Appl. Math. and Comp. Sci., 1994, vol.4, No.3, 371–395

A NEURAL NETWORK APPROACH TO GENOME SEQUENCE ALIGNMENT

ROMAN W. ŚWINIARSKI*, D. WAAGEN*

A technique for the alignment of genome sequences based on an adaptive non-linear dynamic neural network is proposed. We present an extension to the fixed weight Hopfield neural network, creating a non-linear dynamic neural network, with weights values adaptively changing during neural processing. The weights of the proposed neural network are not fixed during the processing, but are continuously updated to achieve the minimal alignment according to the minimal sequence distance criterion. The binary coding of the alignment process has been adopted from the original work of Sellers (1979) to adaptive dynamic neural processing. The behaviour of the proposed neural network is modelled by computer simulation and the corresponding results are discussed.

1. Introduction

One of the main objectives of nucleic acid and protein sequence analysis is discovering significant patterns and interpreting them in relation to the course of DNA processing, polypeptyde folding, protein biochemical function, and evolutionary development related to each of these levels. Similarities and differences detected among the sequences can be used among others for phylogenetic classifications, analogy versus homology evaluation, and understanding of evolutionary events of substitution, insertion, deletion and recombination (Waterman, 1989; Sankoff and Kruskal, 1983; von Heijne, 1989; Weir, 1983; Lesk, 1988).

Due to the enormous possibilities of combinations and size of existing genome data bases the new effective techniques for genome sequence analysis and recognition are needed. There are now thousands of DNA sequences with millions of bases in the GenBank, EMBL and other international huge genome data bases (Coulson *et al.*, 1987; Dayhoff *et al.*, 1978). Thousands of these sequences contain coding regions that can be translated into aminoacid sequences and may be considered as a supplement to directly determined protein sequences existing in the Protein Identification Resource (PIR) data-base.

The fundamental works of biological sequence metrics (Sellers, 1974; Waterman *et al.*, 1976; Waterman, 1989), the dynamic programming concept (Bellman, 1957), sequence analysis and transformations (Delahaye, 1988; Wimp, 1981), as well as finite automata and regular expressions (Hopcroft and Ulman, 1979) may be considered as

* Department of Mathematical Sciences, Computer Science, San Diego State University, San Diego, USA, (and Warsaw University of Technology, Warszawa, Poland)

a basis for the biological sequence processing. In the mathematical analysis of macromolecular sequences one of the most demanding and developed areas is the comparison and characterizing of sequence differences. A fundamental problem is associated with the development of efficient optimal sequence alignment algorithms and corresponding fast computer programs. The sequence alignment, due to a large size of sequences and enormous number of possible insertions, deletions or substitutions of bases in the sequences (growing exponentially with a number of bases being considered) belongs to the class of hard NP-complete combinatorial optimization problems.

So far most of sequence alignment algorithms have been dynamic programming algorithms. Originally introduced by Needleman and Wunsch (1970), the application of the dynamic programming technique (Bellman, 1957) in sequence alignment assumed the maximization of a similarity measure (maximizing the amount of matching in an alignment). Sellers (1974) introduced the minimization of a distance measure, but it has been shown (Smith *et al.*, 1981) that similarity maximizing and distance minimizing are essentially equivalent. Both techniques allow for the inclusion of gaps (by insertion or deletion operations), but the weighting (penalty) functions considered only the existence of single sequence element gap per sequence. Over the last decade many different sequence alignment techniques both for a pair of sequences and for multiple sequences have been developed (Waterman, 1989). Waterman *et al.* (1976) have introduced a generalization of the treatment of gap weights by including gaps of more than one sequence element in length with the only restriction that the weight function must be monotonically increasing with the gap length. Then Sellers (1979) proposed the idea of separating the weights assigned to the gaps at the end of sequences from those in the interior of a sequence, why particularly suggesting that the terminal gap should be unweighted. Many sensitive methods for protein sequence alignment use the PAM–250 matrix (Dayhoff *et al.*, 1978) of similarity scores for pairwise amino acid matching. Dayhoff's matrix bias can be shown to produce a linear gap weighting, which makes it possible to find a solution in time proportional to the square of sequence length, unlike the cubic time required for weights as a non-linear function of gap lengths. Fitch and Smith (1983) as well as Altschul and Erickson (1986) studied assigning the penalty for gap in alignment as a linear function of the inserted or deleted residues. The early sequence alignment algorithms were designed for the single pair of sequences. Fitch (1970) showed how to align the third sequence, and Murata *et al.* (1985) extended the pioneering dynamic programming algorithm of Needleman and Wunsch (1970) to operate on three sequences. Recently several methods have been proposed for multiple sequence alignment. The techniques of Sankoff (1975), Felsenstein (1982), Sobel and Martinez (1986), Queen *et al.* (1982) and Carillo and Lipman (1988) are well applicable in the situation when the sequence homology there is known to be strong. Freedman (1984) proposed an algorithm for similarity measures with length independent gap penalties. Waterman *et al.*, (1984) proposed an algorithm for finding unknown patterns that occur imperfectly above preset frequency. Gotoh (1986) proposed some efficient algorithms for three sequences alignment using a traceback procedure. Bacon and Anderson (1986) introduced a technique for aligning segments of several sequences at once. Miller and Myers (1988) analysed sequence comparison with concave weighting function. Zucker and

Somorjai (1989) extended the use of dynamic programming algorithm into molecular sequence comparison in three dimensions. Altschul and Lipman (1989) have extended the concept of Carillo and Lipman's algorithm (1988) of alignment to the definition of multiple alignment cost as the cost of an evolutionary tree.

Recently, a few attempts have been made to develop a rapid alignment method to facilitate sequence data base search (Fickett, 1984; Lipman and Pearson, 1985; Coulson *et al.*, 1987; Arbanel *et al.*, 1984). These algorithms extend the classical distance minimizing dynamic programming algorithms for sequence alignment to the alignment of protein molecules in three dimensions using a linear penalty. The concept of match density in pattern similarities in genetic sequences was introduced by Goad and Keneisha (1982) and was extended by Sellers (1984) as a continuation of the dynamic programming technique, proposed by Needleman and Wunsch (1970). There are also several works on sequence alignment which use statistical procedures. Fitch (1970) and have used a modified likelihood-ratio test to estimate the probabilities of best scores. Ohya (1989) employed the mutual entropy for the analysis of molecular evolution.

The majority of sequence alignment algorithms belong to the class of the sequential digital dynamic programming techniques. All sequence alignment algorithms are time consuming, particularly for multiple sequence alignment, and critically depend on the sequences length. For example, dynamic programming methods take time and storage of $O[(2n)^l]$ to compare l sequences of length n. Some algorithms mentioned above, provide some reduction of the computation burden, however a computational complexity of classical sequential numerical algorithms is very high, and alignment of the sequences with large length and matching the sequences in a huge data bank create a basic computational difficulty.

There are also limited attempts in the application of formal languages, regular expressions and automata to sequence processing, as well as the application of artificial intelligence concepts (perceptron) (Stormo *et al.*, 1982; Minsky and Papert, 1969). These methods are related to the statistical linear discriminant techniques (Gnanadesikan, 1977). Quian and Sejnowski (1987) applied a feedforward backpropagation neural network for predicting the secondary structure of globular proteins. Bohr *et al.* (1989) applied a static neural network for semi-empirical studies of pattern matching between the primary and secondary structures of proteins.

In the paper we propose a technique for the alignment of genome sequences based on an adaptive non-linear dynamic neural network. We propose an extension to the fixed weight Hopfield neural network (Hopfield and Tank, 1985; Świniarski, 1991), creating a non-linear dynamic neural network, with network weights values adaptively changing during neural processing. The weights of the proposed neural network are not fixed during the processing, but are continuously updated to achieve the minimal alignment according to the minimal sequence distance criterion. The binary coding of the alignment process, used in the proposed dynamic neural network, has been adopted from (Sellers, 1979) to adaptive dynamic neural processing. The behaviour of the proposed neural network is modelled by computer simulation, and the results are discussed.

In the sequel, we will use the terms "node" and "neuron" interchangeably. A legal path in the lattice consists of a path that traverses all characters of both sequences. A legal path starts at the upper left node and finishes at the lower right one of the alignment lattice. The internal nodes represent intermediate steps in an alignment process. A diagonal edge denotes a continuation or a mutation. A vertical edge denotes a deletion, and a horizontal one an insertion transformation on a single position of sequences.

Visually, we will consider as legible only the sequences having no positive "gradient" over coordinates of the alignment lattice. The alignment lattice clearly illustrates the possibility of minimal alignment solution by the dynamic programming method. The final result of the alignment process may be interpreted on the alignment lattice as a binary image of states of nodes (i, j) $(i = 1, 2, ..., k;\ j = 1, 2, ..., l)$. The node with state '1' will indicate the node belonging to the transformation path which results in an alignment, while the one with state '0' will indicate other not applied transformations (Fig. 3).

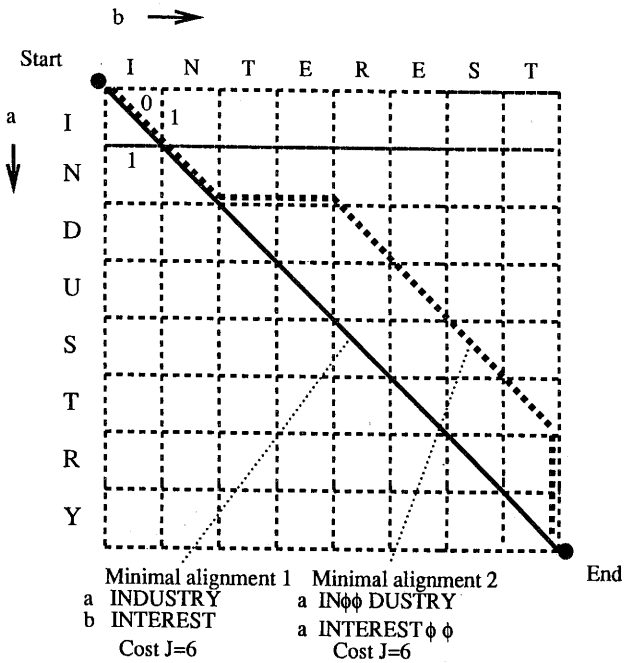

Fig. 3. Example of binary coding of alignment.

On account of the combinatorial complexity of the dynamic programming techniques, an approach was sought that would perform the optimizations based on interactions between the elements of the alignment lattice. Indeed, the work of Hopfield and Tank (1985; 1986) provided the intellectual inertia for this research. Hopfield and Tank illuminated the possibility of using an interconnected set of analog processors to solve optimization problems, including the traveling-salesman problem.

We have proposed the dynamic, Hopfiled type neural network for solving the above-stated sequence alignment problem (Fig. 4).

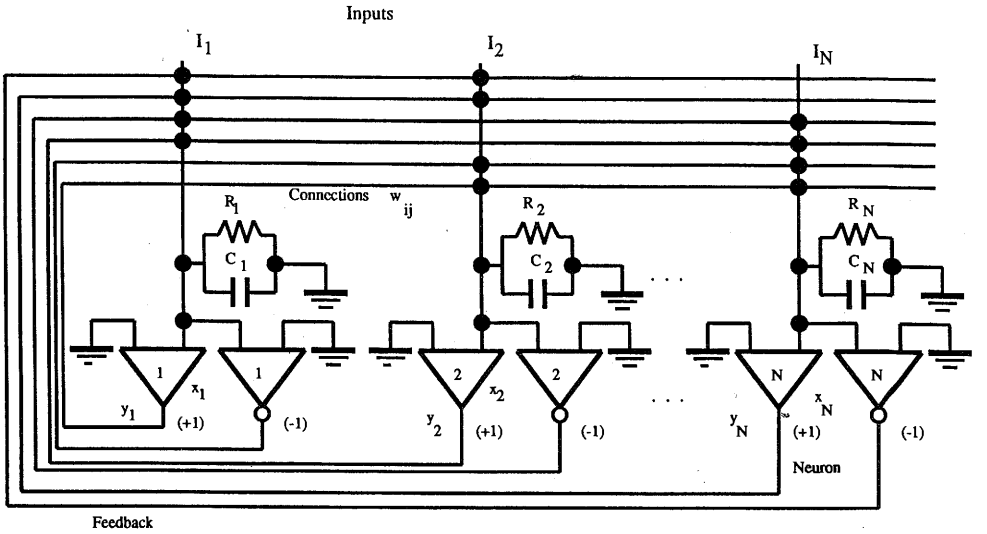

Fig. 4. Dynamic Hopfield neural network.

3. Introduction to the Hopfield Neural Network

The Hopfiled dynamic neural network is the system of N dynamic (with internal memory) neurons interconnected through the weights. From the topological point of view the Hopfiled dynamic neural network consists of one layer of neurons. The output of each neuron is fed back to the inputs of all other neurons. The strength of the feedback connections is modelled by the connection weights. The weight w_{ij} denotes the connection strength from the output of the j-th neuron to the input of the i-th one.

Every neuron is considered as a simple first-order input-output dynamic system, with the internal state (memory) $x_i(t)$ $(i = 1, ..., N)$ and influencing the environment via the output signal $y_i(t)$ $(i = 1, 2, ..., N)$. All the internal states of the network may be denoted by the state vector $\boldsymbol{x}(t) = (x_1(t), x_2(t), ..., x_N(t))^T \in \mathbb{R}^N$ and the output vector $\boldsymbol{y}(t) = (y_1(t), y_2(t), ..., y_N(t))^T \in \mathbb{R}^N$. The input to the neuron (control input) is defined as a weighted sum of outputs of all the neurons of the network with additional input bias I_i

$$u_i = \sum_{j=1}^{N} w_{ij} y_j(t) + I_i, \qquad i = 1, 2, ..., N \tag{4}$$

The signals y_j $(j = 1, 2, ..., N)$ come as a feedback from the neuron outputs, while signals I_i are external inputs to the network (denoted as a vector $\boldsymbol{I} \in \mathbb{R}^N$).

The following model describes the behaviour of the dynamic "motion" of the neuron in the dynamic neural network

$$\frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = -\frac{1}{\tau}x_i(t) + \sum_{j=1}^{N} w_{ij}y_j(t) + I_i, \quad i = 1, 2, ..., N \tag{5}$$

$$x_i(t_0) = \xi_i, \quad \tau = RC \tag{6}$$

The sigmoid (continuous) output activation function $y_i(t) = f_h(x_i(t))$ describes the relation between the neuron internal state (representing the internal memory of neuron) and a neuron output signal. The activation function should be a monotonically increasing and bounded function. Monotonicity guarantees a response without a local maxima or minima, while bounding the response protects the nodes from their mutual excitation into a resonant state. A shifted hyperbolic tangent function was chosen because it fulfils these criteria. The formula for the function is as follows:

$$y = f(x, \beta) = \frac{1}{2}\left(1 + \tanh\left(\frac{x}{\beta}\right)\right) \tag{7}$$

The domain of this function is the interval $(-\infty, \infty)$, while the corresponding range is $[0, 1]$. The parameter β modifies the slope of the response function, allowing the function to vary from a smooth response to approximating a hard delimiter. This function describes the natural saturation of the neuron output. The output activation function with "steep" characteristic of a graph limits the neural network outputs (corresponding to the stable equilibrium states) to the corners of N-dimensional hypercube. Since the neuron output activation functions are non-linear, the whole neural network is a non-linear dynamic system with the internal memory.

As a matter of fact, the neural network under consideration processes the information in a concurrent and distributed way. The evolving of the internal states of the network depends on the initial conditions $\boldsymbol{x}(t_0)$ at the initial time t_0 and external input $\boldsymbol{I}$. All neurons update their states asynchronously as in electric circuits. The output saturation functions limit the network internal state to the interior of an N-dimensional hypercube.

In the neuron (network node) paradigm, the voltage of each neuron (node) multipled by the corresponding connection weight constitutes one of the inputs into another neuron. Let us assume that the output activation function of the neuron f_h produces the output from the interval $[0, y_{max}]$. The voltage values of the j-th neuron output y_j multipled by the connection weight w_{ij} of the i-th neuron is considered as one of the inputs to the i-th neuron. Since voltage values of the neuron output are positive (ranging from 0 to $y_{\max}$), positive weights values will cause excitation of the neuron (rise of neuron voltage). This input with positive weight will be called the *excitatory input to the neuron.* On the other hand, negative values of the weights will inhibit the input voltage influence (lower neuron voltage). The inputs with negative weights are called the *inhibitory inputs to the neuron.*

The Hopfield dynamic neural network with N dynamic neurons having "steep" sigmoid output activation functions, is based on the paradigm of natural tendency

of the evolving dynamic system towards the internal state equilibrium in one of 2^N corners

$$(y_{1,\nu}, y_{2,\nu}, ..., y_{i,\nu})^T, \qquad \nu \in \{\min, \max\} \tag{8}$$

of an N-dimensional hypercube with the boundaries

$$y_{i,\nu}, \quad \nu \in \{\min, \max\}, \quad i = 1, 2, ..., N \tag{9}$$

This equilibrium state is associated with the minimal energy defined in the system. For the network with the neurons having steep (high gain) continuous sigmoid activation function we can write the following Liapunov energy function for the network

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} y_i(t) y_j(t) - \sum_{i=1}^{N} y_i I_i \tag{10}$$

As it was shown by Hopfield (1985), the equations of network motion with symmetric connection $w_{ij} = w_{ji}$ always lead to convergence to the *stable states* x_i^s and, of course, "stable outputs" y_i^s (since output activation function is increasing). This results from the fact that the derivative of the energy equation with respect to time $\partial E/\partial t$ is non-negative or zero for all t for any change of the states. For the steep neurons output activation function (with the high gain) the stable states of network guarantee the local minima of the energy function. These stable states $y^s = (y^s_{1,\nu}, y^s_{2,\nu}, ..., y^s_{N,\nu})^T$ ($\nu \in \{\min, \max\}$) correspond to the state location in the interior of an N-dimensional hypercube with 2^N corners. These states may act as "binary coded" results. The neural network feature of tending to the "binary" stable states which correspond to the minima of network energy function is the basis for analog computations. The stable states (results) may by "programmed" by the preselection of the weights values. The specific weights values depend on the computational problem.

To ensure the stability of the solution, several characteristics or properties of the interconnection matrix $\boldsymbol{W}$ are required. One property is that the interconnection matrix, $\boldsymbol{W}$, representing the reinforcement or inhibitionary relationship between two processors, is symmetric, i.e. $w_{ij} = w_{ji}$ for all i, j. Another property is that the diagonal elements are equal to zero (i.e. $w_{ii} = 0$ for all i). The elements of this matrix are derived from the energy equation, and are fixed or constant in value.

The fixed nature of the weights matrix in the Hopfield neural network is a serious limitation. In fact, we assert that, since the elements of interconnection matrix $\boldsymbol{W}$ are constants and cannot change with time but are fixed, the Hopfield network cannot consistently represent the legal paths in an alignment lattice. This limitation is illustrated in the following section.

4. Neural Network for Genome Sequence Alignment

As it has been discussed earlier, the transformation events of deletion, and matching or substitution, are represented by legal paths on an alignment lattice. The legal paths are shown graphically in Fig. 5. Due to the triangle inequality, an insertion followed by a deletion or vice versa has a higher cost than an equivalent substitution, and constitutes an illegal path.

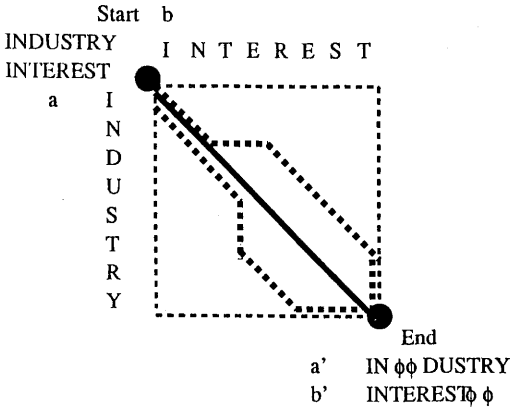

Fig. 5. Binary coding and legal paths.

4.1. Dynamic Neural Network for Genome Sequences Alignment

We propose a dynamic neural network for genome sequences alignment. Even though from the topological point of view the proposed network consists of one layer of neurons, for the design logical purposes we will consider this network as a rectangular lattice. This lattice is similar to the alignment lattice described previously. For the alignment of the sequences $\boldsymbol{a}$ and $\boldsymbol{b}$ of lengths k and l, respectively, the neural network will consist of $k+1$ rows and $l+1$ columns of neurons.

We will denote by x_{ij} the internal state and by y_{ij} the output of the ij-th neuron, corresponding to the node in row i and column j of the alignment lattice, Similarly, we will denote by $x_{(i-1),(j-1)}$ and $y_{(i-1),(j-1)}$ the state and the output of the neuron corresponding to the node in row $(i-1)$ and column $(j-1)$ of the alignment lattice. The connection weight from the output y_{ij} of the neuron ij to the input of the neuron $(i-1),(j-1)$ will be denoted by $w_{(ij);(i-1),(j-1)}$.

4.2. Inadequacy of the Hopfield Paradigm for Genome Sequence Alignment

If the node y_{ij} is not active, then the path $y_{(i+1),j}\ y_{(i+1),j}$ would constitute a legal insertion. To enable the legal insertion, $w_{ij,(i+1)(j+1)}$ should have a positive value, so that if y_{ij} is active, then it will excite the nodes $y_{(i+1),(j+1)}$, $y_{(i+1),j}$, and the formation of this legal path.

If a fragment of a path selected from the ij-th node is a deletion operation, then both y_{ij} and $y_{(i+1),j}$ are active. The weight $w_{ij,(i+1)(j+1)}$ would have to suppress or inhibit the $y_{(i+1),(j+1)}$ node from becoming active, or, otherwise, an illegal path y_{ij}, $y_{(i+1),j}$, $y_{(i+1),(j+1)}$ can be formed. To inhibit the node $y_{i+1,j+1}$, the value of $w_{i+1,j+1}$ would be negative.

Obviously, the weight $w_{ij,(i+1)(j+1)}$ cannot be preselected as in the Hopfiled neural network, since preselection prohibits the formation of all possible paths on the alignment lattice given *a priori* the input data. Thus, as shown in the example above,

to enable all paths in the alignment lattice to be possible, a weight must be able to assume both positive and negative values.

The contradiction stems from the fact that the state of inhibition or excitation of a weight between two nodes in an alignment lattice depends not only on the state of the nodes themselves, but also on the current state of the neighbouring nodes. A constant weight matrix $\boldsymbol{W}$ cannot contain the information necessary to support all legal paths in an alignment lattice. Therefore, the matrix cannot be statically defined before execution, but is a dynamic-valued matrix whose elements change their values according to the current state of the network and the input.

5. Genome Sequence Neural Network

To solve the genome sequence alignment problem, the dynamic nature of the weight matrix forces one to abandon the Hopfield paradigm as a viable solution. We propose a new network, without the constraints imposed by the Hopfield criteria. This network has non-linear weight values that are a function of the current state of the network and input sequences. Unlike the Hopfield weight matrix, the genome sequence matrix is no longer symmetric, i.e. $w_{ij} = w_{ji}$. Without the criteria of symmetry and constant weight values, the Hopfield model does not guarantee stability. The genome sequence network, and the functions which define local and global weight values (which will be discussed in more length below), were therefore designed to produce stable results. This is not to say that stability is mathematically guaranteed, however, the chance of oscillation is small. In the genome sequence network, oscillation is possible only when two viable paths in the lattice have exactly the same node values at exactly the same time. This chance is made negligible, not only by the nature of non-linear functions modifying the weights, but also due to the small randomization of input voltages occurring at the beginning of the system.

The genome sequence network shares many properties with the Hopfield network, including the use of the sigmoidal activation function, as well as the equations describing the state of the network, and the updating of nodes in the network. However, the constants found in previous equations are now replaced with functions. These functions provide the necessary ingredient to allow the system to generate the optimal paths in the alignment lattice.

A description of the genome sequence neural network follows. The weight matrix is described more fully, with the corresponding equations describing local and global interactions. Then, the activation function is discussed, which is the function applied to the input of a node, generating the nodes (bounded) output. The following section describes how the network is controlled by the input sequences, and shows that the matching of sequence character elements ultimately controls the behaviour of the network and the path generated on the alignment lattice. The update formula for a node is discussed, providing a basis for understanding the evolution of a node over time. Finally, an overview of the algorithm is given.

5.1. The Dynamic Weights Matrix

To allow a network to describe the legal paths in the alignment lattice, the weight matrix $\boldsymbol{W}$ needs to be a dynamic structure $\boldsymbol{W}(t)$. The value of the weight $w_{ij,kl}(t)$, which represents the connection strength between the output of the neuron kl and the input of the given neuron ij, must be adaptively changed during the network processing. In the proposed approach, the value of each weight $w_{ij,kl}(t)$ is a function of the current value of the nodes (neurons outputs) in the alignment lattice. We propose the following way of weight changing in time during the processing

$$w_{ij,kl}(t) = f_l(\boldsymbol{v}_{ij;\text{neighb}(ij)}(t)) + f_g(y_{kl}(t)) \tag{11}$$

where ij and kl cover indices of all neurons in the alignment lattice.The vector

$$\boldsymbol{v}_{ij;\text{neighb}(ij)}(t) = \left(y_{(i-1),(j-1)}(t), y_{(i-1),j}(t), y_{i,(j-1)}(t), y_{i,(j+1)}(t), y_{(i+1),j}(t), y_{(i+1),(j+1)}(t)\right) \tag{12}$$

is the vector of the neighbouring neurons outputs (with respect to the ij-th neuron). These neighbouring neurons influence the values of the weights $w_{ij,kl}(t)$ of the given ij-th neuron by the functional term $f_l(\boldsymbol{v}_{ij,\text{neighb}(ij)}(t))$. This is the *local adjustment* of the weights values which are adaptively provided for every moment of time t as a function of the current output values of the neighbouring neurons denoted by the vector $\boldsymbol{v}_{ij;\text{neighb}(ij)}(t)$. The term $f_l(\boldsymbol{v}_{ij,\text{neighb}(ij)}(t))$ represents a function which determines a local path in the alignment lattice. This function is described in the following section.

The second term $f_g(y_{kl}(t))$ in the equation determining the weights value denotes the influence of the kl-th neuron output on the weight $w_{ij,kl}$ of the given ij-th neuron. This is the so-called *global structural influence* on the weights values from every neuron (not necessarily local) in the alignment lattice providing an input to a given neuron.

The weight values are not constant, as in the Hopfield network, but vary depending on whether they represent (at the current time) a legal or illegal path in the alignment lattice. The current value of weight $w_{ij,kl}(t)$ at time t consists of two parts

$$w_{ij,kl}(t) = w^l_{ij,kl}(t) + w^g_{ij,kl}(t) \tag{13}$$

The first part

$$w^l_{ij,kl}(t) = f_l(\boldsymbol{v}_{ij,\text{neighb}(ij)}(t)) \tag{14}$$

denotes the contribution to the weight value from the neighbouring neurons outputs (governing the legal paths in the alignment lattice).
The other part

$$w^g_{ij,kl}(t) = f_g(y_{kl}(t)) \tag{15}$$

denotes the global structural influence of lattice neurons on the weight of a given neuron.

In terms of weights matrix we have

$$\boldsymbol{W}(t) = \boldsymbol{W}^l(t) + \boldsymbol{W}^g(t) \tag{16}$$

where $\boldsymbol{W}^l(t)$ denotes the matrix of the contribution to the weights values $w^l_{ij,kl}$ and $\boldsymbol{W}^g(t)$ denotes the matrix of the contribution to the weights values $w^g_{ij,kl}$.

5.1.1. Local Legal Path Equations and the Weight Matrix $\boldsymbol{W}$

The equations which govern the weight matrix $\boldsymbol{W}^l$ values, contributed by the local neurons, are computed dynamically and, for any node, are represented by a (possibly proper) subset of the following set of equations:

$$w^l_{ij;(i-1),j} = E_l + d_{i/d} + I_l \max(y_{(i-1),(j-1)}, y_{i,(j+1)}) \tag{17}$$

$$w^l_{ij;i,(j+1)} = E_l + d_{i/d} + I_l \max(y_{(i+1),(j+1)}, y_{(i-1),j}) \tag{18}$$

$$\begin{aligned} w^l_{ij;(i+1),(j+1)} &= E_l + d_s \text{mismatch}(char_i, char_j) \\ &\quad + I_l \max(y_{(i+1),j}, y_{i,(j+1)}) \end{aligned} \tag{19}$$

$$w^l_{ij;(i+1),j} = E_l + d_{i/d} + I_l \max(y_{i,(j-1)}, y_{(i+1),(j+1)}) \tag{20}$$

$$w^l_{ij;i,(j-1)} = E_l + d_{i/d} + I_l \max(y_{(i+1),j}, y_{(i-1),(j-1)}) \tag{21}$$

$$\begin{aligned} w^l_{ij;(i-1),(j-1)} &= E_l + d_s \text{mismatch}(char_{i-1}, char_{j-1}) \\ &\quad + I_l \max(y_{i,(j-1)}, y_{(i-1),j}) \end{aligned} \tag{22}$$

The quantities E_l ($E_l < |d_s|$), d_s, and $d_{i/d}$ are constants which reflect the positive excitation value between nodes, and the negative costs of substitution and deletion (or insertion), respectively.

The mismatch$(\cdot)$ is a binary function which returns 1 if two characters are not the same and 0 otherwise

$$\text{mismatch}(char_i, char_j) = \begin{cases} 0 & \text{if} \quad char_i = char_j \\ 1 & \text{if} \quad char_i \neq char_j \end{cases} \tag{23}$$

Note that if two characters match, the cost of matching the characters is zero.

The quantity I_l is a (negative) local inhibition constant which is multiplied by the maximum output voltage of the two competing nodes relative to the node and the weight.

5.1.2. Global Weight Interactions

In the alignment lattice, the position of a node in the array determines which nodes can be linked to it in a path from the upper left node to the lower right. For a node located in a position ij (in a row i and a column j) in the alignment lattice, all nodes with position kl can be linked to node ij in a path if $i \geq k$ and $j \geq l$, or

$i \leq k$ and $j \leq l$. Conversely, if $i \geq k$ and $j < l$, then the two nodes cannot be linked in a legal path in the alignment lattice.

The weight values between nodes which are not in close proximity to each other are determined by the preceding relationship of the existence or non-existence of a legal path between the nodes. If a legal path exists between two non-local nodes, the weight values between them (in both directions) are positive values. These positive values tend to allow a partial path to excite possible nodes and guarantee, after a sufficient length of time, a complete path across the lattice. The weights between these non-local nodes related by a legal path are currently set to a positive constant. In other words, if a legal path exists between the non-local nodes ij and kl, then

$$w^g_{ij,kl} = w^g_{kl,ij} = +C \tag{24}$$

Conversely, if two non-local nodes are oriented so that no legal path can link them, then the two nodes are competing, and the output voltages of the two nodes are mutually exclusive (i.e. they both cannot be in the final state). The weights between these types of nodes should be negative (inhibitory) in nature, forcing the nodes to compete against each other for path inclusion.

Initially, weight values between competing nodes were set to a negative constant. This produced a bias in the network, with nodes in the upper right or lower left of the lattice being inhibited by more nodes than corresponding nodes around the diagonal of the matrix. This caused an unfair bias toward paths formed around the diagonal. To alleviate this problem, a non-linear function which makes the weight value into a function of the output voltage is used. The formula for the computation of the weight value from the node ij to the competing node kl is as follows:

$$w^l_{ij,kl} = -\tan(y_{ij} \times 89.7(\text{deg})) \tag{25}$$

All neurons start at low input voltage levels (due to the low level outputs voltage in the initial network state). Thus, an inhibition is not strong at the start, but becomes stronger as the network evolves, with these neurons which are increasing faster inhibiting their competitors. The excitation of possible nodes, along with the strong inhibition of competing nodes, forces the network to produce one and only one path across the alignment lattice.

5.2. How the Input Sequences Are Used in the Network Processing?

The input strings or sequences ultimately determine the behaviour of the network. A binary function mismatch($char_i, char_j$), which compares sequence elements (i.e. characters), returns 0 if two characters $char_i$ and $char_j$ match and 1 otherwise. The value of this character comparison function is used to alter the weight values for the corresponding diagonal weights. Since every character of one sequence is being compared (in some location on the lattice) with every character of the other sequence, the diagonal weights from the node (ij) to the node $(i+1),(j+1)$ correspond to the weights affected by the comparison of the i-th character of sequence 1 with the j-th character of sequence 2. These diagonal weight values of the lattice correspond to the cost of substitution or matching of the respective characters. If the characters

match, a positive weight value is assigned between the nodes, whereas a less positive or even negative value is applied if the characters mismatch. These relationships are described by the equations

$$w^l_{ij;(i+1),(j+1)} = E_l + d_s \text{mismatch}(char_i, char_j) + I_l \max(y_{(i+1),j}, y_{i,(j+1)}) \tag{26}$$

$$w^l_{ij;(i-1),(j-1)} = E_l + d_s \text{mismatch}(char_{i-1}, char_{j-1}) + I_l \max(y_{i,(j-1)}, y_{(i-1),j}) \tag{27}$$

which express the influence of the neighbouring neurons on the weights values. Since the system evolves in time, the nodes corresponding to characters which match and are along a possible path have their output voltages converging to 1 faster than other nodes which do not match. This accelerated convergence, along with the constraints from other weight interactions described in the following sections, allows the system to evolve into paths which minimize the cost associated with the path across the alignment lattice, thus minimizing the dissimilarity between the aligned sequences.

6. Neural Network Simulation Algorithm

The real implementation of the proposed neural alignment technique must involve using VLSI technology, which will provide essential reduction of computation time. In the following section we will discuss computer simulation of the dynamic neural network under consideration.

6.1. Numerical Update Formula for the Nodes Dynamics

Since the dynamics of the neural network implementing the alignment matrix is modelled by a set of dynamic non-linear differential equations, a numerical integration technique is used to model the evolution of dynamic states of the network which should converge to a solution. The Euler integration technique was used to model the dynamic interaction between the neurons in the dynamic neural network. This practically means solving the set of non-linear differential equations representing the network dynamics by using Euler's integration method.

For a given neuron (node) ij, the equations for the intermediate output and output neuron values (or voltage levels) are given by the following formulae:

$$x_{ij}(t) = x_{ij}(t - \Delta t) + \Delta t\Big(\frac{-1}{\tau} x_{ij}(t - \Delta t) + \sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t - \Delta t) w_{ij;k,l}(t - \Delta t)\Big) \tag{28}$$

$$y_{ij}(t) = \frac{1}{2}\left(1 + \tanh\Big(\frac{x_{ij}(t)}{\beta}\Big)\right) \tag{29}$$

where for every time t the input, output, and weight values are recomputed. The time t is assumed to be less than one to allow a smoother transition to a stable state and to avoid oscillatory behaviour in the alignment lattice.

These equations, with the right-hand side values defined for various constants (i.e. inhibition between local nodes $d_{i/d}$, local substitution cost d_s, etc.), evolve toward a stable state where a single path is defined. The active nodes thereby compete and kill all competitors, while general excitation maintains a constant source of energy to the system so that output voltage of all the nodes will not go to zero.

6.2. Simulation Algorithm

The previous sections described the types of interactions between the nodes and the way in which the weights are dynamically assigned values in the alignment lattice. This section will give an overview of the steps involved in the algorithm.

The algorithm proceeds as follows:

1. The length and contents of the sequences are read from a file.

2. The network is initialized to the appropriate size based on the size of the input sequences. The initial values of the input voltages of the network are initialized with a randomized negative number (around -2). This negative input to the node produces a small output voltage from the node of approximately 0.006. The variance of the negative input voltages is kept very small (less than 0.002) so that the system is not pushed into a local minima by the random values.

3. A positive external bias is added to the input voltage of the top-left corner node to apply an external force to guarantee that the network's top-left node output voltage is driven to 1.0 as the system evolves to a solution.

4. The algorithm now iterates over the nodes and weights, allowing the interaction between the nodes to drive the network to a solution. During each iteration, the algorithm performs steps (a) through (d), which computes the next internal states $x_{ij}(t)$ $(i = 1, 2, ...k + 1,\ j = 1, 2, ..., l + 1)$ of the neurons at time t based on the previous internal state $\mathbf{x}(-\Delta t)$ at time $t - \Delta t$ (and of course the previous output signals $\mathbf{y}(t - \Delta t)$ of the neurons). This algorithm is an implementation of Euler's approximation of the solution to the set of non-linear differential equations being the model of the proposed neural network.

   (a) We assume that the previous internal state vector $\boldsymbol{x}(t - \Delta t) = (x_{11}(t - \Delta t), x_{12}(t - \Delta t), ..., x_{kl}(t - \Delta t))^T$ of the network, and the output voltage vector $\boldsymbol{y}(t) = (y_{11}(t), y_{12}(t), ..., y_{kl})^T$ of each neuron (node) for the previous time $t - \Delta t$ is computed. The initial state of the network $\boldsymbol{x}(t_0)$ at the initial time t_0 is assumed to be known.

   First, the total input voltage of the neuron is computed as a sum of weighted outputs of all the neurons (computed in the previous step $t - \Delta t$) creating feedback in the network (including feedback from the given neurons as well). The total input voltage is transformed by a sigmoid output activation producing the current output voltage of a given neuron.

   (b) For the current computation time t, the weight values are recomputed, using the current output voltages and a function comparing sequence element values. For local or neighbourhood weight computation, the input

sequences are compared for matching or mismatching, with excitation or positive weight values for a match and less positive (or negative) values for a mismatch. The neighbouring node output values also control the final result of the weight, to guarantee legal paths. Global weight values are excitatory or inhibitory in nature, and are designed to force the network to find a single complete path across the network. The inhibition values are computed as a non-linear bounded (exponential) function of the output voltage. The excitatory weight value, which denotes a possible path to or from a given node, is a positive constant value. This value helps drive possible nodes up in voltage to guarantee the generation of a complete path across the alignment lattice.

(c) The instantaneous total input voltage at the current time t of a neuron is computed from the summation of output voltages from all the neurons output voltages multiplied by the corresponding weight values from the nodes to the given node.

(d) The new internal state $\boldsymbol{x}(t)$ of the network for the current time t is computed using the Euler approximation. The Euler approximation technique is applied for each neuron in the random order. We assume that the previous network state $\boldsymbol{x}(t-\Delta t)$ and previous output $\boldsymbol{y}(t-\Delta t)$ have been "frozen" for the sampling period Δt. For each neuron, the total input voltage at time t is computed first, which is the sum of weighted outputs $y_{ij}(t-\Delta t)$ $(i=1,...,k,\ j=1,2,...,l)$ of all the neurons plus external bias input (if it exists)

$$\sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t-\Delta t)w_{ij;kl}(t-\Delta t) \tag{30}$$

Then, the "derivative value" at the previous time is computed as

$$\frac{-1}{\tau}x_{ij}(t-\Delta t)+\sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t-\Delta t)x_{ij}(t) = x_{ij}(t-\Delta t)$$
$$+\Delta t\Big(\frac{-1}{\tau}x_{ij}(t-\Delta t)+\sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t-\Delta t)w_{ij;k,l}(t-\Delta t)\Big) \tag{31}$$

To compute the next state of the neuron, according to the Euler scheme, the above computed approximation of derivative for the whole period Δt is multipled by the time constant Δt which gives the increment of the neuron state at time t with respect to the previous state $x_{ij}(t-\delta t)$. Finally, the next state of the neuron at time t is computed as

$$x_{ij}(t) = x_{ij}(t-\Delta t)+\Delta t\Big(\frac{-1}{\tau}x_{ij}(t-\Delta t)\Big)+\sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t-\Delta t)x_{ij}(t)$$
$$= x_{ij}(t-\Delta t)+\Delta t\Big(\frac{-1}{\tau}x_{ij}(t-\Delta t)+\sum_{k=1}^{m}\sum_{l=1}^{n} y_{k,l}(t-\Delta t)\Big)w_{ij;k,l}(t-\Delta t) \tag{32}$$

This computation is repetad for all the neurons in the network.

(e) The output values $y_{ij}(t)$ $(i = 1, 2, ..., k,\ j = 1, 2, ..., l)$ of the neurons at the current time t are computed from the equation for the output activation function

$$y_{ij}(t) = \frac{1}{2}\left(1 + \tanh\left(\frac{x_{ij}(t)}{\beta}\right)\right) \tag{33}$$

The algorithm iterates and the constraints force the network to a stable state. The state is a complete path through the alignment lattice which minimizes the cost function associated with the traversing of the lattice. This is equivalent to maximizing the alignment of the sequences.

6.3. Numerical Results

Using the values shown below in the equations defined previously produced a system which found optimal or near-optimal paths maximizing the alignment of two string sequences. The values used for both local and global weight calculations are as follows:

- Global weight constants:

P = 0.05	possible path excitation
I = −15.0	impossible path inhibition

- Local weight constants:

E_l= 20.0	local excitation between cooperating nodes
d_c=0.0	local cost associated with a continuation
$d_{i/d}$= −10.0	local cost associated with an insert (delete) path
d_s=−12.0	local cost associated with a substitution (mutation) path
I_l= −30.0	local inhibition between competing nodes

6.4. Numerical Experiments

Several computer numerical experiments have been conducted to test the proposed design of dynamic neural networks applied to the human genome alignment. The C language program has been developed to design and simulate dynamic adaptive neural networks. The experiments allowed us to select the proper design parameters of the networks, like P–possible path excitation, N–impossible path inhibition, local excitation E_l, local inhibition I_l, insert-delete cost $d_{i/d}$, substitution cost d_s, as well as the simulation timing conditions like the time interval T.

```
Experiment 1.

***Parameter Values***
      Possible path excitation (P) =      0.250000
      Local excitation (E) =             20.000000
      Local inhibition (I) =            -20.000000
      Insert-Delete Cost (D) =          -10.000000
```

```
Substitution Cost (S) =          -12.000000
Activation Lambda (l) =            0.700000
Node Initial Input Voltage (V) =  -2.000000
External Voltage Bias (b) =        1.000000
time interval (T) =                0.025000
random number seed (r) =           1
number iterations (n) =           60
output mod number (m) =           10

    a    c    d    c    d    a    c    c    f    a
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
f
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
f
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
f
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

    a    c    d    c    d    a    c    c    f    a
0.03 0.02 0.02 0.02 0.02 0.02 0.02 0.01 0.02 0.01 0.01
f
0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01
a
0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
d
0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01
c
0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01
d
0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
a
```

```
      0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
c
      0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
d
      0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
a
      0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
f
      0.01 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
f
      0.01 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
c
      0.01 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
c
      0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
a
      0.01 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
d
      0.01 0.01 0.01 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.03

         a    c    d    c    d    a    c    c    f    a
      0.72 0.33 0.17 0.09 0.05 0.03 0.01 0.01 0.01 0.00 0.00
f
      0.79 0.48 0.34 0.25 0.14 0.17 0.04 0.02 0.01 0.01 0.00
a
      0.29 0.70 0.74 0.20 0.36 0.11 0.14 0.04 0.02 0.01 0.00
d
      0.19 0.71 0.22 0.91 0.12 0.33 0.18 0.11 0.03 0.02 0.00
c
      0.13 0.23 0.85 0.15 0.91 0.13 0.14 0.18 0.10 0.03 0.01
d
      0.29 0.19 0.17 0.62 0.16 0.90 0.13 0.12 0.09 0.07 0.01
a
      0.06 0.65 0.17 0.56 0.21 0.20 0.81 0.27 0.11 0.07 0.04
c
      0.04 0.10 0.70 0.15 0.85 0.21 0.22 0.54 0.31 0.11 0.03
d
      0.05 0.08 0.11 0.48 0.16 0.87 0.23 0.26 0.21 0.27 0.06
a
      0.02 0.09 0.10 0.14 0.23 0.27 0.70 0.26 0.50 0.21 0.18
f
      0.01 0.05 0.09 0.13 0.19 0.30 0.31 0.33 0.57 0.48 0.12
f
      0.01 0.06 0.06 0.20 0.15 0.20 0.63 0.53 0.26 0.56 0.16
c
      0.00 0.03 0.07 0.13 0.23 0.16 0.41 0.87 0.47 0.30 0.21
c
      0.00 0.01 0.04 0.05 0.15 0.31 0.18 0.42 0.74 0.73 0.27
a
      0.00 0.01 0.02 0.03 0.09 0.10 0.34 0.25 0.35 0.48 0.78
d
      0.00 0.00 0.00 0.01 0.02 0.06 0.06 0.11 0.19 0.36 0.73

         a    c    d    c    d    a    c    c    f    a
      1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

```
f
     1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.92 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
d
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

        a    c    d    c    d    a    c    c    f    a
     1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
f
     1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
```

```
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
d
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

         a    c    d    c    d    a    c    c    f    a
     1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
f
     1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
d
     0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
f
     0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
c
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 0.00
a
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
d
     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00

Input Sequences:
      acdcdaccfa
      fadcdacdaffccad

Sequence Alignment Results:
      *acdcd***a**ccfa*
      fa*dcdacdaffcc*ad
```

7. Conclusions

As the results show, the genome sequence neural network has the capability of finding optimal and near-optimal alignments of strings. Experimentation has shown that the

genome network is very stable, but sub-optimal results can occur when the system of equations evolve to a solution too quickly. Research on the relationship between the size of the delta-time constant and the size of the sequences is continued.

Due to the parallel nature of the genome sequence neural network, this algorithm provides a distributed and asynchronous solution to the genome sequence alignment problem, and can find minimal cost paths in alignment lattices in a fraction of the time needed for sequential techniques. Once implemented in a VLSI curcuit, or ported to a parallel computer, the computational time required will not depend on the size of the sequences, but on the capability of the respective curcuit or computer.

Currently, the network does not optimize the path for multiple (without gap) matching, neither does it assign a penalty for non-adjacent matching of characters. This capability is a focus for future work.

Because of the immense storage requirements (for a 50×50 string comparison, a weight matrix requires 25MB of storage space), the proposed neural algorithm for genome sequences alignment would be practical only if embedded in a VLSI electronic device. On the other hand, the computation time of the alignment a in VLSI device will not depend on the size of processed sequences. Of course, for longer sequences the VLSI circuit will require more neurons implementing the alignment lattice. The number of neurons N_n required for implementing this alignment matrix, for a sequence $\boldsymbol{a}$ of length L_a and a sequence $\boldsymbol{b}$ of length L_b, is equal to

$$N_n = L_a \times L_b \tag{34}$$

The neural alignment method presented in the paper may be also implemented in the parallel computer, using the simulation technique described here.

References

Altschul S.F. and Erickson B.W. (1986): *Optimal sequence alignment using affine gap costs.* — Bull. Math. Biol., v.48, pp.603–616.

Altschul S.F. and Lipman D.J. (1989): *Trees, stars, and multiple biological sequence alignment.* — SIAM J. Appl. Math., v.49, pp.197–208.

Arbanel R., Wiencke P.R., Mansfield E., Jaffe D.A. and Brutlag D.L. (1984): *Rapid searches for computer patterns in biological molecules.* — Nucl. Acid. Res., v.12, pp.263–280.

Bacon D.J. and Anderson W.F. (1986): *Multiple sequence alignment.* — J. Molec. Biol., v.191, pp.153–161.

Bellman R.E. (1957): *Dynamic Programming.* — New Jersey, Princeton: Princeton University Press.

Bohr H., Bor J., Burnak S., Cotterill R.M.J., Lautrup B., Norskov L., Olsen O.H. and Peresen S.B. (1989): *Protein secondary structure and homology by neural networks: the alpha helices in rhodopsin.* — Private notes.

Carrilo H. and Lipman D. (1988): *The multiple sequence alignment problem in biology.* — SIAM J. Apl. Math., v.48, No.5, pp.1073–1081.

Coulson A.F.W., Collins J.F. and Lyall A. (1987): *Protein and nucleic acid sequence database searching: a suitable case for parallel processing.* — Comput. J., v.30, pp.420–424.

Delahaye J.-P. (1988): *Sequence Transformation.* — Berlin: Springer-Verlag,

Dayhoff M.O., Schwartz R.M. and Orcutt B.C. (1978): *A model of evolutionary change in proteins.* — In: Atlas of Protein Sequences and Structure, Washington: National Biomedical Research Foundation, v.5, Suppl.3, pp.345–352.

Felsenstein J. (1982): *Numerical methods for inferring evolutionary trees.* — The Quart. Rev. Biol., v.57, pp.379–404.

Fickett J.W. (1984): *Fast optimal alignment.* — Nucl. Acids Res., v.12, pp.175–180.

Fitch W.M. (1970): J. Mol. Bil., v.49, pp.1–14.

Fitch W.M. and Smith T.M. (1983): *Optimal sequence alignments.* — Proc. Nat. Acad. Sci., USA, v.80, pp.1382–1386.

Freedman M.L. (1984): *Algorithm for computing evolutionary similarities measures with length independent gap penalties.* — Bull. Math. Biol., v.46, pp.553.

Gnanadesikan R. (1977): *Methods of Statistical Data Analysis of Multivariate Observations.* — New York: John Wiley.

Goad W.B. and Keneisha M.I. (1982): *Pattern recognition in Nucleic Acid Sequences. I. A General Method for Finding Local Homologies and Symmetries.* — Nucl. Acids Res., v.10, pp.247–278.

Gotoh O. (1986): *Alignment of three biological sequences with an efficient traceback procedure.* — J. Theor. Biol., v.121, pp.327.

Von Heijne G. (1989): *Sequence Analysis in Molecular Biology.* — San Diego: Academic Press.

Hopcroft J.E., Ullman J.D. (1979): *Introduction to Automata Theory, Languages and Computation.* — Reading: Addison-Wesley.

Hopfield J.J. (1982): *Neural networks and physical systems with emergent collective computational abilities.* — Proc. Natl. Acad. Sci., USA, v.79, pp.2554–2558.

Hopfield J.J. (1984): *Neurons with grade response have collective computational properties like those of two-state neurons.* — Proc. Natl. Acad. Sci., USA, v.81, pp.3088–3092.

Hopfield J.J. and Tank D.W. (1985): *Neural computation of decisions in optimization problems.* — Biol. Cybern., v.52, pp.141–152.

Hopfield J.J. and Tank D.W. (1986): *Computing with neural circuits: a model.* — Science, v.233, pp.625–633.

Lesk A.M. (Ed.) (1988): *Computational Molecular Biology. Sources and Methods for Sequence Analysis.* — Oxford: Oxford University Press.

Lipman D.J. and Pearson W.R. (1985): *Rapid and sensitive protein similarity searches.* — Science, v.227, pp.1435–1441.

Miller W. and Myers E.W. (1988): *Sequence comparison with concave weighting function.* — Bull. Math. Biol., v.50, pp.97–103.

Minsky M. and Papert S. (1969): *Perceptrons.* — Cambridge: MIT Press.

Murata M., Richardson J.S. and Sussman J.L. (1985): *Simultaneous comparison of three protein sequence.* — Proc. Natl. Acad. Sci., USA, v.82, pp.7657.

Needleman S.B. and Wunsch C.D. (1970): *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* — J. Molec. Biol., v.48, pp.443–453.

Ohya M. (1989): *Analysis of molecular evolution by mutual entropy. An application of informational theory in genetics.* — Int. Rep. Dept. Information Sciences, Univ. of Tokyo, Japan.

Quian Ning, and Sejnowski T.J. (1987): *Predicting the secondary structure of globular proteins using neural network models.* — Private notes.

Queen C., Wegman M.N. and Sommer R. (1982): *Improvement to a program for DNA analysis: a procedure to find homologies among many sequences.* — Nucleic Acids Res., v.10, No.1, pp.449–456.

Sankoff D. (1975): *Minimal mutation trees of sequences.* — SIAM, J. Appl. Math., v.28, pp.35–42.

Sankoff D. and Kruskal D. (Eds.) (1983): *Time Warps, Strings Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* — Reading: Addison-Wesley Publ. Comp. Inc.

Sellers P.H. (1974): *An algorithm for the distance between two finite sequences.* — J. Combinatorial Theory, v.16A, pp.253–258.

Sellers P.H. (1979): *Pattern recognition in genetic sequences.* — Proc. Natl. Sci., USA, v.76, pp.3041–3049.

Sellers P.H. (1984): *Pattern recognition in genetic sequences by mismatch density.* — Bull. Math. Biol., v.46, No.4, pp.501–514.

Smith H., Waterman M.S. and Fitch W.M. (1981): *Comparative biosequence metrics.* — J. Mol. Biol., v.18, pp.38–46.

Sobel E. and Martinez J.T. (1986): *A multiple sequence alignment program.* — Nucl. Acids Res., v.14, pp.487–496.

Stormo G.D., Schneider T.D., Gold L. and Ehrenfeucht A. (1982): *Use of the 'Perceptron' algorithm to distinguish translation internal sites in E. coli.* — Nucl. Acids Res., v.10, pp.2997–3011.

Świniarski R. (1991): *Introduction to Neural Networks*, In: Handbook of Cybernetics and Systems (C. V. Negoita, Ed.). — New York: Marcel Dekker.

Tank D.W. and Hopfield J.J. (1986): *Simple 'neural' optimization networks: an A/D converter, signal decision circuits, and linear programming circuits.* — IEEE Trans. Circuits Systems, v.CAS-33, pp.533–241.

Waterman M.S. (Ed.) (1989): *Mathematical Methods for DNA Sequences.* — Boca Raton, Florida: CRC Press, Inc.

Waterman M.S., Smith T.F. and Beyer W.A. (1976): *Some biological sequence metrics.* — Adv. Math., v.20, pp.367–387.

Weir B.S. (Ed.) (1983): *Statistical Analysis of DNA Sequence Data.* — New York: Marcel Dekker, Inc.

Wimp J. (1981): *Sequence Transformations and Their Applications.* — New York: Academic Press.

Zucker M. and Somorjai R.L. (1989): *The alignment of protein structures in three dimensions.* — Bulletin of Mathematical Biology, v.51, No.1., pp.55–78.