAgCdbE, rAbCdbE,
gArCdrE, gAgCdrE, gAbCdrE, gArCdgE, gAgCdgE, gAbCdgE, gArCdbE, gAgCdbE, gAbCdbE, bArCdrE,
bAgCdrE, bAbCdrE, bArCdgE, bAgCdgE, bAbCdgE, bArCdbE, bAgCdbE, bAbCdbE, arCrDrE, agCrDrE,
abCrDrE, arCrDgE, agCrDgE, abCrDgE, arCrDbE, agCrDbE, abCrDbE, arCgDrE, agCgDrE, abCgDrE,
arCgDgE, agCgDgE, abCgDgE, arCgDbE, agCgDbE, abCgDbE, arCbDrE, agCbDrE, abCbDrE, arCbDgE,
agCbDgE, abCbDgE, arCbDbE, agCbDbE, abCbDbE, rArCrDrE, rAgCrDrE, rAbCrDrE, rArCrDgE,
rAgCrDgE, rAbCrDgE, rArCrDbE, rAgCrDbE, rAbCrDbE, rArCgDrE, rAgCgDrE, rAbCgDrE, rArCgDgE,
rAgCgDgE, rAbCgDgE, rArCgDbE, rAgCgDbE, rAbCgDbE, gAcrDrE, gAcrDgE, gAcrDbE, gAcgDrE,
gAcgDgE, gAcgDbE, gAcbDrE, gAcbDgE, gAcbDbE, bAcrDrE, bAcrDgE, bAcrDbE, bAcgDrE, bAcgDgE,
bAcgDbE, bAcbDrE, bAcbDgE, bAcbDbE, rArCbDrE, rArCbDgE, rArCbDbE, rAgCbDrE, rAgCbDgE,
rAgCbDbE, rAbCbDrE, rAbCbDgE, rAbCbDbE, gArCrDrE, gArCrDgE, gArCrDbE, gAgCrDrE, gAgCrDgE,
gArCgDrE, gArCbDrE, gAgCgDrE, gAgCbDrE, gAbCrDrE, gAbCgDrE, gAbCbDrE, gArCgDgE, gArCbDgE,
gAgCgDgE, gAgCbDgE, gAbCrDgE, gAbCgDgE, gAbCbDgE, gArCgDbE, gArCbDbE, gAgCrDbE,
gAgCgDbE, gAgCbDbE, gAbCrDbE, gAbCgDbE, gAbCbDbE, bArCrDrE, bArCgDrE, bArCbDrE, bAgCrDrE,
bAgCgDrE, bAgCbDrE, bAbCrDrE, bAbCgDrE, bAbCbDrE, bArCrDgE, bArCgDgE, bArCbDgE,
bAgCrDgE, bAgCgDgE, bAgCbDgE, bAbCrDgE, bAbCgDgE, bAbCbDgE, bArCrDbE, bArCgDbE,
bArCbDbE, bAgCrDbE, bAgCgDbE, bAgCbDbE, bAbCrDbE, bAbCgDbE, bAbCbDbE ]
```

7 CONCLUDING REMARKS AND FUTURE WORK

This paper introduced a variant of hybrid networks of evolutionary processors models, called *octahedron networks of evolutionary processors* (*NEP[OCT]*). The main idea consists of modular building networks sets of evolutionary processors that can solve *NP problems*. An octahedron representation of the underlying graph is chosen since a network with such configuration has computational completeness, (see Theorems 8 and 9). Different octahedron networks dynamics have been proposed ($NEP[OCT]_{c\alpha}$,$NEP[OCT]_{p\alpha}$) in order to obtain a massive parallel nondeterministic behavior. Therefore different kinds of rules inside the processors belong to the network ($NEP[OCT]_{X*}, NEP[OCT]_{Xr}, NEP[OCT]_{Xd}$, where $X \in \{p, c\}$).

Next, theoretical research work in this area covers *NP problem* solutions. Using these simple mechanisms we can solve NP-complete problems (such as Bounded Post Correspondence Problem, 3-Colorability Problem and Common Algorithmic Problem) in linear time following [6] and also [7, 15].

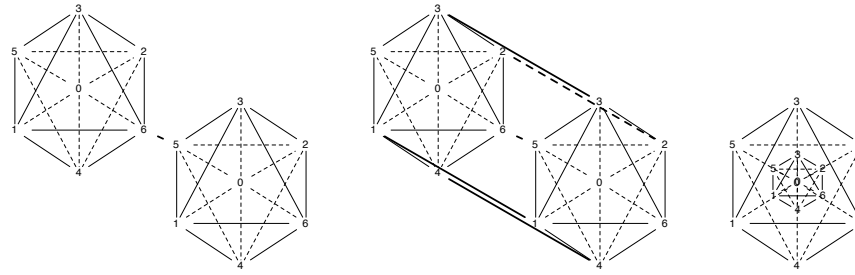


Figure 3. Different ways to communicate octahedron networks of evolutionary processors: a) surface-to-surface, b) edge-by-edge, c) inner cube

Networks of evolutionary processors are computation systems based on the biomolecular processes of living cells. According to this, the investigations are based on the idea of the imitation of the procedures that take place in nature, and their application to machines, can lead to discover and to develop new computation models which will give place to a new generation of intelligent computers. There are many papers about software tools implementing different NEP system variants. However, they are very interesting in order to define hardware implementation of these kinds of systems. Moreover, evolution of NEP systems is very complicate to be translated into hardware devices due mainly to inherent behavior of processing capabilities of rules. Besides that, the non-deterministic maximally parallel manner in which rules are applied inside processors is more appropriated to be implemented in digital hardware devices. In the case of NEPs hardware implementations only a few related papers can be found.

We have proposed a new topology whose underlying architecture is a cube graph having evolutionary processors placed in its nodes. Being a bio-inspired system:

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.

An interesting research area covers the software simulation, such as [16]. It will be performed as the first step towards a real hardware implementation of these computing devices by using models proposed by [4, 5]. Octahedron networks are computing units and they can be combined (easily in real hardware devices) in different ways, see Figure 3, in order to solve *big* problems. Two octahedronds can be connected by using one processor of each cube (Figure 3 a)), information will be controlled by these two processors among the communicating path. They can also be connected using 5 processors of each cube (Figure 3 b)), communication between the cubes is distributed in several processors. Finally, a recursive building can be constructed (Figure 3 c)). These models are designed in order to produce modular hardware devices.

REFERENCES

[1] Alhazov, A.—Martin-Vide, C.—Rogozhin, Yu.: On the Number of Nodes in Universal Networks of Evolutionary Processors. Acta Informatica, Vol. 43, 2006, pp. 331–339.

[2] Alhazov, A.—Martin-Vide, C.—Rogozhin, Yu.: Networks of Evolutionary Processors with Two Nodes Are Unpredictable. Technical Report 35/07, Rovira i Virgili University, Language and Automata Theory and Applications (LATA 2007), Tarragona, and TUCS Report, Vol. 818, Turku 2007, pp. 521–527.

[3] Alhazov, A.—Martin-Vide, C.—Truthe, B.—Dassow, J.—Rogozhin, Yu.: On Networks of Evolutionary Processors with Nodes of Two Types. Fundam. Inform., Vol. 91, 2009, No. 1, pp. 1–15.

[4] Alonso, S.—Fernández, L.—Arroyo, F.—Gil, J.: A Circuit Implementing Massive Paralelism in Transition P Systems. International Conference on Information Research and Applications (ITECH'07), Varna, Bulgary, June 25–30, 2007, pp. 159–167.

[5] Bravo, G.—Fernández, L.—Arroyo, F.—Frutos, J. A.: A Hierarchical Architecture with Parallel Communication for Implementing P Systems. International Conference Information Research and Applications (ITECH'07), Varna, Bulgary, June 25–30, 2007, pp. 168–174.

[6] Castellanos, J.—Martín-Vide, C.—Mitrana, V.—Sempere, J.: Solving NP-Complete Problems with Networks of Evolutionary Processors. Lecture Notes in Computer Science, Springer-Verlag, Vol. 2084, 2001, pp. 621–628.

[7] Castellanos, J.—Martin-Vide, C.—Mitrana, V.—Sempere, J.: Networks of Evolutionary Processors. Acta Informatica, Vol. 39, 2003, pp. 517–529.

[8] Csuhaj-Varjú, E.—Mitrana, V.: Evolutionary Systems, a Language Generating Device Inspired by Evolving Communities of Cells. Acta Informatica, Vol. 36, 2000, pp. 913–926.

[9] Dassow, J.—Mitrana, V.: Splicing Grammar Systems. Computers and Artificial Intelligence, Vol. 15, 1996, No. 2-3, pp. 109–122.

[10] Geffert, V.: Normal Forms for Phrase-Structure Grammars. Theoretical Informatics and Applications, Vol. 25, 1991, No. 5, pp. 473–496.

[11] The Universal Turing Machine. A Half-Century Survey. Springer-Verlag, 1994.

[12] Kari, L.—Păun, Gh.—Thierrin, G.—Yu, S.: At the Crossroads of DNA Computing and Formal Languages: Characterizing RE Using Insertion-Deletion Systems. Proceedings of the 3rd DIMACS Workshop on DNA Based Computing, Philadelphia, 1997, pp. 318–333.

[13] Margenstern, M.—Mitrana, V.—Pérez-Jiménez, M.-J.: Accepting Hybrid Networks of Evolutionary Processors. In: Ferretti, C., Mauri, G., Zandron, C. (Eds.): DNA 10. Springer, Lecture Notes in Computer Science, Vol. 3384, 2005, pp. 235–246.

[14] Martín-Vide, C.—Păun, Gh.—Salomaa, A.: Characterizations of Recursively Enumerable Languages by Means of Insertion Grammars. Theoretical Computer Science, Vol. 205, 1998, No. 1-2, pp. 195-205.

[15] Martín-Vide, C.—Mitrana, V.—Perez-Jimenez, M.—Sancho-Caparrini, F.: Hybrid Networks of Evolutionary Processors. Lecture Notes in Computer Science, Vol. 2723, 2003, pp. 401–412.

[16] Diaz, M. A.—Peña, M. A.—de Mingo, L. F.: Simulation of Networks of Evolutionary Processors with Filtered Connections. WSEAS Transactions on Information, Science and Applications. Vol. 4, 2007, No. 3, pp. 608–616.

[17] Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences, Vol. 61, 2000, No. 1, pp. 108–143.

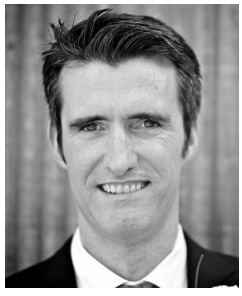
[18] Păun, G.—Rozenberg, G.—Salomaa, A.: DNA Computing, New Computing Paradigms. Springer-Verlag, 1998.

[19] Rozenberg, G.—Salomaa, A.: The Handbook of Formal Languages. Springer-Verlag, 1997.

[20] Sankoff, D.—Leduc, G.—Antoine, N.—Paquin, B.—Lang, B. F.—Cedergren, R.: Gene Order Comparisons for Phylogenetic Inference: Evolution of the Mitochondrial Genome. Proceedings of the National Academy of Sciences of USA, Vol. 89, 1992, pp. 6575–6579.

Luis Fernando de Mingo López is currently Associate Professor at the Technical University of Madrid from 1999. He works in the Department Organization of Data Structures. His main area of interest includes the study of new computational models such as neural networks, ant colony optimization, particle swarm optimization, grammatical swarm, etc. He holds his Ph.D. in this field and has published more than 100 papers. He is also an editor of the "International Scientific Society Journal" and participates in various international research projects funded by the EU.

Nuria Gómez Blas is currently Associate Professor at the Technical University of Madrid from 2005. She works in the Department Organization of Data Structures. Her main area of interest includes the study of new computational models such as neural networks and particle swarm optimization, and the innovation of educational method optimization. She holds her Ph.D. in this field.



Alberto Arteta currently works as Assistant Professor in Troy University of Alabama. He works in the Computer Science Department since August 2015. Previously he was Associate Professor at the Technical University of Madrid from 2007 at the Departments of Applied Mathematics and Information Systems. His main area of interest includes the study of new computational models such as membrane computing, cells computing, evolutionary computing and neural networks. He holds his Ph.D. in this field and has published papers in prestigious journals and in symposium including Engineering Application of Artificial Intelligence, Evolutionary Computation, and EEE Membrane Computing Symposium. Furthermore, in the last 13 years, he was working mainly in Holland and Spain for private companies. Most of his roles have been: a developer, a system engineer or a database consultant. The last companies he has worked for are IBM, Citco, Roche Pharma, Progress, McAfee, Amadeus IT and Everis Spain.