String-based learning in networks of evolutionary processors with filtered connections

Luis Fernando de Mingo López, Alberto Arteta Albert, Nuria Gómez Blas

Escuela Universitaria de Informática, Universidad Politécnica de Madrid
Crta. de Valencia km. 7, 28031 Madrid, Spain
{lfmingo,aarteta,ngomez}@eui.upm.es

Abstract. This paper presents some connectionist models that are widely used to solve NP-problems. Most well known numeric models are Neural Networks that are able to approximate any function or classify any pattern set provided numeric information is injected into the net. Neural Nets usually have a supervised or unsupervised learning stage in order to perform desired response. Concerning symbolic information new research area has been developed, inspired by George Paun, called Membrane Systems. A step forward, in a similar Neural Network architecture, was done to obtain Networks of Evolutionary Processors (NEP). A NEP is a set of processors connected by a graph, each processor only deals with symbolic information using rules. In short, objects in processors can evolve and pass through processors until a stable configuration is reach. This paper shows some ideas about these two models and how to incorporate a learning stage, based on self-organizing algorithms, in networks of evolutionary processors.

1 Networks of Evolutionary Processors

P-systems represent a class of distributed and parallel computing devices of a biological type that was introduced in [10, 9] which are included in the wider field of cellular computing. Several variants of this model have been investigated and the literature on the subject is now rapidly growing. The main results in this area show that P-systems are a very powerful and efficient computational model [5, 13]. There are variants that might be classified according to different criteria. They may be regarded as language generators or acceptors, working with strings or multisets, developing synchronous or asynchronous computation. Two main classes of P-systems can be identified in the area of membrane computing [11]: cell-like P-systems and tissue-like P-systems. The former type is inspired by the internal organization of living cells with different compartments and membranes hierarchically arranged; formally this structure is associated with a tree. Tissue P-systems have been motivated by the structure and behaviour of multicellular organisms where they form a multitude of different tissues performing various functions [8]; the structure of the system is instead represented as a graph where nodes are associated with the cells which are allowed to communicate alongside the edges of the graph.
Universality results have been obtained [7, 12, 6] for a number of variants of population P-systems. The following different rules are considered: transformation rules for modifying the objects that are present inside the cells, communication rules for moving objects from a cell to another one, cell division rules for introducing new cells in the system, cell differentiation rules for changing the types of the cells, and cell death rules for removing cells from the system. As well as this, bond-making rules are considered that are used to modify the links between the existing cells (i.e., the set of edges in the graph) at the end of each step of evolution performed by means of the aforementioned rules. In other words, a population P-system in [1, 2] is basically defined as an evolution-communication P-system but with the important difference that the structure of the system is not rigid and it is represented as an arbitrary graph. In particular, bond making rules are able to influence cell capability of moving objects from a place to another one by varying the set of edges in the underlying graph.

1.1 Formal Definition

A network of evolutionary processors [4, 3] of size $n$ is $\Gamma = (V, N_1, N_2, \ldots, N_n, G)$, 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
  - $a \rightarrow b, a, b \in V$ (substitution rules)
  - $a \rightarrow \epsilon, a \in V$ (deletion rules)
  - $\epsilon \rightarrow a, a \in V$ (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 $w \in PI_i (w \in PO_i)$.

A configuration can change either by an evolutionary step or by a communicating step [4]. 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. [4]

Theorem 2. Each recursively enumerable language can be generated by a star NEP of size 5. [4]

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. [3]

1.2 Simple Networks of Evolutionary Processors

A simple NEP [4] of size $n$ is a construct $\Gamma = (V, N_1, N_2, \cdots, N_n, G)$, 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. [4]

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. [4]

2 Networks of Evolutionary Processors with Filtered Connections

Artificial Neural Networks (ANN) and Networks of Evolutionary Processors (NEP) can be considered as the present and the future of connectionist models. Both of them are based on the idea of simple processors that communicate in order to achieve a global objective. But there are two important facts that must be taken into account:
ANN are numeric models while NEP are symbolic ones.

There exists a learning algorithm that control the ANN behavior in order to achieve a desired result while NEP do not incorporate any kind of learning paradigm.

Some ideas of ANN can be translated into a NEP architecture since ANN are considered, in the literature, a good model to solve non conventional problems. Following this point of view some kind of learning can be added to a NEP to obtain a more general model than simple NEP. Among all the neural networks architectures unsupervised neural networks, called Self Organizing Maps (SOM), are the most suitable one to translate into a NEP.

First of all, the learning concept in self-organizing maps are explained in order to translate such ideas to a NEP, then a model with filtered connections is shown to finally include learning in NEP models.

2.1 Filtered Connections

Main idea in NEPs is based on the fact that filters are inside processors in order to control what objects can pass through connections, but these filters make complex processors. If such filters are in connections, instead in processors, the simplicity of processors will increase compare to classical NEPs.

A network of evolutionary processors with filtered connections of size $n$ is a construct $\Gamma = (V, N_1, N_2, \cdots, N_n, G)$, where $V$ is an alphabet and for each $1 \leq i \leq n$, $N_i = (M_i, A_i)$ 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 (substitution, 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.

Finally, $G = (\{N_1, N_2, \cdots, N_n\}, (E, F))$ is an directed graph called the underlying graph of the network. The edges of $G$, that is the elements of $(E, F)$, are given in the form $(e_i, f_i)$ where $f_i$ is the filter associated to connection $e_i$. Elements in $F$ are just object sets, an element $w$ pass the filter in $f_i$ if $w \in f_i$.

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 $C = (L_1, L_2, \cdots, L_n)$, 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 $C_0 = (A_1, A_2, \cdots, A_n)$.

A configuration can change either by an evolutionary step or by a communicating step. 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 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 all sent/received information pass filters in connections).

**Theorem 6.** Solved problems using NEPs can be solve using NEPs with filtered connections.

Given two processor of a NEP, \(N_i\) and \(N_j\) connected by edge \(e_{ij}\) and with filters \((FI_i, FO_i)\) and \((FI_j, FO_j)\), they can be transformed into two processors of a NEP with filtered connections in the following way:

- Remove filters from processors \(N_i\) and \(N_j\) to obtain processors of NEPFC \(M_i\) and \(M_j\).
- Add the filter \(f_{ij} = FO_i \land FI_j\) to a connection from processor \(M_i \rightarrow M_j\).
- Add the filter \(f_{ji} = FO_j \land FI_i\) to a connection from processor \(M_i \leftarrow M_j\).

It is clear that this kind of transformation produces a NEPFC with the same behaviour than a NEP.

**Theorem 7.** A NEPFC with \(m\) processors and less than \(c = 2m\) connections can not be transformed into an equivalent NEP.

Each \(c\) connection is an equation with two unknows, so there are \(2c\) unknows (input and output filters in NEPs) and there exists \(2m\) filters to compute. So if the \(c < 2m\) the system has infinite solutions but the behaviour will not be the same in all cases. If \(c = 2m\) the system has only one solution, and if \(c > 2m\) the system has only one solution. Therefore, if NEPs and NEPFCs are equivalent under some constraints then all theorems in NEPs are valid for NEPFC. This new model can solve NP-problems in linear time.

### 3 Learning with Filtered Connections

NEPs and NEPFCs can be consired universal models since the are able to solve NP-problems. The great disadvantage is that a given NEP/NEPFC can only solve a given problem, if it is necessary to solve another problem (maybe a little variation) then another different NEP/NEPFC has to be implemented. The idea of learning tries to undertake such disadvantage proposing a model able to solve different kinds of problems (that is a general class of problems). Learning proposed here is based on the self organizing maps describe above.

Let \(w\) a string object where \(w = a_1a_2\cdots a_n\). Distance between two objects \(\delta(w_i, w_j)\) can be computed as the length of symbol \(z = w_i \land w_j\).

A learning stage can be added to NEPFCs in the following way:

- if an object pass a filter in a given direction \(\rightarrow\) then the filter in the other direction \(\leftarrow\) is modified in order to avoid this object to come back.
– if an object does not pass a filter in a given direction $\rightarrow$ then the filter in the other direction $\leftarrow$ is modified in order to permit this object to go forward.

An object $w$ in a NEPFC with a learning stage $\Delta$ will pass through a connection $c$ if $\sum_i \delta(w, x_i) < \Delta$, where $x_i \in f_c$ and $\Delta$ is the threshold of the learning algorithm. If the object successfully pass the filter $\rightarrow$ then this object $w$ will be added to $f_c$ on the other direction $\leftarrow$. If object $w$ does not pass the filter $\rightarrow$ then it will be added to filter $\leftarrow$.

As NEPs, NEPFC with learning are non deterministic models and they also are massive parallel models that one reason why they can solve NP-problems in linear time.

There are some open problems that are part of our future research:

– Can NEPFCs with learning solve NP-problems? Probably yes, since their behavior is similar to NEPs.
– Are NEPFCs with learning and $\Delta = 0$ equivalent to NEPFCs? And to NEPs?
– If previous question is affirmative, then what is the maximum value of $\Delta$ to obtain equivalent models?

4 Simulation Results: 3-Colorability Problem

A software tool 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.

Fig. 1. 3-colorability problem that has been solved using a massive paralell NEP.

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 1.
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. Figure 2 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.

5 Concluding Remarks and Future Work

This paper has introduced the novel computational paradigm Networks of Evolutionary Processors. Connectionists models such as Neural Networks can be taken into account to develop NEP architecture in order to improve behaviour. As a
future research, learning concepts in neural networks can be adapted in a NEP architecture provided the numeric-symbolic difference in both models.

Artificial Neural Networks as universal approximators, Transition P Systems and Networks of Evolutionary Processors as NP-problem solvers are the main connectionist models in the field of Natural Computation. All of them are bio-inspired and try to model biological process that happens in nature. This paper has proposed a new model as a combination of NEPs and SOMs adding some kind of learning to NEPs.

There are a lot of open problems in grammar theory that need to be solved in order to show the computational power of this model, but the possibility to compute NP-problems is promising apart from the massive parallelization and non-determinism of the model.

References