ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **12**, Number 2, 2009, 235–247

Extended Networks of Evolutionary Processors – ENEPs

Luis Fernando DE MINGO[1], Nuria GÓMEZ[1], Juan CASTELLANOS[2]

[1] Dept. of Organización y Estructura de la Información, Escuela de Informática
Universidad Politécnica de Madrid, Crta. de Valencia km.7, Madrid, Spain, 28031
E-mail: {lfmingo,ngomez}@eui.upm.es

[2] Dept. of Inteligencia Artificial, Facultad de Informática
Universidad Politécnica de Madrid, Campus de Montegancedo, Madrid, Spain, 28660
E-mail: jcastellanos@fi.upm.es

Abstract. This paper presents some connectionist models that are widely used to solve NP-problems. This paper shows some ideas about how to incorporate a learning stage, based on self-organizing algorithms, in networks of evolutionary processors. *T. Kohonen* and *P. Somervuo* have shown that self organizing maps (SOM) are not restricted to numerical data. This paper proposes a symbolic measure that is used to implement a string self organizing map based on SOM algorithm. Such measure between two strings is a new string. Computation over strings is performed using a priority relationship among symbols, in this case, symbolic measure is able to generate new symbols. A complementary operation is defined in order to apply such measure to DNA strands. Finally, an algorithm is proposed in order to be able to implement a string self organizing map. This paper discusses the possibility of defining networks of evolutionary processors to rely on similarity instead of distance and shows examples of such networks for symbol strings.

1. Introduction

Neural Networks are well known numeric models, which are able to approximate any function or classify any pattern set, provided that sufficient numeric information, regarding that set, is injected into the network. This injection of information is usually formalized by a supervised or unsupervised learning stage. On the other hand, a new research area, concerning symbolic information processing and classification, has been developed, inspired by the works of G. Păun [1], the theory of Membrane Systems. A step forward in this direction was done to obtain Networks of Evolutionary Processors (*NEP*), introduced by Victor Mitrana [2]. A *NEP* is a set of processors placed in the nodes of a graph, and connected by its edges; each processor only deals with symbolic information using simple rewriting rules. For short, the objects (strings) in processors can evolve and pass through processors until a stable configuration is reached.

Self Organizing maps are usually used for mapping complex, multidimensional numerical data onto a geometrical structure of lower dimensionality, like a rectangular or hexagonal two-dimensional lattice [3]. The mappings are useful for visualization of data, since they reflect the similarities and vector distribution of the data in the input space. Each node in the map has a reference vector assigned to it. Its value is a weighted average of all the input vectors that are similar to it and to the reference vectors of the nodes from its topological neighborhood. For numerical data, average and similarity are easily computed: for the average, one usually takes the arithmetical mean, and the similarity between two vectors can be defined as their inverse distance, which is most often the Euclidian one. However, for non-numerical data [4]– like symbol strings – both measures tend to be much more complicated to compute. Still, like their numerical counterparts, they rely on a distance measure. For symbol strings one can use the Levenshtein distance or feature distance.

For strings, one such measure is the *Levenshtein* distance [5], also known as edit distance, which is the minimum number of basic edit operations – insertions, deletions and replacements of a symbol – needed to transform one string into another. Edit operations can be given different costs, depending on the operation and the symbols involved. Such weighted *Levenshtein* distance can, depending on the chosen weighting, cease to be distance in the above sense of the word.

Another measure for quantifying how much two strings differ is *feature distance* [3]. Each string is assigned a collection of its substrings of a fixed length. The substrings the features are typically two or three symbols long. The feature distance is then the number of features in which two strings differ. It should be noted that this measure is not really a distance, for different strings can have a zero distance. Nevertheless, feature distance has a practical advantage over the Levenshtein by being much easier to compute.

A *similarity measure* is simpler than distance. Any function $\mathcal{S} : X^2 \to R$ can be declared similarity – the question is only if it reflects the natural relationship between data. In practice, such functions are often symmetrical and assign a higher value to two identical elements than to distinct ones, but this is not required.

2. String Measure

Let V an alphabet over a set of symbols. A string x of length m belonging to an alphabet V is the sequence of symbols $a_1a_2\cdots a_m$ where the symbol $a_i \in V$ for all $1 \leq i \leq m$. The set of all strings over V is denoted by V^*, the empty symbol is λ and the empty string is denoted by $\epsilon = (\lambda)^*$.

Let $\mathcal{O} : x \rightarrow n, x \in V, n \in \mathcal{N}$ a mapping that establish a priority relationship among symbols belonging to V, $u \leq v$ iff $\mathcal{O}(u) \leq \mathcal{O}(v)$. Obviously $\mathcal{O}^{-1}(\mathcal{O}(x)) = x, x \in V$ and $\mathcal{O}(\mathcal{O}^{-1}(n)) = n, n \in \mathcal{N}$, and $\mathcal{O}(\lambda) = 0, \mathcal{O}^{-1}(0) = \lambda$. This mapping can be extended over an string w in such a way that $\mathcal{O}(w) = \sum \mathcal{O}(w_i), w_i \in w$. Usually, such mapping $\mathcal{O}$ covers a range of integer numbers, that is, the output is $0 \leq i \leq k$, where $k = card(S), S \subseteq V$.

It is important to note that new symbols can be generated provided that given two symbols $a, b \in V$ $|\mathcal{O}(a) - \mathcal{O}(b)| > 1$, and there is no symbol c such that $\mathcal{O}(a) < \mathcal{O}(c) < \mathcal{O}(b)$. That is,

$$\mathcal{O}^{-1}(k) = \left\{ \begin{array}{cc} x \in V & \text{iff } \mathcal{O}(x) = k \\ s_k & \text{i.o.c.} \end{array} \right. , \text{ with } k \in \mathcal{N}.$$

Symbolic measure between two strings $u, v \in V^*$, denoted by $\Delta(u, v)$, with $|u| = |v| = n$ is another string defined as:

$$\Delta(u, v) = \bigcup_{i=1}^{n} \mathcal{O}^{-1}(|\mathcal{O}(u_i) - \mathcal{O}(v_i)|), \text{ where } u_i/v_i \text{ is the } i\text{-th symbol} \in u/v. \quad (1)$$

For example, let $u = (abcad), v = (abdac)$, and $\mathcal{O}$ the index of such symbol in the latin alphabet, that is, $\mathcal{O}(a) = 1, \mathcal{O}(b) = 2, \mathcal{O}(c) = 3, \mathcal{O}(d) = 4$ then $\Delta(u, v) = \lambda\lambda a \lambda a$. If $u = (jonh), v = (mary)$ then $\Delta(u, v) = s_3 n j s_{11}$, two new symbols s_3, s_{11} are generated (that correspond to $s_3 = c$ and $s_{11} = k$, usually such correspondence is unknown).

A numeric value $\mathcal{D}$ can be define over a string w:

$$\mathcal{D}(w) = \sqrt{\sum_{i=0}^{|w|} \mathcal{O}(w_i)^2}, w_i \in w. \quad (2)$$

It is clear to proof that: $\mathcal{D}(\Delta(u, v)) = \mathcal{D}(\Delta(v, u))$, $\mathcal{D}(\Delta(u, u)) = 0$, $\mathcal{D}(\Delta(u, \epsilon)) = \mathcal{D}(u)$ and $\mathcal{D}(\Delta(u, w)) \leq \mathcal{D}(\Delta(u, v)) + \mathcal{D}(\Delta(v, w))$.

Mappings $\mathcal{O}/\mathcal{D}$ also define a priority relationship among strings in V^* is such a way that

$$u \leq v \text{ iff } \sqrt{\sum_{i=1}^{n=|u|} \mathcal{O}(u_i)^2} \leq \sqrt{\sum_{i=1}^{n=|v|} \mathcal{O}(v_i)^2},$$

$$u \leq v \text{ iff } \mathcal{D}(u) \leq \mathcal{D}(v).$$

For short, symbolic measure between two string u, v is obtained using $\Delta(u, v)$, see equation (2), and numeric measure is obtained using $\mathcal{D}(\Delta(u, v))$, see equation (1).

Let $x, y \in S \subseteq V$ two symbols belonging to alphabet, two symbols are complementary, denoted by $(x, y)^-$, iff $\Delta(x, y) = x$ or $\Delta(x, y) = y$. Such property can be extended over strings, let $u, v \in S^* \subseteq V^*$, two strings are complementary, denoted by $(u, v)^-$, iff $\Delta(u, v) = u$ or $\Delta(u, v) = v$.

Theorem 1. *Let $u, v \in S^*$, u and $\Delta(u,v)$ are complementary iff $\mathcal{O}(u_i) >= \mathcal{O}(v_i)$ for all $1 \leq i \leq n$.*

Proof.

$$\Delta(u,v) = \bigcup_{i=1}^{n} \mathcal{O}^{-1}(|\mathcal{O}(u_i) - \mathcal{O}(v_i)|)$$

Hence:

$$\begin{aligned}\Delta(u,\Delta(u,v)) &= \bigcup_{i=1}^{n} \mathcal{O}^{-1}(|\mathcal{O}(u_i) - \mathcal{O}(\Delta(u_i,v_i))|) = \\ &= \bigcup_{i=1}^{n} \mathcal{O}^{-1}(|\mathcal{O}(u_i) - \mathcal{O}(\mathcal{O}^{-1}(|\mathcal{O}(u_i) - \mathcal{O}(v_i)|)|) = \\ &= \bigcup_{i=1}^{n} \mathcal{O}^{-1}(|\mathcal{O}(u_i) - (|\mathcal{O}(u_i) - \mathcal{O}(v_i)|)|) = \\ &= \bigcup_{i=1}^{n} \mathcal{O}^{-1}(\mathcal{O}(v_i)) = v\end{aligned}$$

□

Two strings $u, v \in S^*$ are *Watson-Crick* complementary (WC complementary), denoted by $(u,v)^{-WC}$, iff $(u_i,v_i)^-$ for all $1 \leq i \leq |u|$.

Theorem 2. *Let $u, v \in S^*$, if $(u,v)^-$ then $(u,v)^{-WC}$.*

Such duality in symbolic/numeric measures, see equations (1) (2), is a good mechanism in order to implement algorithms on biological *DNA* strands [7, 8]. Like *DNA* or amino-acid sequences which are often subject to research in computational molecular biology. There, a different measure – similarity – is usually used. It takes into account mutability of symbols, which is determined through complex observations on many biologically close sequences. To process such sequences with neural networks, it is preferable to use a measure which is well empirically founded.

2.1. Different Length on Strings

Given two strings u, v, such that $|u| = n \geq |v| = m$, and $U(u)$ the set of all substring $w \subseteq u$ such that,

$$U(u)^m = \{w^{(j)} || w^{(j)}| = m, w = w_1 \cdots w_m, w_i = u_k, i = k + j\}, \\ \forall\, 0 \leq j \leq |u| - m.$$

The string measure between u, v, denoted by $\delta(u,v)$, is:

$$\delta(u,v) = \{\Delta(s,v) | s \in U(u)^{|v|}, \mathcal{O}(\Delta(s,v)) \leq min_{x \in U(u)^{|v|}}\{\mathcal{O}(\Delta(x,v))\}\}.$$

In this case, measure δ is the set of strings with the lower distance (see table below). Such distance can be read as a the set of all matching strings with lower distance. This δ can be used to identify cuting points (index j) over a *DNA* string when applying a restriction enzyme, from a biological point of view.

$u = abcdabcdab, v = cda$

$U(u)^{\|v\|}$										u
a	b	c								
	b	c	d							
		c	d	a						$\mathcal{O}(\Delta(cda, v)) = 0$
			d	a	b					
				a	b	c				
					b	c	d			
						c	d	a		$\mathcal{O}(\Delta(cda, v)) = 0$
							d	a	b	

$\delta(u, v) = \{\lambda\lambda\lambda, \lambda\lambda\lambda\}$

Let $|u| = |v|$, it is clear that $\delta(u, v) = \Delta(u, v)$ since $U(u) = u$.

3. String Self-Organizing Maps

The self-organizing map of symbol strings [3] (*SSOM* for short) does not differ much from ordinary numerical *SOM*. It is also a low dimensional lattice of neurons (usually two-dimensional quadratic or hexagonal lattice, sometimes one or three-dimensional), but instead of having a reference vector of input space dimensionality assigned to each node, reference strings are used. In the ordinary *SOM*, the reference vectors approximate the average of similar input vectors and input vectors similar to the reference vectors of the nodes from the topological neighborhood. In *SSOM*, the reference strings aproximate the averages of corresponding input strings [4].

A string self-organizing map of size n ($SSOM^n_{(i,j)}$ for short) is a construct $\Phi = \{I, C, \Omega\}$ where (i, j) are the dimensions of the competitive layer, other parameters are define as:

– $I = \{i_1, i_2, \cdots, i_n\}$ is the input nodes set,

– C is the competitive set, with $(i \times j)$ nodes,

$$C = \left\{ \begin{array}{cccc} c_{11} & c_{12} & \cdots & c_{1j} \\ c_{21} & c_{22} & \cdots & c_{2j} \\ \cdots & \cdots & \cdots & \cdots \\ c_{i1} & c_{i2} & \cdots & c_{ij} \end{array} \right\}$$

– and $\Omega : n \times (i \times j) \rightarrow \omega_{n,ij}$ is a function that identifies the connection between a given input node i and a competitive node (i, j), where $\omega_{n,ij} \in U \subseteq V$.

Given a set $U \subseteq V$ and $S = \{s_1, s_2, \cdots, s_k\}$ of strings in $U^* \subseteq V^*$ in such a way that the length of every string $s_i \in S$ is $|s_i| = n$ and a priority relationship among strings in S defined using a given mapping $\mathcal{O}$. The problem consists on finding the set defined by mapping Ω such that it minimizes the overall distance Δ with respect the input set S.

The algorithm is based on the *SOM* algorithm, but in this case everything is symbolic.

1. *Inizialitation*: Each element $\omega_{n,ij}$ is randomly assigned a symbol in U.

2. *Feeding*: One string $s_i \in S$ in presented in the input nodes set I. Nodes in I work in a simple way, they just store information they received. Each node $i_j \in I$ stores one symbol of string s_i, that is, $i_j = (s_i)_j, 1 \leq j \leq n$.

3. *Propagation*: Information on input nodes will pass through connection till competitive layer. Nodes in competitive layer works as follows, and they will store this information:

$$c_{ij} = \bigcup_{k=1}^{n} \Delta(i_k, \omega_{k,ij}) = \Delta(s_i, (\omega_{1,ij}\omega_{2,ij} \cdots \omega_{n,ij})).$$

   Such behavior is equivalent to compute distance between the input string and the connection string. This way competitive nodes calculate all distances with respect to the input string.

4. *Winning*: Nodes in C have all possible string measures, so there exists one node $c_{lm} \in C$ such that

$$\mathcal{D}(c_{lm}) \leq \mathcal{D}(c_{ij}), \forall i, j$$

   that is, node c_{lm} has the lower distance.

5. *Learning*: Only winning node will adjust his weights (based on winner-takes-all algorithm) according to following equation:

$$\omega_{i,lm} = \mathcal{O}^{-1}(\mathcal{O}(\omega_{i,lm}) + \alpha(\mathcal{O}(i_i) - \mathcal{O}(\omega_{i,lm}))).$$

Some results, in literature, that could be checked with this new measure can be: for an example application of the string SOM, *Igor Fisher* generated a set of 500 strings by intorducing noise to 8 english words: always, certainly, deepest, excited, meaning, remains, safety, and touch, and initialized a quadratic map with the Sammon projection of a random sample from the set [9]. Another real world example is the mapping produced from 320 hemoglobine alpha and beta chain sequences of different species [10]. *SOM* and *LVQ* algorithms for symbol strings have been introduced by [6, 4] and applied to isolated word recognition, for the construction of an optimal pronunciation dictionary for a given speech recognizer.

Table 1. Strings used in a *SSOM*,
they have been modified with uniform noise

universe	networks	emulsion	elements	referred	printing	moonlike
vnfyctpb	ndwwprjt	fpwktjok	fogmeqvt	rggcopbb	spkkrhkh	lprnljjb
rlkxhpth	mctunpnu	hmtiqlnm	bjdpdqtv	rbhhrscg	rsfnwiqd	olrkjjjg
rojtdprg	kgrxlumr	gjsnrlpk	bjcpdotu	sbihoqed	osiouiqf	nmqklhkh
rliwbuth	lctzppmt	bjwksknq	emfjdkqv	pggeoudd	oufqsjpe	lqqpmflh
ulfudotb	ohuxrthu	eoumsllp	bobjhlsu	shihpseg	orlqvflg	lnlplfjb
unfwhuve	lcwvorls	dormsjml	bocngnvq	qcfcuqba	pohosgmi	noomjgkc
tklubstg	phwvqtmu	dnslufnq	dleleovv	segdqocc	mpimqiqg	nlpkkhhb
uniufuuc	phsxlshu	bkvnrkpn	ckbpgmqt	obicqoed	qplkulnf	lmolkjjd
wmhwcpqc	peqwpphq	flululnm	glhmekss	qgdftugf	rojnqlnj	jqrkniie
sogveusc	qfuyqukr	gnvmpgqo	gibnfptv	qgfetrfg	stikvlpg	lrmpkllg
sphsdotb	kgtznoiv	hmwlslrk	dnbneorq	rffeorff	moimtflg	lnmmkjkc
uqgscrrd	qcturslt	boslqlrn	didmcmur	qdfeordg	qtkosiof	krmlnjnb
rmhsfrtd	pfrupqnp	bltiqfpl	foejbowu	tegbtuhb	osgnvimj	omnqkhng
xnkyequh	ndrymrls	elwlpflq	emfmhluv	phhgophe	osglsgpd	kmnnnflb
vniucqpf	kdsuptlp	cnwmtjqm	bkfpdotq	ubdcprbf	mrgotkqi	nmpomihf
vofsguth	nguyqslv	gmtkuhrl	fmelfosp	rhdgtpdd	qujqtlqj	kqrmokkf
smlthqrf	mcsymrlp	hmtipknl	djhpektq	pceeppff	srfowgmg	nmlpmgmb
skixbsvd	mdtyoukq	hpwirglq	gmhpcqrp	tgffuscg	pplowhqd	lmppkjie
rmfudtqf	lfsyqsmp	cjuouhqn	gkdkbptv	thcbsqce	nrklsfkh	mnqkjjlh
vkjvfttd	mcsynsns	gktmvhmq	dmbkfoqv	pgderpdb	qtgkuimi	oonpjgnf
uohvhrse	pertlqnv	cntouhqk	dihneqsr	scidosed	pqjosjkh	nqlqjflg
wphuhuqc	ngvyrpkq	eproqlnp	hlbmbqqr	rhdcusbc	mpkoqlqe	pnlplhhh
skltdrsd	kbtwntkt	bpxjtglk	bieoekus	ucfhoodc	mrgovgpi	pomknhhb
uplygpph	ndtuoojt	hnukvkpm	hlejcoqq	pgfbtobb	nuhpsgme	nmmqlgmb
snhxhtrb	qbsxopip	cpuiqgqq	fjhnblws	tecfsubg	sphkqjqe	kromjkjf

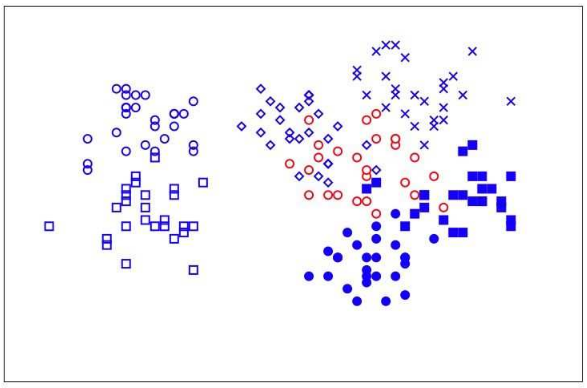

Fig. 1. Projection of string samples corresponding to Table 1.

Table 1 shows strings used in a *SSOM* with a 3×3 competitive layer. Strings are obtained adding uniform noise to original strings (first row in table). After the training phase is finished clusters are named using original strings. Data in table

1 are projected in a $2-d$ surface, figure 1 shows such projection. In this case we can observe that there is a clear separation among the 7 different clusters. This is a simple projection, since it seems that some clusters are mixed, we can use the Sammon projection to obtain a better data projection.

4. Networks of Evolutionary Processors with Filtered Connections

We consider that Artificial Neural Networks (*ANN*) and Networks of Evolutionary Processors (*NEP*) [11, 14] are 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.

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 proccessors. 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 3. *The class of languages accepted (decided) by NEPs equals the class of languages accepted (decided) by NEPFCs, and they are both equal to the class of recursively enumerable languages (recursive languages, respectively) [12, 15]. Remark: the class of problems solved by NEPs is equal to the class of problems solved by NEPFCs [13].*

The behavior of the two models is really similar due to their definitions. Also, in [12] and [13] the authors show that the main complexity/computability results holding for *NEPs* also hold for *NEPFCs*. Thus, the considerations on how *NEPFCs* with learning behave hold easily, with similar proofs as in the case of *NEPs*.

Theorem 4. *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 [12]. Moreover, a little discussion should be made regarding on what we understand by solving a problem with NEPs/NEPFCs [16].

5. Learning with Filtered Connections

NEPs and *NEPFCs* can be consired universal models since they were proven to be so in the above mentioned references [14, 12]. The great disadvantage is that a given *NEP/NEPFC* can olny solve a given problem, if it is necessary to solve another problem (maybe a little variation) then another diferent *NEP/NEPFC* has to be implemented. The idea of learning tries to undertake such disadvantage proposing a model able to solve diferent 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 $w = a_1 a_2 \cdots a_n$. Distance between two objects $\delta(w_i, w_j)$ can be computed as the length of symbol $z = w_i \wedge w_j$.

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 Δ will pass through a connection c if $\sum_i \delta(w, x_i) < \Delta$, where $x_i \in f_c$ and Δ is the threshold of the learning algorithm. If the object succefully 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 afirmative, then what is the maximum value of Δ to obtain equivalent models?

6. 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.

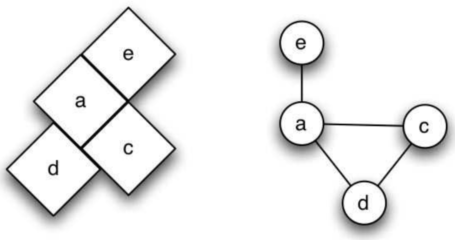

Fig. 2. 3-colorability problem that has been solved using a massive paralell NEP.

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.

```
Processor 16 : 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 0 :  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. Conclusions and Future Work

In some applications, like molecular biology, a similarity measure is more natural than distance and is preferred in comparing protein sequences. It is possible that such data can be successfully processed by self organizing neural networks. It can therefore be concluded that similarity-based neural networks are a promising tool for processing and analyzing non-metric data. This paper has proposed a string measure that can be applied to self organizing maps with the possibility of new symbols generation. Watson-Crick complementary concept was defined using such measure.

This paper has introduced the novel computational paradigm Networks of Evolutionay 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 lof 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.

Acknowledgements. We all thank to *Victor Mitrana* his support concerning this research. This work has been partially supported by spanish projects CCG08-UAM TIC-4425-2009 and TEC2007-68065-C03-02.

References

[1] PĂUN G., ROZENBERG G., SALOMAA A., *DNA Computing. New Computing Paradigms*, Springer, Berlin,1998.

[2] MARGENSTERN M., MITRANA V., PEREZ-JIMENEZ M., *Accepting hybrid networks of evolutionary systems*, Lecture Notes in Computer Science, **3384**, Springer-Verlag, Berlin, pp. 235–246, 2005.

[3] KOHONEN T., *Self-Organization and Associative Memory.* Springer, Berlin Heidelberg, (1988).

[4] KOHONEN T., SOMERVUO P., *Self-organizing maps of symbol strings*, Neurocomputing, **21**, pp. 19–30, 1998.

[5] LEVENSHTEIN L. I., *Binary codes capable of correcting deletions, insertions, and reversals*, Soviet Physics–Doklady, **10**, pp. 707–710, 1966.

[6] KOHONEN T., SOMERVUO P., *Self-Organizing Maps of Symbol Strings with Application to Speech Recognition*, (1997).

[7] SANCHEZ M., GOMEZ N., MINGO L., *DNA Simulation of Genetic Algorithms: Fitness Function*, International Journal on Information Theories and Applications, **14 (3)**, ISSN 1310-0513, pp. 211–217, 2007.

[8] GOMEZ N., SANTOS E., DIAZ M. A., *Symbolic Learning (Clustering) over DNA Strings*, WSEAS Transactions on Information Science and Applications, **3(4)**, ISSN: 1709-0832, pp. 617–624, 2007.

[9] FISCHER I., ZELL A., *String averages and self-organizing maps for strings*, *Proceeding of the ICSC Symposia on Neural Computation (NC'2000)*, May 23–26, 2000 in Berlin, Germany, pp. 208–215.

[10] FISCHER I., *Similarity-based neural networks for applications in computational molecular biology*, Lecture notes in computer science, **2779**, ISSN 0302-9743, pp. 208–218, 2003.

[11] CASTELLANOS J., MANEA F., MINGO L. F., MITRANA V., *Accepting Networks of Splicing Processors with Filtered Connections*, MCU, pp. 218–229, 2007.

[12] DRAGOI C., MANEA F., MITRANA V., *Accepting Networks of Evolutionary Processors with Filtered Connections*, J. UCS, **13(11)**, pp. 1598–1614, 2007.

[13] DRAGOI C., MANEA F., *On the Descriptional Complexity of Accepting Networks of Evolutionary Processors With Filtered Connections*, International Journal of Foundations of Computer Science, **19(5)**, World Scientific (2008), pp. 1113–1132.

[14] MANEA F., MITRANA V., *All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size*, Inf. Process. Lett., **103(3)**, pp. 112–118, 2007.

[15] MANEA F., MARTIN-VIDE C., MITRANA V., *Accepting networks of splicing processors: Complexity results*, Theor. Comput. Sci., **371(1–2)**, pp. 72–82, 2007.

[16] MANEA F., MARTIN-VIDE C., MITRANA V., *On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors*, Mathematical Structures in Computer Science, **17(4)**, Cambridge University Press, pp. 753–771, 2007.