# Rule representation in distributed environments with accepting networks of splicing processors

Luis Fernando de Mingo López, Nuria Gómez Blas, Juan Castellanos

**Abstract.** This paper presents the model named Accepting Networks of Evolutionary Processors as NP-problem solver inspired in the biological DNA operations. A processor has a rules set, splicing rules in this model, an object multiset and a filters set. Rules can be applied in parallel since there exists a large number of copies of objects in the multiset. Processors can form a graph in order to solve a given problem. This paper shows the network configuration in order to solve the SAT problem using linear resources and time. A rule representation arquitecture in distributed environments can be easily implemented using these networks of processors, such as decision support systems, as shown in the paper.

## 1 Introduction

This work is a continuation of the investigation started in [1] and [2] where one has considered a mechanism inspired from cell biology, namely networks of evolutionary processors, that is networks whose nodes are very simple processors able to perform just one type of point mutation (insertion, deletion or substitution of a symbol). These nodes are endowed with filters which are defined by some membership or random context condition.

Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell having a genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of strings, each copy being processed in parallel such that all the possible evolutions events that can take place do actually take place.

These networks may be used as language (macroset) 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. It is worth mentioning here the similarity of this model to that of a P system, a new computing model inspired by the hierarchical and modularized cell structure recently proposed in [3, 4].

Networks of evolutionary processors (NEP) [1], [2] are language generating device, 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, then using these simple mechanisms we can solve NP-complete problems in linear time. Such solutions are presented for the *Bounded Post Correspondence Problem* in [5], for the *3-Colorability Problem* in [1] and for the *Common Algorithmic Problem* in [6]. As a further step, in [6] the so-called hybrid networks of evolutionary processors are considered. Here deletion node or insertion node has its own 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.

This organization is suitable for a hardware implementation, similar to those carried out with transition P systems [7, 8].

## 2 Accepting Networks of Splicing Processors

Networks of evolutionary processors (NEP) have proof to be a powerful tool in order to solve NP-problems [5, 1]. Evolution is based on the idea of mutation rules but rules can be modified to obtain a closer model to biological ones based on DNA operations [9]. That is, the crossover operation can be modeled with splicing rules. Such NEPs, with splicing rules, are name splicing NEPs. A possible application is to check if a word over a languaje is accepted or not by the NEP, named accepting networks of splicing processors (ANSP) [10, 11].

A splicing processor over $V$ is a 6-tuple $(S, A, PI, FI, PO, FO)$ where: $S$ is a finite set of splicing rules over $V$, $A$ is a finite set of auxiliary words over $V$ (these words are used in a processor to perform the splicing operation), $PI, FI \subseteq V$ are the permiting/forbidden input context in the processors, while $PO, FO \subseteq V$ are the permiting/forbidden output context in processors (with $PI \cap FI = \emptyset$ and $PO \cap FO = \emptyset$). The set of all processors over $V$ is denoted by $SP_V$.

A network of evolutionary processors with splicing rules is a 6-tuple $\Gamma = (V, U, G, \mathcal{N}, \alpha, x_I, x_O)$, where:

- $V$ an $U$ are the input and network alphabets respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ in an undirected graph with the vertex set $X_G$ and the edge set $E_G$. $G$ is named the underlying graph of the net.
- $\mathcal{N} : X_G \to SP_U$ is a mapping that maps a given node $x \in X_G$ the splicing processor $\mathcal{N}(x) = (S_x, A_x, PI_x, FI_x, PO_x, FO_x)$.
- $\alpha : X_G \to \{(1), (2), (3), (4)\}$ defines the behaviour of the input/output filters. Given a node $x \in X_G$, he has the following filters:

$$\text{input filter: } \rho_x(.) = \varphi^{\beta(x)}(.; PI_x, FI_x)$$

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

That is, $\rho_x(w)$ (respectively $\tau_x$) shows if a word $w$ can pass or not the input/output filter in $x$. $\rho_x(L)$ (respectively $\tau_x(L)$) is the set of words in $L$ that can pass the input/output filter in $x$.

- $x_I, x_O \in X_G$ are the input/output nodes in the net $\Gamma$.

Let $card(X_G)$ the size of $\Gamma$. If $\alpha$ is a constant mapping then the net is homogeneous. In the field of network theory some kinds of graphs are widely used: complete, ring, start, mesh, etc. Networks proposed in this paper can have such shapes. This work deals with complete graphs, denoted by $K_n$, where $n$ is the number of vertex.

An $ANSP$ configuration is a mapping $C : X_G \to 2^{V^*}$ that maps a node belonging to the graph with a word set. A configuration can be understood as the sets of words that are in the nodes of the network in a given moment. Given a word $w \in V^*$, initial configuration of $\Gamma$ in $w$ is defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)} = \emptyset$ for all $x \in X_G - \{x_I\}$.

A configuration can change by means of a splicing step or a communication step. When a splicing step is performed, each element $C(X)$ in configuration $C$ is changed according to the set of splicing rules $M_x$ in processor $x$ using set $A_x$. Formally, a configuration $C'$ is obtained with one splicing step from configuration $C$, denoted by $C \Rightarrow C'$, iff:

$$C'(x) = S_x(A_x, C(x)) \forall x \in X_G$$

When changing due to a communication step, each node $x \in X_G$ sends a copy of every word, provided they are able to pass the input filter in $x$, to all connected nodes to $x$ and receives all words sent by any other processor connected to $x$ provided they pass its input filter. Formally, a configuration $C'$ is obtained in one communication step from configuration $C$, denoted by $C \vdash C'$, iff:

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

Let $\Gamma$ an $ANSP$, a computation of $\Gamma$ over input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_3^{(w)}, \cdots,$, where $C_0^{(w)}$ is the initial configuration of $\Gamma$ over $w$, $C_{2i}^{(w)} \Rightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Configuration $C_i^{(w)}$ is determined by $C_{i-1}^{(w)}$. Every computation in an $ANSP$ is deterministic. A configuration finish (and it is said to be finite) if some of below conditions are accomplished:

- There exists a configuration in which the set of words in node $x_O$ is a not empty set. In this case, the computation is an accepting computation.
- There are two identical configurations.

The language accepted by $\Gamma$ is:

$$L_a(\Gamma) = \{w \in V^* | \text{ accepting computation of } \Gamma \text{ over } w\}$$

An $ANSP$ decides the language $L \subseteq V^*$, denoted by $L(\Gamma) = L$ iff $L_a(\Gamma) = L$ and the computation of $\Gamma$ over any $x \in V^*$ finishes.

### 2.1 Solution to SAT problem

SAT problem can be solved in linear time using an Accepting Network of Splicing Processors – $ANSP$ –. Also resources in the whole net are lineally bounded by the size of the $SAT$ instance.

Let $V$ the set of variables, $V = \{x_1, x_2, \cdots, x_n\}$ and let $\phi = (C_1) \wedge (C_2) \wedge \cdots (C_m)$ a boolean formula, where the negation of a given variable $x_i$ is denoted by $\bar{x}_i$. Each formula can be considered as a word over the alphabet $V \cup \bar{V} \cup \{\wedge, \vee, (, )\}$, where $\bar{V} = \{\bar{x} | x \in V\}$. The following alphabet is defined:

$$W = \{[x_i = 1], [x_i = 0] | 1 \leq i \leq n\} \cup V \cup \bar{V} \cup \{\vee, \wedge, (, ), \$, \#, \uparrow\}$$

Next, an $ANSP$ is defined by $\Gamma = (V, W, K_{2n+2}, \mathcal{N}, \alpha, In, Out)$, where $K_{2n+2}$ is the complete graph with $2n + 2$ nodes: $In$, $Out$, $(x_i \leftarrow 0)$, $(x_i \leftarrow 1)$ with $1 \leq i \leq n$. Each node is defined by:

- $In$:
  - $S_{In} = \{[(\$[x_n = b], \epsilon); (\$, ()] \ b \in \{0, 1\}\} \cup \{[(\$[x_i = b], \epsilon); (\$, [x_{i+1} = c]) | b, c \in \{0, 1\}, 1 \leq i \leq n - 1\}$
  - $A_{In} = \{\$[x_i = b] | b \in \{0, 1\}, 1 \leq i \leq n\}$
  - $PI_{In} = \emptyset, FI_{In} = W, PO_{In} = \{[x_1 = 0, [x_1 = 1]\}, FO_{In} = \emptyset, \alpha(In) = (4)$
- $(x_1 \leftarrow 0), 1 \leq i \leq n$:
  - $S_{(x_i \leftarrow 0)}$ has all splicing rules in form $[(\uparrow, C'\beta\#); ([x_n = b], (C)\beta\#)]$, where
    $(i)$ $b \in \{0, 1\}, C)\beta$ is a suffix of $\phi$, with $C \in (V \cup \bar{V} \cup \{\vee\})^+$
    $(ii)$ $C'$ $\begin{cases} \epsilon, \text{ if } C \text{ starts with } \bar{x}_i \\ (\gamma), \text{ if } C = x_i \vee \gamma, \gamma \in (V \cup \bar{V})^+, \\ \uparrow, \text{ if } C = x_i \text{ or } C \text{ starts with } x_j \text{ or } \bar{x}_j \\ \text{ for some } j \neq i \end{cases}$
  - $A_{(x_i \leftarrow 0)} = \{\uparrow C'\beta\# | C', \beta\}$
  - $PI_{(x_i \leftarrow 0)} = \{[x_i = 0]\}, FI_{(x_i \leftarrow 0)} = \{\uparrow\}, PO_{(x_i \leftarrow 0)} = \emptyset, FO_{(x_i \leftarrow 0)} = \emptyset, \alpha((x_i \leftarrow 0)) = (1)$
- $(x_1 \leftarrow 1), 1 \leq i \leq n$:
  - $S_{(x_i \leftarrow 1)}$ has all splicing rules in form $[(\uparrow, C'\beta\#); ([x_n = b], (C)\beta\#)]$, where
    $(i)$ $b \in \{0, 1\}, C)\beta$ is a suffix of $\phi$, with $C \in (V \cup \bar{V} \cup \{\vee\})^+$
    $(ii)$ $C'$ $\begin{cases} \epsilon, \text{ if } C \text{ starts with } \bar{x}_i \\ (\gamma), \text{ if } C = x_i \vee \gamma, \gamma \in (V \cup \bar{V})^+, \\ \uparrow, \text{ if } C = x_i \text{ or } C \text{ starts with } x_j \text{ or } \bar{x}_j \\ \text{ for some } j \neq i \end{cases}$

- $A_{(x_i \leftarrow 1)} = \{\uparrow C'\beta\#|C', \beta\}$
- $PI_{(x_i \leftarrow 1)} = \{[x_i = 0]\}, FI_{(x_i \leftarrow 1)} = \{\uparrow\}, PO_{(x_i \leftarrow 1)} = \emptyset, FO_{(x_i \leftarrow 1)} = \emptyset, \alpha((x_i \leftarrow 1)) = (1)$

– *Out*:
- $S_{Out} = A_{Out} = PI_{Out} = PO_{Out} = \emptyset$
- $FI_{Out} = V \cup \bar{V} \cup \{(, )\vee, \wedge, \uparrow\}, FO_{Out} = W, \alpha(Out) = (1)$

Resources in the $ASNP$ are lineally bounded, that is, the size of $\Gamma$ ant the number of symbols provided the auxiliary words and splicing rules are lineally bounded in each node by the input formula. It is important to note that the underlying architecture of the $ANSP$ does not change if the number of instance variables is constant. It can be said that the net is a computer program: filters and splicing rules are user dependent and the formula is computed substituting variables with values, one by one, from left to rights.

## 3 Rules in Distributed Environments

Opportunities for building business expert systems abound for both small and large problems. In each case, the expert system is built by developing its rule set. The planning that precedes rule set development is much like the planning that would precede any project of comparable magnitude within the organization. The development process itself follows an evolutionary spiral composed of development cycles.

Each cycle picks up where the last ended, building on the prior rule set. For a developer, the spiral represents a continuing education process in which more and more of an expert's reasoning knowledge is discovered and formalized in the rule set. Here, each development cycle was presented in terms of seven consecutive stages. Other characterizations of a development cycle (involving different stages or sequences) may be equally valuable. Many aspects of traditional systems analysis and project management can be applied to the development of expert systems. Rule set development is a process of discovery and documentation. Research continues in search of ways of automating various aspects of the process. It would not be surprising to eventually see expert systems that can assist in this process – that is, an expert system that "picks the mind" of a human expert in order to build new expert systems. Until that time comes, the topics discussed in this chapter should serve as reminders to developers of expert decision support systems about issues to consider during the development process.

As it stands, rule-based systems are the most widely used and accepted AI in the world outside of games. The fields of medicine, finance and many others have benefited greatly by intelligent use of such systems. With the combination of rule-based systems and ID trees, there is great potential for most fields.

It is necessary to define the underlying graph In order to simulate an example with a network of evolutionary processors. First of all, an initial processor containing the assertions and basic rules will forward important information to

a second processor, which is in charge of a given disease, and forward its result to a container processor. This simple example can be extended to a more general diagnosis system. There exists one processor or even more processors in charge of a located diagnosis problem such as: influenza, migraines, heart diseases, etc These local diagnosis processors can communicate each other to auto complete information diagnosis. Finally, each result of diagnosis processors is sent to an information processor that can combine multiple diagnoses or just show them.

Figure 1 shows an XML file with the NEP initial configuration corresponding to the medical diagnosis.

There are three processors: assertion process (name = 0), specific diagnosis processor (name = 1) and diagnosis information processor (name = 2). Objects travel trough these processors until a final diagnosis is present in the last one. Obviously, this is a simple example, but the NEP architecture could be complicated in order to obtain a more sophisticated diagnosis. Main idea is to put some assertions in one or more processors and then let them evolve using evolution steps (rules application) and communication steps. This configuration file is parsed into JAVA objects and a separate thread for each processor is created; also each rule and filter are coded as threads in order to keep the massive parallelization defined in the theoretical model of Networks of Evolutionary Processors.

### 3.1 Simulation Results

Results concerning simulation of NEP configuration can be seen below. Processor $2:3$ , which is the output processor, has the object influenza, desired result. This object is generated in Processor $1:2$ using rules inside it. NEP behavior is totally non-deterministic since rules, filters and processors run together in parallel. This example only uses substitution rules, neither insertion nor deletion rules are coded.

```
[----------
Processor 0 : 1
Rules: [[runny nose] --> [nasal congestion],
[body-aches] --> [achiness],
[high temperature] --> [fever],
[headache] --> [achiness]]
Objects: [runny nose, high temperature, headache]
Output Filter: [nasal congestion, fever, achiness, cough]
Input Filter: []
---------- , ----------
Processor 1 : 2
Rules: [[nasal congestion, viremia] --> [influenza],
[fever, achiness, cough] --> [viremia]]
Objects:
[cough, fever, achiness, viremia, nasal congestion, influenza]
Output Filter: [influenza]
Input Filter: [nasal congestion, fever, achiness, cough]
---------- , ----------
```

```
Processor 2 : 3
Rules: []
Objects: [influenza]
Output Filter: []
Input Filter: [influenza]
----------]
```

The great disadvantage is that a given NEP can only solve a given problem; if it is necessary to solve another problem (maybe a little variation) then another different NEP 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 can be based on the self-organizing maps. There are a lot of open problems that need to be solved in order to show the computational power of this learning idea, but the possibility to compute NP-problems is promising apart from the massive parallelization and non-determinism of the model.

## 4 Conclusions and Future Work

This paper has introduced the computational paradigm Accepting Networks of Evolutionary Processors. *ANSP* can be easily applied to Knowlede-driven Decision Support Systems due to the inherent rule-based behavior of *ANSP*. JAVA implementation of this model works as defined by the theoretical background of *ANSP*: massive parallelization and non-deterministic behavior. Connectionists models such as Neural Networks can be taken into account to develop a new *ANSP* architecture in order to improve behavior. As a future research, learning concepts in neural networks can be adapted in *ANSP* architecture provided the numeric-symbolic difference in both models. *ANSP* can be considered universal models since they are able to solve NP-problems.

## References


1. J. Castellanos, C. Martin-Vide, V. Mitrana, and C. Sempere, "Networks of evolutionary processors," *Acta Informatica*, vol. 39, pp. 517–529, 2003.
2. E. C. Varju and V. Mitrana, "Evolutionary systems, a language generating device inspired by evolving communities of cells," *Acta Informatica*, vol. 36, pp. 913–926, 2000.
3. G. Paun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.
4. G. Paun, G. Rozenberg, and A. Salomaa, *DNA Computing, new computing paradigms.* Springer-Verlag, 1998.
5. J. Castellanos, C. Martin-Vide, V. Mitrana, and C. Sempere, "Solving np-complete problems with networks of evolutionary processors," *Lecture Notes in Computer Science*, vol. 2084, pp. 621–628, 2001.
6. C. Martin-Vide, V. Mitrana, and M. Perez-Jimenez, "Hybrid networks of evolutionary processors," *Lecture Notes in Computer Science*, vol. 2723, pp. 401–412, 2003.

7. S. Alonso, L. Fernandez, F. Arroyo, and J. Gil, "A circuit implementing massive paralelism in transition p systems," in *Information Research and Applications*, 2007, pp. 159–167.
8. G. Bravo, L. Fernandez, F. Arroyo, and J. A. de Frutos, "A hierarchical architecture with parallel communication for implementing p systems," in *Information Research and Applications*, 2007, pp. 168–174.
9. C. Martin-Vide and V. Mitrana, "Networks of evolutionary processors: results and perspectives," *Molecular Computational Models: Unconventional Approaches*, pp. 78–114, 2005.
10. F. Manea, C. Martin-Vide, and V. Mitrana, "All np-problems can be solved in polynomial time by accepting networks of splicing processors of constant size," *Lecture Notes in Computer Science*, vol. 4287, pp. 47–57, 2006.
11. ——, "Accepting networks of splicing processors: complexity results," *Theoretical Computer Science*, no. 371, pp. 72–82, 2007.

```
<?xml version="1.0"?>
<NEP>
   <processor>
      <name>0</name>
      <object>runny nose</object>
      <object>high temperature</object>
      <object>headache</object>
      <object>cough</object>
      <rule>
         <antecedent>
            <object>runny nose</object>
         </antecedent>
         <consequent>
            <object>nasal congestion</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>body-aches</object>
         </antecedent>
         <consequent>
            <object>achiness</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>high temperature</object>
         </antecedent>
         <consequent>
            <object>fever</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>headache</object>
         </antecedent>
         <consequent>
            <object>achiness</object>
         </consequent>
      </rule>
      <inputfilter>
      </inputfilter>
      <outputfilter>
         <object>nasal congestion</object>
         <object>fever</object>
         <object>achiness</object>
         <object>cough</object>
      </outputfilter>
   </processor>
   <processor>
      <name>1</name>
      <rule>
         <antecedent>
            <object>nasal congestion</object>
            <object>viremia</object>
         </antecedent>
         <consequent>
            <object>influenza</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>fever</object>
            <object>achiness</object>
            <object>cough</object>
         </antecedent>
         <consequent>
            <object>viremia</object>
         </consequent>
      </rule>
      <inputfilter>
         <object>nasal congestion</object>
         <object>fever</object>
         <object>achiness</object>
         <object>cough</object>
      </inputfilter>
      <outputfilter>
         <object>influenza</object>
      </outputfilter>
   </processor>
   <processor>
      <name>2</name>
      <inputfilter>
         <object>influenza</object>
      </inputfilter>
      <outputfilter>
      </outputfilter>
   </processor>
   <conn>
      <from>0</from>
      <to>1</to>
   </conn>
   <conn>
      <from>1</from>
      <to>2</to>
   </conn>
</NEP>
```

**Fig. 1.** XML sample corresponding to a network of evolutionary processors in medical diagnosis applications