A nonvectorial SOTA

Susana Vegas-Azcárate
Statistics and Decision Sciences Group
Rey Juan Carlos University
28933 Móstoles, Spain
susana.vegas@urjc.es

Joaquín Dopazo
Bioinformatics Department
Príncipe Felipe Research Center
46013 Valencia, Spain
jdopazo@ochoa.fib.es

Jorge Muruzábal
Statistics and Decision Sciences Group
Rey Juan Carlos University
28933 Móstoles, Spain
jorge.muruzabal@urjc.es

Abstract - *It may be the case that regular grids are not best suited for certain kinds of biological analysis. Specifically, sometimes a tree structure can provide the best means to organize and interpret the output neurons [Wen et al. (1992); Inoue and Narihisa (2003)]. In the case of biological data, the Self-Organizing Tree Algorithm (SOTA) [Dopazo and Carazo (1997a)] has shown promising results applied to phylogenetic analysis. Self-organizing neural networks have already been used for protein classification using sequence data [Ferrán et al. (1994); Ferrán and Pflugfelder (1993); Ferrán and Ferrara (1991, 1992); Andrade et al. (1996)]. The SOTA approach introduces a new type of self-organizing tree structure, namely, a structure which grows according to the pattern of speciation in the set of sequences analyzed [Dopazo and Carazo (1997a)]. Moreover, since sequences are coded residue by residue, all the information contained in the homologous positions of the alignment is available to SOTA, whereas standard SOM approaches encode the sequences as frequencies of residues or dipeptides, thus loosing potentially important sequence information. In this paper we approach a cross-fertilization of the above ideas. In particular, we study a variant of SOTA suitable for nonvectorial data. The core of the relevant algorithms is described first, then we look at a specific example in detail.*

Key words - Self-organizing map, tree structure, nonvectorial models.

Acknowledgement - *We are grateful to Diego Vidaurre for valuable computational assistance. The authors are partially supported by grants from the European Science Foundation.*

1 Introduction

The standard SOM is a nonlinear projection method that maps a high-dimensional metric vector space onto a (usually) two-dimensional regular grid in an orderly fashion [Kohonen (1982, 1995)]. The SOM algorithm follows an unsupervised learning process, implemented by competitive learning through a topological neighborhood. Although the SOM algorithm has been widely used with vectorial data, its principle is not restricted to metric vector spaces. Indeed, it has been pointed out [Kohonen (1996)] that any set of items for which a similarity or pseudo-distance measure is available could be mapped onto the SOM grid in an orderly fashion. Recently, Kohonen and Somervuo [Kohonen and Somervuo (2002)] have shown how to implement this principle in the case of fixed-size standard maps. Interestingly, they have illustrated their method using protein sequences as basic items and FASTA scores [Pearson and Lipman (1988); Pearson (1999)] as similarity values.

2 Symbol-SOM

It is shown in [Kohonen and Somervuo (2002)] how an extension of the SOM [Kohonen (1996)] called Symbol-SOM can be used for the clustering, organization and visualization of large databases of nonvectorial items such as protein sequences. The new method (originally suggested in [Kohonen (1996)]) allows the construction of the SOM when only a similarity measure is defined for pairs of items. Hence, a vectorial representation is not really needed; this is very good news, as many vectorial representations of biological data present important drawbacks and limitations. As remarked in [Kohonen and Somervuo (2002)], in order to define an ordered projection, it will be sufficient to compare the pairwise distances or similarities among items.

Symbol-SOM is based on the batch-learning version of the SOM [Kohonen (1995)], and it requires the computation of the *generalized median* of symbol strings [Kohonen (1985, 1995)]. Briefly, SOM-batch works as follows. Let $\mathbf{x}(j)$ be a vectorial training item and let $\mathbf{m}_i$ be the pointer associated with each grid neuron (just another vector of the same dimension). First, find the best matching unit or winner $\mathbf{m}_{w(j)}$ for each $\mathbf{x}(j)$ according to the usual requirement, namely

$$w(j) = \arg\min_i \{d[\mathbf{x}(j), \mathbf{m}_i]\} \tag{1}$$

where $d(\cdot, \cdot)$ is the underlying distance measure (typically the standard Euclidean distance). Second, update every pointer as,

$$\mathbf{m}_i = \frac{\sum_j h_{w(j),i}\mathbf{x}(j)}{\sum_j h_{w(j),i}} \tag{2}$$

where $h_{..} \equiv h_{..}(t)$ is the underlying time-varying neighborhood function for map units [Mulier and Cherkassky (1995)].

The generalized median is defined as follows [Kohonen (1985, 1995)]. Let $\Upsilon = \{x(i)\}$ be a set of items, and let $d[x(i), x(j)]$ be some distance (or pseudo-distance) measure between $x(i)$ and $x(j) \in \Upsilon$. Note that we denote general input items as x and m, whereas we write $\mathbf{x}$ and $\mathbf{m}$ for ordinary vectors. The generalized median m over Υ is defined as the item that

minimizes the sum of distances to all other items in Υ,

$$m = \arg \min_{x(i)\in\Upsilon} \sum_{x(j)\in\Upsilon:j\neq i} d[x(i), x(j)] \tag{3}$$

Hence, if the input samples had been real scalars and the distance measure were the absolute value of their difference, the generalized median would coincide with the usual (arithmetic) median.

We can now review the main features in symbol-SOM. To initialize the algorithm, *auxiliary vectorial pointers* are introduced, for the convergence of this algorithm is significantly faster and safer if the initial pointers are already two-dimensionally ordered [Kohonen and Somervuo (2002)]. In the case of proteins, these vectorial pointers can be selected as the usual 400-dimensional dipeptide histogram vectors [Ferrán and Ferrara (1991)]. Thus, each map node is provided with a 400-dimensional vector, each component of which is initialized with a random value between zero and unity (the whole vector is finally normalized to unit length). The standard SOM-batch algorithm is then trained with the dipeptide vectors, and the final pointers obtained are recoded to get symbol-SOM initialized. Specifically, for each vectorial pointer the usual subset of input items (including all items having that pointer as winner in the vectorial sense) is associated to it, and the corresponding nonvectorial pointer is chosen as the generalized median of that subset. With this labelling, a two-dimensionally set of relatively ordered input sequences is achieved, so that the non-vectorial symbol-SOM can proceed. From this point on, all vectorial representations are dropped.

For each pointer m_i, we now define two sets. First, we would recollect in W_i the input items associated to it, i.e., the input items that have m_i as its best-matching unit. Winning neurons could be determined as usual according to the FASTA method [Pearson and Lipman (1988); Pearson (1999)], but note that an input item could then have exactly the same distance to two or more pointers. Therefore, in order to make the winner unique in this case, we ask the winner to minimize the sum of distances from the input to all pointers in a small neighborhood around the winner candidate i, say N_i. This neighborhood includes all pointers within a certain radius from node i on the grid ; like in the traditional SOM, this radius can shrink monotonically with time. Mathematically, $x(j) \in W_i$ if and only if

$$m_i = \arg \min_{l} \sum_{k\in N_l} d(x(j), m_k).$$

Recollect now in Ω_i the input items associated to each pointer in N_i in the previous sense, that is, $\Omega_i = \bigcup_{k\in N_i} W_k$, and update each m_i as the generalized median of Ω_i. This is called the *adaptation process*. For each new pointer m_i, recollect in W_i the new input items associated to it as before. If the old W_i, say W_i^{old}, coincide with the new W_i for all i, then the process has converged. If not, continue with the adaptation process. When convergence is reached, pointers approximate the input items in an orderly fashion, since each pointer coincides with the generalized median of the input items mapped onto its neighborhood. See algorithm 1 for a flow-chart of symbol-SOM and algorithm 2 for details on the initialization method.

Algorithm 1
The nonvectorial version of SOM algorithm (symbol-SOM).

Initialize the map (see below).
repeat for each iteration, t,
  for each input sequence,$\mathbf{x}(n)$, **do**
    Find the best matching unit for $\mathbf{x}(n)$, see Equation 1.
  end for.
  Recollect in W_i the input items associated to pointer m_i.
  Store in Ω_i the items associated to each pointer in its neighbourhood N_i.
  Update each z_i as the generalized median (see Equation 3) of Ω_i.
until $W_i^t = W_i^{t-1}$, $\forall i$.

Algorithm 2
Initialization method for symbol-SOM in the case of proteins.

Convert input sequences into 400-dimensional dipeptide histogram vectors.
Provide each map neuron with a 400-dimensional vector.
repeat,
  Train a SOM-batch cycle,
until neurons are 2D ordered.
Label neurons by those proteins that represent the generalized medians of the sequences associated to them.

3 SOTA

The *Self-Organizing Tree Algorithm* [Dopazo and Carazo (1997a)] is based on both the Kohonen self-organizing map [Kohonen (1990)] and the growing cell structures algorithm due to Fritzke [Fritzke (1994)]. The input space is defined by the experimental input data, whereas the output space is arranged following a binary tree topology. The 'growing' of the output nodes can be stopped at the desired taxonomic level. Since SOTA accurately finds the phylogenetic tree that best relates the data [Dopazo and Carazo (1997a)], it becomes an appropriate tool for phylogenetic analysis of a large number of biological sequences such as proteins or DNA. Moreover, SOTA algorithm can also be used to classify other molecular data than proteins or nucleotide sequences, such as binary-coded restriction-site-map data [Dopazo and Carazo (1997a)]. Furthermore, other molecular fingerprinting data such as restriction enzyme fragment patterns [Nei and Li (1979)] or RAPD patterns [Clark and Lanigan (1993)] can also be analyzed [Dopazo and Carazo (1997a)] if the appropriate distance function is used.

We now review the main features of SOTA [Dopazo and Carazo (1997a)]. First, all input sequences must be aligned to the same length (say L) and then encoded as vectors [Casari et al. (1995)], see below. The initial structure consists of two external *cells* stemming from an internal cell (the root node). Each cell is a vector of length L randomly initialized with

Algorithm 3
The Self-Organizing Tree Algorithm (SOTA).

Align the input sequence.
Code the proteins into vectors.
Initialize the root node, C_{root}, and its descendants, C_{left} and C_{right}, with L random numbers ranging from 0 to 1,
repeat
  Change the state of the new mother from cell to node and initialize its children to be identical to it, (not in the first cycle).
  Generate two descendants from the cell having the highest R_i, see Equation 6, (not in the first cycle).
  repeat
    for each input sequence,
      Find the best matching cell.
      Store the obtained minimum distance.
      Update the winning cell and its neighbourhood via Equation 5.
    end for, (an epoch finishes).
    Store in R_i the local resource value.
    Store in E (see Equation 7) the actual network error.
  until $\epsilon = \left|\frac{E-E^{old}}{E^{old}}\right|$, (a cycle finishes).
until $\max_i R_i < R_0$.

real numbers between 0 to 1. Second, if A is the number of characters used in the alphabet, the distance between cell C_i and sequence S_j can be defined as

$$d(S_j, C_i) = \frac{\sum_{l=1}^{L}\left(1 - \sum_{a=1}^{A} S_j(a,l)\cdot C_i(a,l)\right)}{L} \tag{4}$$

It is easy to see that this function increases monotonically as the compared vectors become further and further apart. Notice that vector S_j encoding a protein sequence L amino-acids long is a $20 \times L$ matrix (or $21 \times L$, if gaps in the alignment are taken into account). Since the value stored in $C_i(a,l)$ is related to the probability of finding residue a in position l [Dopazo and Carazo (1997a); Casari et al. (1995)], sequences are encoded as vectors $S_j(a,l)$ with ones placed at the entries corresponding to the observed residues (and the rest set to 0) [Casari et al. (1995)].

As the initial structure grows, cells subdivide into two new (sister) external cells. Each subdivided cell is no longer external. To find the best matching cell for each input sequence, we stress that only external cells (not internal cells) will be compared to the sequence vectors (thus departing from Fritzke's original approach ; this is not surprising as Fritzke's structures are not trees in general). From now on, we drop the 'external' label for external cells and we rename all internal cells as nodes.

Next, the topological neighborhood of a winning cell depends on whether its sister neuron is a cell or not. If so, this neighborhood consists of the winning cell together with

its mother node and sister cell. If not, that is, if its sister neuron has descendants, then the neighborhood includes only the winning cell itself. More sophisticated variations can be considered. In any case, neighborhood adaptation follows the standard Kohonen learning rule [Kohonen (1990)],

$$C_i(t+1) = C_i(t) + \alpha_i \cdot \eta_t \cdot (S_j - C_i(t)) \; \forall i \in \mathcal{N}_j \tag{5}$$

where $\mathcal{N}_j$ equals one of the two previous neighborhoods, the various α_i control the magnitude of the updating ($\alpha_{winner} > \alpha_{mother} > \alpha_{sister}$) and η_t is a factor that accounts for the number of presentations. Notice that SOTA is updated by means of the *sequential* Kohonen algorithm, whereas symbol-SOM is based on the batch-learning version.

To continue with the description of the algorithm, two new concepts must be introduced. The process of (i) finding the *best matching cell*, Eq. (1), C_w, for each input sequence S_j, (ii) *storing* the minimum distances $d(S_j, C_w)$, and (iii) finally *updating* each winning cell and its neighborhood is called an *epoch* [Dopazo and Carazo (1997a)]. The series of epochs performed by the algorithm until the tree network converges is called a *cycle* (see below).

Once all input sequences have been presented to the tree, two measures are stored. First, the mean value of the distances between a cell and each of the sequences associated to it, namely the *local resource variable* [Dopazo and Carazo (1997a)], will be used to control the variability under each cell,

$$R_i = \frac{\sum_{k=1}^{K_i} d(S_k, C_i)}{K_i} \tag{6}$$

with K_i being the number of sequences associated to cell i. The second measure stored is the actual *error* of the network [Dopazo and Carazo (1997a)],

$$E = \sum_{i=1}^{I} K_i \cdot R_i \tag{7}$$

where I the number of cells in the current tree network. Note that the first epoch is only used to compute the critical resource and error variables.

At this point, check the relative decrease of the error, i.e.,

$$\epsilon = \left| \frac{E - E^{old}}{E^{old}} \right| \tag{8}$$

where E^{old} is the network error in the previous epoch (E^{old} is given a fixed value at the very beginning). If ϵ is bigger than a given threshold, ϵ_0, another epoch is performed, another local resource variable is associated to each cell and another network error value is calculated. When $\epsilon < \epsilon_0$ the previous cycle concludes.

Once a cycle has finished, two new descendants are generated for the cell having the highest resource value R_i, i.e., the most heterogeneous collection of associated inputs. This cell changes its status from cell to node, its descendants are initialized as identical copies and a new cycle begins. When thus splitting a cell, the sequences associated to it are assigned to its *range of influence*, i.e., in the next cycle these sequences will only be compared to the two new children cells [Dopazo and Carazo (1997b)]. In case the highest resource value reaches below a given threshold, say R_0, the desired taxonomic level is achieved and the algorithm finishes. See algorithm 3 for a flow-chart of SOTA.

Algorithm 4
The nonvectorial SOTA (nv-SOTA).

Initialize the root node and its descendants, see Equations 9, 10 and 11.
repeat
  Change the state of the new mother form cell to node and initialize its children (not in the first cycle),
  Generate two descendants from cell having the highest R_i, see Equation 6, (not in the first cycle).
  repeat
    for each input sequence,
      Find the best matching cell for actual input sequence.
      Store the obtained minimum distance.
    end for.
    Update the winning cell and its neighbourhood, by the generalized median of the set composed by Γ_i and itself.
    Store in R_i (see Equation 6) the local resource value.
    Store in E (see Equation 7) the actual network error.
  until $\epsilon = \left|\frac{E-E^{old}}{E^{old}}\right|$.
until $\max_i R_i < R_0$.

4 A nonvectorial SOTA

We now introduce a new SOTA approach for nonvectorial data combining ideas from symbol-SOM and the original SOTA. We denote the new algorithm as nv-SOTA ; it can be described as follows. We focus on the case of proteins for clarity, although any other symbolic structures could be used. First, like symbol-SOM, nv-SOTA always takes training sequences to characterize cells (and nodes). The FASTA method [Pearson and Lipman (1988); Pearson (1999)] can be used for the computation of similarities between protein sequences as before, since, as [Kohonen and Somervuo (1998)] pointed out, if x and y are any entities, a sufficient condition for them to be mapped into a SOM diagram is that some kind of *symmetric distance function* $d(x, y)$ is definable for all pairs (x, y). As in SOTA, the initial tree includes two (external) cells connected by an (internal) node. The root node is initialized to be the generalized median, Eq. (3), of all input sequences but, since sequence similarities (instead of distances) are actually used, the generalized medians are defined via the largest (instead of the smallest) sums of similarity values [Kohonen and Somervuo (2002)], i.e.,

$$m_{root} = \arg\max_{S_i \in \Upsilon} \sum_{S_j \in \Upsilon : j \neq i} d(S_i, S_j) \tag{9}$$

where Υ is the input sequence data set and $d(S_i, S_j)$ is extracted from the initial similarity matrix.

The left descendant is initialized to be the generalized median of all input sequences

except the previous root sequence. That is, if we define $\Upsilon \setminus S_{root} = \Upsilon_r$, then

$$m_{left} = \arg \max_{S_i \in \Upsilon_r} \sum_{S_j \in \Upsilon : j \neq i, root} d(S_i, S_j) \tag{10}$$

The right descendant is initialized similarly as the generalized median of all input sequences excluding the previous root sequence S_{root} and left-descendant sequence S_{left}, that is

$$m_{right} = \arg \max_{S_i \in \Upsilon_{r,l}} \sum_{S_j \in \Upsilon : j \neq i, root, left} d(S_i, S_j) \tag{11}$$

where $\Upsilon \setminus \{S_{root}, S_{left}\} = \Upsilon_{r,l}$. These roles could obviously be exchanged. It will be seen that all proteins m_i stored at the tree nodes and cells are different.

For each cell C_i, recollect now in L_i the input items associated to it, i.e., the input sequences S_j that have C_i as their best-matching cell, Eq. (1). As in SOTA, only cells but no nodes will be compared to the input sequence. Then, update each cell C_i by the generalized median of the set composed by L_i and itself, following the ideas developed in [Kohonen and Somervuo (2002)]. Studies about ancestor node adaptation are in course, following SOTA adaptation process ideas developed in [Dopazo and Carazo (1997a)]. Notice that, while in SOM-batch a neuron is updated as the *mean* of its Voronoi region, Eq. (2), in symbol-SOM and nv-SOTA each neuron is updated by means of the generalized *median* of its neighborhood. As in SOTA, the set of all previous operations is called an epoch. Note that, unlike SOTA, the adaptation is performed only after all input sequences have been presented to the tree. This is necessary due to the type of adaptation carried out by this network, which involves the generalized median of the cell neighborhood and not the sequential Kohonen learning rule as in SOTA.

Once an epoch is finished, recollect the local resource variable, Eq. (6), and the actual error of the network, Eq. (7). At this point, check the error's, Eq. (7), relative decrease. If this value is bigger than a given threshold, another epoch is performed (find the list of input sequences associated to each cell and update them by the generalized median of their lists and themselves), another local resource variable is associated to each cell and another network error value is calculated. The serie of epoch performed until the network has converged (when the relative decrease of the error falls below a given threshold) is again called a cycle.

Once a cycle has finished, two new descendants are generated from the cell having the highest resource value. This cell changes its status from cell to node and a new cycle begins. The left descendant is initialized as in Eq. (10) above, except Υ_r is now L_{mother}, the set of sequences associated to the mother. The right descendant is similarly initialized as in Eq. (11), except $\Upsilon_{r,l}$ is now $L_{mother} \setminus S_{left}$. As before, all ancestor and sister nodes are excluded from the summation indices. In case the highest resource value reaches below a given threshold, the desired taxonomic level is achieved and the algorithm finishes. See algorithm 4 for a summary of nv-SOTA.

5 Detailed example

We will see how nv-SOTA works in a simple, artificial scenario. Figure 1 shows the 31 items in our data set together with their topological organization. The data set can be divided into 5 levels, the first one only contains element A, the second level has the closest

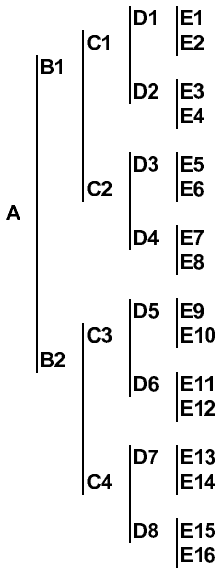

FIG. 1 – Data set in which A is the mother of $B1$ and $B2$, $B1$ is the mother of $C1$ and $C2$, $B2$ has $C3$ and $C4$ as its descendants, and so on. See text for further explanations.

descendants of A, $B1$ and $B2$; the third level includes the descendants of $B1$ and $B2$ and so on. To define a suitable distance measure, we simply look at proximity within the tree. Since we are reproducing a kind of phylogenetic tree, 'sister' cells at the fifth level must be less distant than sisters at the fourth level, sisters at the fourth level less distant than sisters at the third level, and so on. We assume that the distance between any pair of sisters at level five is 1 (unit), and also that the distance between any pair of sisters at level four is 2 (units). All the way up to level 1, where we assume that the distance between $B1$ and $B2$ is 4 (units). We also assume that the distance between a mother and its direct descendants is always 1. Figure 2 shows the resulting distance between any pair of elements in our set. For example, the distance between element $E2$ and element $C4$ is $d(E2, C4) = 1 + 1 + 1 + 4 + 1 = 8$, since we must traverse the path comprising the steps $E2$-$D1$ (add 1), $D1$-$C1$ (add 1), $C1$-$B1$ (add 1), $B1$-$B2$ (add 4) and finally $B2$-$C4$ (add 1).

Equipped with this distance information, nv-SOTA can start. First, the sum of the distances between each element and each of the remaining input items is calculated; the last column of Figure 2 provides these values. The root sequence is the element showing the minimum sum of distances, which in our case is 98 and corresponds to A. Second, the smallest sum of distances to all other elements (excluding the root A) is calculated, and this corresponds to $B1$ and $B2$, both with a minimum value of 128 units. Hence, the initial tree has A as its root node and $B1$ and $B2$ as its left and right descendants, respectively.

Now the first epoch of the first cycle begins. We must first find the input elements that have $B1$ and $B2$ as their best-matching cells, see the left table in Figure 3. Next we update $B1$ as the generalized median of the set composed by the items associated to it and itself, see the right upper table in Figure 3. The smallest sum of distances within this table corresponds to $B1$ (34 units), so this cell does not change. Likewise for $B2$, the generalized median of the set of input items associated to it and itself (see the right lower table in Figure 3) is again $B2$ (34 units). Hence, the first epoch finishes with A as the root node and $B1$ and $B2$ as its

	A	B1	B2	C1	C2	C3	C4	D1	D2	D3	D4	D5	D6	D7	D8	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	
A	0	1	1	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	**98**
B1	1	0	4	1	1	5	5	2	2	2	2	6	6	6	6	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	**129**
B2	1	4	0	5	5	1	1	6	6	6	6	2	2	2	2	7	7	7	7	7	7	7	7	3	3	3	3	3	3	3	3	**129**
C1	2	1	5	0	3	6	6	1	1	4	4	7	7	7	7	2	2	2	2	5	5	5	5	8	8	8	8	8	8	8	8	**153**
C2	2	1	5	3	0	6	6	4	4	1	1	7	7	7	7	5	5	5	5	2	2	2	2	8	8	8	8	8	8	8	8	**153**
C3	2	5	1	6	6	0	3	7	7	7	7	1	1	4	4	8	8	8	8	8	8	8	8	2	2	2	2	5	5	5	5	**153**
C4	2	5	1	6	6	3	0	7	7	7	7	4	4	1	1	8	8	8	8	8	8	8	8	5	5	5	5	2	2	2	2	**153**
D1	3	2	6	1	4	7	7	0	2	5	5	8	8	8	8	1	1	3	3	6	6	6	6	9	9	9	9	9	9	9	9	**178**
D2	3	2	6	1	4	7	7	2	0	5	5	8	8	8	8	3	3	1	1	6	6	6	6	9	9	9	9	9	9	9	9	**178**
D3	3	2	6	4	1	7	7	5	5	0	2	8	8	8	8	6	6	6	6	1	1	3	3	9	9	9	9	9	9	9	9	**178**
D4	3	2	6	4	1	7	7	5	5	2	0	8	8	8	8	6	6	6	6	3	3	1	1	9	9	9	9	9	9	9	9	**178**
D5	3	6	2	7	7	1	4	8	8	8	8	0	2	5	5	9	9	9	9	9	9	9	9	1	1	3	3	6	6	6	6	**178**
D6	3	6	2	7	7	1	4	8	8	8	8	2	0	5	5	9	9	9	9	9	9	9	9	3	3	1	1	6	6	6	6	**178**
D7	3	6	2	7	7	4	1	8	8	8	8	5	5	0	2	9	9	9	9	9	9	9	9	6	6	6	6	1	1	3	3	**178**
D8	3	6	2	7	7	4	1	8	8	8	8	5	5	2	0	9	9	9	9	9	9	9	9	6	6	6	6	3	3	1	1	**178**
E1	4	3	7	2	5	8	8	1	3	6	6	9	9	9	9	0	1	4	4	7	7	7	7	10	10	10	10	10	10	10	10	**206**
E2	4	3	7	2	5	8	8	1	3	6	6	9	9	9	9	1	0	4	4	7	7	7	7	10	10	10	10	10	10	10	10	**206**
E3	4	3	7	2	5	8	8	3	1	6	6	9	9	9	9	4	4	0	1	7	7	7	7	10	10	10	10	10	10	10	10	**206**
E4	4	3	7	2	5	8	8	3	1	6	6	9	9	9	9	4	4	1	0	7	7	7	7	10	10	10	10	10	10	10	10	**206**
E5	4	3	7	5	2	8	8	6	6	1	3	9	9	9	9	7	7	7	7	0	1	4	4	10	10	10	10	10	10	10	10	**206**
E6	4	3	7	5	2	8	8	6	6	1	3	9	9	9	9	7	7	7	7	1	0	4	4	10	10	10	10	10	10	10	10	**206**
E7	4	3	7	5	2	8	8	6	6	3	1	9	9	9	9	7	7	7	7	4	4	0	1	10	10	10	10	10	10	10	10	**206**
E8	4	3	7	5	2	8	8	6	6	3	1	9	9	9	9	7	7	7	7	4	4	1	0	10	10	10	10	10	10	10	10	**206**
E9	4	7	3	8	8	2	5	9	9	9	9	1	3	6	6	10	10	10	10	10	10	10	10	0	1	4	4	7	7	7	7	**206**
E10	4	7	3	8	8	2	5	9	9	9	9	1	3	6	6	10	10	10	10	10	10	10	10	1	0	4	4	7	7	7	7	**206**
E11	4	7	3	8	8	2	5	9	9	9	9	3	1	6	6	10	10	10	10	10	10	10	10	4	4	0	1	7	7	7	7	**206**
E12	4	7	3	8	8	2	5	9	9	9	9	3	1	6	6	10	10	10	10	10	10	10	10	4	4	1	0	7	7	7	7	**206**
E13	4	7	3	8	8	5	2	9	9	9	9	6	6	1	3	10	10	10	10	10	10	10	10	7	7	7	7	0	1	4	4	**206**
E14	4	7	3	8	8	5	2	9	9	9	9	6	6	1	3	10	10	10	10	10	10	10	10	7	7	7	7	1	0	4	4	**206**
E15	4	7	3	8	8	5	2	9	9	9	9	6	6	3	1	10	10	10	10	10	10	10	10	7	7	7	7	4	4	0	1	**206**
E16	4	7	3	8	8	5	2	9	9	9	9	6	6	3	1	10	10	10	10	10	10	10	10	7	7	7	7	4	4	1	0	**206**

FIG. 2 – Distance matrix

	B1	B2
C1	1	5
C2	1	5
C3	5	1
C4	5	1
D1	2	6
D2	2	6
D3	2	6
D4	2	6
D5	6	2
D6	6	2
D7	6	2
D8	6	2
E1	3	7
E2	3	7
E3	3	7
E4	3	7
E5	3	7
E6	3	7
E7	3	7
E8	3	7
E9	7	3
E10	7	3
E11	7	3
E12	7	3
E13	7	3
E14	7	3
E15	7	3
E16	7	3

Resource

B1	2,43
B2	2,43

Error

68

	B1	C1	C2	D1	D2	D3	D4	E1	E2	E3	E4	E5	E6	E7	E8	
B1	0	1	1	2	2	2	2	3	3	3	3	3	3	3	3	**34**
C1	1	0	3	1	1	4	4	2	2	2	2	5	5	5	5	**42**
C2	1	3	0	4	4	1	1	5	5	5	5	2	2	2	2	**42**
D1	2	1	4	0	2	5	5	1	1	3	3	6	6	6	6	**51**
D2	2	1	4	2	0	5	5	3	3	1	1	6	6	6	6	**51**
D3	2	4	1	5	5	0	2	6	6	6	6	1	1	3	3	**51**
D4	2	4	1	5	5	2	0	6	6	6	6	3	3	1	1	**51**
E1	3	2	5	1	3	6	6	0	1	4	4	7	7	7	7	**63**
E2	3	2	5	1	3	6	6	1	0	4	4	7	7	7	7	**63**
E3	3	2	5	3	1	6	6	4	4	0	1	7	7	7	7	**63**
E4	3	2	5	3	1	6	6	4	4	1	0	7	7	7	7	**63**
E5	3	5	2	6	6	1	3	7	7	7	7	0	1	4	4	**63**
E6	3	5	2	6	6	1	3	7	7	7	7	1	0	4	4	**63**
E7	3	5	2	6	6	3	1	7	7	7	7	4	4	0	1	**63**
E8	3	5	2	6	6	3	1	7	7	7	7	4	4	1	0	**63**

	B2	C3	C4	D5	D6	D7	D8	E9	E10	E11	E12	E13	E14	E15	E16	
B2	0	1	1	2	2	2	2	3	3	3	3	3	3	3	3	**34**
C3	1	0	3	1	1	4	4	2	2	2	2	5	5	5	5	**42**
C4	1	3	0	4	4	1	1	5	5	5	5	2	2	2	2	**42**
D5	2	1	4	0	2	5	5	1	1	3	3	6	6	6	6	**51**
D6	2	1	4	2	0	5	5	3	3	1	1	6	6	6	6	**51**
D7	2	4	1	5	5	0	2	6	6	6	6	1	1	3	3	**51**
D8	2	4	1	5	5	2	0	6	6	6	6	3	3	1	1	**51**
E9	3	2	5	1	3	6	6	0	1	4	4	7	7	7	7	**63**
E10	3	2	5	1	3	6	6	1	0	4	4	7	7	7	7	**63**
E11	3	2	5	3	1	6	6	4	4	0	1	7	7	7	7	**63**
E12	3	2	5	3	1	6	6	4	4	1	0	7	7	7	7	**63**
E13	3	5	2	6	6	1	3	7	7	7	7	0	1	4	4	**63**
E14	3	5	2	6	6	1	3	7	7	7	7	1	0	4	4	**63**
E15	3	5	2	6	6	3	1	7	7	7	7	4	4	0	1	**63**
E16	3	5	2	6	6	3	1	7	7	7	7	4	4	1	0	**63**

FIG. 3 – First epoch in nv-SOTA training. The left table shows the input items associated to $B1$ (dark shadow) and to $B2$ (light shadow). The other tables show the adaptation at these cells following this assignment.

	C1	C2	D1	D2	D3	D4	E1	E2	E3	E4	E5	E6	E7	E8	
C1	0	3	1	1	4	4	2	2	2	2	5	5	5	5	41
C2	3	0	4	4	1	1	5	5	5	5	2	2	2	2	41
D1	1	4	0	2	5	5	1	1	3	3	6	6	6	6	49
D2	1	4	2	0	5	5	3	3	1	1	6	6	6	6	49
D3	4	1	5	5	0	2	6	6	6	6	1	1	3	3	49
D4	4	1	5	5	2	0	6	6	6	6	3	3	1	1	49
E1	2	5	1	3	6	6	0	1	4	4	7	7	7	7	60
E2	2	5	1	3	6	6	1	0	4	4	7	7	7	7	60
E3	2	5	3	1	6	6	4	4	0	1	7	7	7	7	60
E4	2	5	3	1	6	6	4	4	1	0	7	7	7	7	60
E5	5	2	6	6	1	3	7	7	7	7	0	1	4	4	60
E6	5	2	6	6	1	3	7	7	7	7	1	0	4	4	60
E7	5	2	6	6	3	1	7	7	7	7	4	4	0	1	60
E8	5	2	6	6	3	1	7	7	7	7	4	4	1	0	60

FIG. 4 – Initialization of $B1$'s children.

left and right descendants.

Three values are stored at this point : i) the mean value of the distances between $B1$ and each of the sequences associated to it, namely, the local resource value of $B1$, $R_{B1} = \frac{1}{14} \cdot [d(B1,C1) + d(B1,C2) + d(B1,D1) + d(B1,D2) + d(B1,D3) + d(B1,D4) + d(B1,E1) + d(B1,E2) + d(B1,E3) + d(B1,E4) + d(B1,E5) + d(B1,E6) + d(B1,E7) + d(B1,E8)] = \frac{34}{14} = 2.43$; ii) the local resource value of $B2$, $R_{B2} = 2.43$; and iii) the actual error of the network, $E_1 = 14 \cdot R_{B1} + 14 \cdot R_{B2} = 68$. Then, letting the initial network error be, say $E_{initial} = 100,000$, we check the relative error decrease, $\epsilon = \left| \frac{E_1 - E_{initial}}{E_{initial}} \right| = 0.99$. Since this value is bigger than a given threshold, say $\epsilon_0 = 0.00001$, another epoch is performed.

We next find the input items associated to $B1$ and $B2$, respectively, which are of course the same as in the previous epoch (see left table of Figure 3). Then the updating process goes as before (see the right tables in Figure 3) and $B1$ and $B2$ are again the two cells of the tree. The three values stored now are i) $R_{B1} = 2.43$, ii) $R_{B2} = 2.43$ and iii) $E_2 = 68$. Since now the relative decrease of the error $\epsilon = \frac{E_2 - E_1}{E_1}$ equals 0, the first cycle ends.

Since all local resource values are bigger than a given threshold, say $R_0 = 0.01$, i.e., $\min(R_{B1}, R_{B2}) = 2.43 > 0.01$, two new descendants are generated from the cell having the highest resource value, say $B1$. This changes its status from cell to node and will no longer be compared with input items. The sum of distances among all items in the set of elements associated to $B1$, see the last column in Figure 4, is calculated to initialize $B1$'s children, and the minimum sum is found at $C1$ and $C2$. Hence, the first epoch of the second cycle starts with A as the root node, $B1$ and $B2$ as its descendants and $C1$ and $C2$ as the descendants of node $B1$.

Next, the input items associated to $B2$, $C1$ and $C2$ are collected, see the left table in Figure 5. Each cell is updated as the generalized median of the set composed by the items associated to it and itself, that is, i) $B2$ is updated as itself (see the right lower table in Figure 3), ii) $C1$ is updated as itself (see the right upper table in Figure 5) and iii) $C2$ is updated as itself (see the right lower table in Figure 5). Hence, the first epoch of the second cycle finishes with A as the root node, $B1$ and $B2$ as its descendants and $C1$ and $C2$ as the descendants of node $B1$.

Once the epoch finishes, the local resource values together with the actual network error are stored, i.e., i) $R_{B1} = 2.43$, ii) $R_{C1} = 1.67$, iii) $R_{C2} = 1.67$ and iv) $E_3 = 54$. Since the relative decrease of the error, $\epsilon = \left| \frac{E_3 - E_2}{E_2} \right| = \left| \frac{54 - 68}{68} \right| = 0.21$ is bigger than the given threshold ϵ_0, another epoch is needed. Nevertheless, we will obtain a new network error of $E_4 = 54$, since no cell has been changed during the previous updating process. Hence, as before the tree at the end of the second epoch will be exactly as that at the end of the first epoch, and

	B2	C1	C2
C3	1	6	6
C4	1	6	6
D1	6	1	4
D2	6	1	4
D3	6	4	1
D4	6	4	1
D5	2	7	7
D6	2	7	7
D7	2	7	7
D8	2	7	7
E1	7	2	5
E2	7	2	5
E3	7	2	5
E4	7	2	5
E5	7	5	2
E6	7	5	2
E7	7	5	2
E8	7	5	2
E9	3	8	8
E10	3	8	8
E11	3	8	8
E12	3	8	8
E13	3	8	8
E14	3	8	8
E15	3	8	8
E16	3	8	8

	C1	D1	D2	E1	E2	E3	E4	
C1	0	1	1	2	2	2	2	10
D1	1	0	2	1	1	3	3	11
D2	1	2	0	3	3	1	1	11
E1	2	1	3	0	1	4	4	15
E2	2	1	3	1	0	4	4	15
E3	2	3	1	4	4	0	1	15
E4	2	3	1	4	4	1	0	15

	C2	D3	D4	E5	E6	E7	E8	
C2	0	1	1	2	2	2	2	10
D3	1	0	2	1	1	3	3	11
D4	1	2	0	3	3	1	1	11
E5	2	1	3	0	1	4	4	15
E6	2	1	3	1	0	4	4	15
E7	2	3	1	4	4	0	1	15
E8	2	3	1	4	4	1	0	15

FIG. 5 – Input items associated to $B2$ (darker shadow), $C1$ (lighter shadow) and $C2$ (medium shadow) at the beginning of the first epoch of the second cycle.

	C3	C4	D5	D6	D7	D8	E9	E10	E11	E12	E13	E14	E15	E16	
C3	0	3	1	1	4	4	2	2	2	2	5	5	5	5	41
C4	3	0	4	4	1	1	5	5	5	5	2	2	2	2	41
D5	1	4	0	2	5	5	1	1	3	3	6	6	6	6	49
D6	1	4	2	0	5	5	3	3	1	1	6	6	6	6	49
D7	4	1	5	5	0	2	6	6	6	6	1	1	3	3	49
D8	4	1	5	5	2	0	6	6	6	6	3	3	1	1	49
E9	2	5	1	3	6	6	0	1	4	4	7	7	7	7	60
E10	2	5	1	3	6	6	1	0	4	4	7	7	7	7	60
E11	2	5	3	1	6	6	4	4	0	1	7	7	7	7	60
E12	2	5	3	1	6	6	4	4	1	0	7	7	7	7	60
E13	5	2	6	6	1	3	7	7	7	7	0	1	4	4	60
E14	5	2	6	6	1	3	7	7	7	7	1	0	4	4	60
E15	5	2	6	6	3	1	7	7	7	7	4	4	0	1	60
E16	5	2	6	6	3	1	7	7	7	7	4	4	1	0	60

FIG. 6 – Initialization of $B2$'s children.

the relative decrease of the error will be $\epsilon = 0$.

Thus, the second cycle finishes. Since $B2$ has the highest resource value (and this value is bigger than the threshold R_0), it will change its status from cell to node and two descendants will be generated from it. Figure 6 shows the initialization of $B2$'s descendants, that is, $C3$ and $C4$, since they have the smallest sum of distances (41) among all items of the elements associated to $B2$. The first epoch of the third cycle will begin with $C1$, $C2$, $C3$ and $C4$ as cells. Continuing this process, the tree of Figure 1 will be obtained as desired.

6 Real proteins

Now we will analyze nvSOTA using real protein sets. Kohonen and Somervuo [Kohonen and Somervuo (2002)] have processed a very large database of 77,977 protein sequences, obtained from the SWISS-PROT database, release 37 [Bairoch and Apweiler (1999); Somervuo and Kohonen (2000)], publicly available at `http ://www.expasy.ch/sprot`. The computation of similarities between protein sequences was determined using version 3.0 of the FASTA program package [Pearson (1999)], with the *ktup* parameter set to 2 and the amino acid substitution scores taken from the BLOSUM50 matrix. A 30-by-20-unit hexagonal symbol-SOM was trained for 20 batch cycles, with a neighborhood radius of 1. Due to the high computing cost incurred if all protein sequences are used as inputs at each batch cycle, Kohonen and Somervuo picked up a random sample of size 6,000 for each batch cycle. One final training

cycle was carried out with all data. Hence, the computing load is reduced to less than 10% and the statistical accuracy of the algorithm is not deteriorated [Kohonen and Somervuo (2002)]. The final map was labelled using the SWISS-PROT identifiers. To characterize the quality of the obtained map, Kohonen and Somervuo showed the clustering of some known protein families together with the projections of the most frequent classes [Kohonen and Somervuo (2002)].

We have processed another data set for analysis. This consists of the 1,758 human protein sequences stored in the SWISS-PROT database, release 19.0. This data set was first used by [Ferrán et al. (1994)] to test the method they proposed in [Ferrán and Ferrara (1991)], namely, a method based on the Kohonen's unsupervised learning algorithm trained with the dipeptide composition of the proteins. We have extracted the Similarity Matrix of this data set using version 3.0 of FASTA [Pearson (1999)], with a *ktup* of 2 and a BLOSUM50 matrix for the amino acid substitution.

7 Relationship with other self-generating neural tree methods

We now discuss the differences between the previous nv-SOTA and other related methods. First, we emphasize that, to the best of our knowledge, nv-SOTA is the first growing architecture capable of processing nonvectorial information. We also note the advantages of our approach when compared to the Self-Generating Neural Tree (SGNT) proposed by [Wen et al. (1992)]. This is a method to learn neural trees where not only the weights but also the structure of the network, including the number of neurons and the connections between them are all learned from the training set. The main disadvantages of the SGNT algorithm, compared to nv-SOTA, are the following : i) both cells and nodes are compared to the input sequences, so more computing time is required ; ii) once the algorithm converges, a pruning phase is necessary to obtain an optimal tree ; and, iii) besides pruning, the size of the generated tree must be artificially reduced to have practical use, since the generated network may still have an unacceptable big size. The optimization and pruning phase is developed following McKusick and Langley's method [McKusick and Langley (1991)]. Here, the concepts of *Horizontally Well Placed* (a neuron n is HWP if $d(n, n_p) \leq d(n, n_s)$, $\forall n_s \in \{$chil neuron set of $n_p\}$, where n_p is the parent of n), *Vertically Well Placed* (a neuron n is VWP if $d(n, n_p) \leq d(n, n_g)$, where n_g is the parent of n_p) and *Well Organized* (a neural tree is WO if all of its neurons are both HWP and VWP) are used to optimize the tree and turn it into a better shape for clustering and classification tasks [Wen et al. (1992)].

The artificial reduction is achieved using Quinlan's method [Quinlan (1987)]. Here, the SGNT is tested against the full training set, and the number of errors incurred in this test is accumulated and recorded at each neuron. For each neuron m_i, the following quantity is stored, $\varepsilon + \frac{1}{1+\beta} \leq (\varepsilon' + \frac{\beta}{1+\beta}) \cdot \sigma(\varepsilon' + \frac{\beta}{1+\beta})$, where ε and ε' are the number of errors occurred at m_i in the pruned and unpruned network, respectively, β is the number of children of m_i and $\sigma(\cdot)$ is the standard deviation function. Then, every subnetwork rooted by neuron m_i that satisfies the previous condition may be pruned away (but m_i itself is still kept) and the simplified network will have the same accuracy as it had prior to simplification [Wen et al. (1992)].

The need for an optimization step to make the network in a better shape together with the posterior simplification of the obtained tree may lead to computational problems when the

training data set becomes large enough [Wen et al. (1992)]. Furthermore, the interpretation of the resulting SGNT as a phylogenetic tree is not proved yet.

8 Summary and Conclusions

We have presented a new growing algorithm for clustering symbolic data based on trees. The only information needed is a similarity matrix, no vectorial representation is used whatsoever. The new algorithm has been shown to perform well in the case of real proteins.

Références

Andrade, M. A., Casari, G., Sander, C. and Valencia, A. (1996). Classification of protein families and detection of the determinant residues with a self-organizing neural network.

Bairoch, A. and Apweiler, R. (1999). The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999, *Nucleic Acids Research*, **27** : 49–54.

Casari, G., Sander, C. and A., V. (1995). Functional residues predicted in protein sequence space, *Nat Struc Biol*, **2** : 171–178.

Clark, A. and Lanigan, C. (1993). Prospects for estimating nucleotide divergence with rapds, *Mol. Biol. Evol.*, **10** : 1096–1111.

Dopazo, J. and Carazo, J. (1997a). Phylogenetic reconstruction using an unsupervised growing neural netrwork that adopts the topology of a phylogenetic tree, *Journal of Molecular Evloution*, **44** : 226–233.

Dopazo, J. and Carazo, J. (1997b). Sota algorithm, *http ://www.cnb.uam.es/∼bioinfo/Software/sota*.

Ferrán, E. A. and Ferrara, P. (1991). Topological maps of protein sequences, *Biological Cybernetics*, **65** : 451–458.

Ferrán, E. A. and Ferrara, P. (1992). Clustering proteins into families using artificial neural networks, *Cambios*, **8(1)** : 39–44.

Ferrán, E. A. and Pflugfelder, B. (1993). A hybrid method to cluster protein sequences based on statistic and artificial neural networks, *Comput. Appl. Biosci.*, **9** : 671–680.

Ferrán, E. A., Pflugfelder, B. and Ferrara, P. (1994). Self-organized neural maps of human protein sequences, *Protein Science*, **3** : 507–521.

Fritzke, B. (1994). Growing cell structures : a self organizing network for unsupervised and supervised learning, *Neural Networks*, **7** : 1141–1160.

Inoue, H. and Narihisa, H. (2003). Efficiency of self-generating neural networks applied to pattern recognition, *Mathematical and Computer Modelling*, **38 (11-13)** : 1225–1232.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, **43** : 59–69.

Kohonen, T. (1985). Median strings, *Pattern Recognition Letters*, **3** : 309–313.

Kohonen, T. (1990). The self-organizing map, in *Proc. IEEE*, vol. 78, pp. 1464–1480.

Kohonen, T. (1995). *Self-Organizing Maps*, Berlin : Springer-Verlag.

Kohonen, T. (1996). Self-organizing maps of symbol strings, in *Technical Report A42*, Laboratory of Computer and Information Science, Helsinki University of Technology, Finland.

Kohonen, T. and Somervuo, P. (1998). Self-organizing maps of symbol strings, *Neurocomputing*, **21** : 19–30.

Kohonen, T. and Somervuo, P. (2002). How to make large self-organizing maps for nonvectorial data, *Neural Networks*, **15** : 945–952.

McKusick, K. and Langley, P. (1991). Constrains on tree structure in concept formation, in *Proceedings of IJCAI'91, Sydney*, edited by Kaufmann, M., vol. 2, pp. 810–816.

Mulier, F. and Cherkassky, V. (1995). Self-organization as an iterative kernel smoothing process, *Neural Computation*, **7** : 1165–1177.

Nei, M. and Li, W. (1979). Mathematical model for studying genetic variation in terms of restriction endonucleases, in *Proc. Natl. Acad. Sci. USA*, vol. 76, pp. 5269–5273.

Pearson, W. (1999). The FASTA program package, `ftp ://ftp.virginia.edu/pub/fasta`.

Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence comparison, in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, pp. 2444–2448.

Quinlan, J. R. (1987). Simplifying decision trees, *Int. J. Man-Machine Studies*, **27** : 221–234.

Somervuo, P. and Kohonen, T. (2000). Clustering and visualization of large sequence databases by mean of an extension of the self-organizing map, in *Proceedings of the Discovery Science*, edited by Arikawa, S. and Morishita, S., pp. 76–85, Berlin : Springer.

Wen, W., Jennings, A. and Liu, H. (1992). Learning a neural tree, *International Joint Conference on Neural Networks, IJCNN'92, Beijing, China.*, **2** : 751–756.