Complex Systems **9** (1995) 305–327

A Temporal Sequence Processor Based on the Biological Reaction-diffusion Process

Sylvian R. Ray
Hillol Kargupta
Department of Computer Science,
University of Illinois at Urbana-Champaign,
Urbana, IL 61801, USA

Abstract. Temporal sequences are a fundamental form of information and communication in both natural and engineered systems. The biological control process that directs the generation of iterative structures from undifferentiated tissue is a type of temporal sequential process. A quantitative explanation of this temporal process is *reaction-diffusion*, initially proposed in [1] and later widely studied and elaborated.

We have adapted the reaction-diffusion mechanism to create a temporal sequence processor (TSP) composed of cells in a diffusion-supporting medium that performs storage, associative retrieval, and prediction for temporal sequences. The TSP has several interesting attributes: (1) Its achievable depth (or degree) is constrained only by storage capacity, (2) it tolerates substantial time warp, and (3) it supports user-specified flexible groupings of stored sequences into coarser classes at retrieval time. The TSP is also capable of preserving the time extent of stored symbols, as in a musical melody, and permits retrieval of both the symbols and their temporal extent. Experimental verification of the properties of the TSP was performed with Reber grammar sentences and musical melodies.

1. Introduction: Temporal sequences

Temporal sequences arise naturally when one attempts to process any time domain signal for the purpose of recognizing or predicting the presence of features relevant to the particular application. Human speech, biomedical signals, music, or any function of time originating from a sensor constitutes a temporal sequence whose meaning or intelligence content depends not only on the existence of certain features, but also on their particular temporal order. A time domain signal is most conveniently abstracted as a temporal sequence of feature vectors, often called a *spatiotemporal sequence*,

$$X = \{\mathbf{x}(t_i)\}, \qquad t_i = 1, 2, \ldots, \infty.$$

where each vector $\mathbf{x}(t_i)$ is a feature vector, a condensed but adequately veridical representation of the signal in the vicinity of time t_i.

The feature vectors X originating directly from signal applications usually span a space of high dimensionality, for example, R^{15}. In many practical applications, the infinite number of possible feature vectors in a real space are reduced to a finite number of classes by a clustering operation such as a self-organizing feature map or k-means clustering. Upon representing each of K clusters or equivalence classes with a symbol $s_i, i = 1, \ldots, K$, we obtain a manageable set of K symbols forming the assumed input for the temporal sequence processor. Abstracted spatiotemporal sequences are therefore represented as strings such as *cjkltprmmbos* without significant loss of representational power compared with the actual signal information. The basic sequences can be supplemented with time extent information by minor extension of the fundamental representation, as we discuss in section 4.7.

Also, the actual cardinality K of the finite alphabet, if too large, can become a major consideration, but this is a question of scale that we will ignore.

2. The objectives of temporal sequence processing

There are two distinct sequence processing tasks that we are particularly interested in pursuing here. These tasks arise in the context of semantic content discovery in signals [2].

1. *Embedded sequence recognition (ESR).* A number of short pattern sequences, $(ps_1, ps_2, \ldots) = \mathrm{PS}$, are stored in the device as shown in Figure 1. An unbounded argument sequence, ARGSEQ, is compared with the set, PS, contained in the sequence memory. When a subsequence of ARGSEQ is found that matches any member of the set PS within a prespecified accuracy, a successful recognition is indicated. As an example of this case in speech recognition, PS is some transformed representation of phonemes or quasiphonemes, and the ARGSEQ is a similarly preprocessed speech signal.

2. *Sequentially addressed sequence memory (SASM).* "Long" sequences, ST, are stored (see Figure 2). Short "address" sequences are compared with ST for the purpose of locating regions of ST that match the applied address sequence. If a sufficiently close match is found, we want to recognize the condition and possibly read out the continuation of ST from the endpoint of the successful match or, possibly, find the beginning of ST and read the whole stored sequence. This case is the traditional CAM problem, modified by the requirement of *sequential* presentation of the address and *sequential* readout from the match point.

Two distinct algorithmic procedures (ESR and SASM) are needed to solve these two problems.

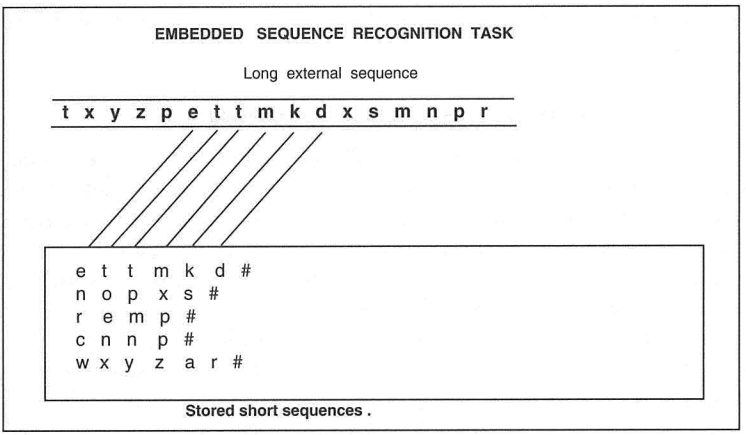

Figure 1: Embedded sequence retrieval is the identification of "short" stored subsequences within a long or unbounded argument sequence. It is performed by the guided sequence retrieval algorithm.

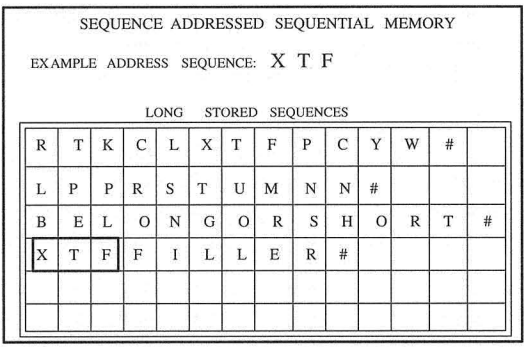

Figure 2: The task defined as sequence-addressed sequential memory is defined as the location of short sequences within long stored strings, followed possibly by retrieval of the identified sequence.

- For ESR and for the addressing phase of SASM, an algorithm is used to guide the comparison of an external address sequence with internal sequences. We will call this algorithm *guided sequence retrieval* (GSR).
- For the readout phase of SASM, after a unique stored sequence has been identified, the algorithm used is *free sequence retrieval* (FSR). It depends solely on internal stored states. FSR is also viewed as time-series prediction.

Thus, our main objectives from an application viewpoint (ESR and SASM) will be seen to reduce to the two algorithms, GSR and FSR, appropriately applied. If the effectiveness of GSR and FSR can be demonstrated convinc-

ingly, the success of the applications follows. The GSR and FSR algorithms will be covered later.

2.1 Desired qualities of a temporal sequence processor

Let us briefly review the qualities we desire for a TSP from an engineering viewpoint. These qualities will affect how well and how completely the TSP can achieve GSR and FSR over a range of problems, that is, its versatility.

First, some definitions are needed. A *complex sequence* is one constructed from any number and order of symbols from a permissible alphabet. We assume only complex sequences in this paper.

The *depth* or *degree* of a sequence is the minimum number of symbols preceding s_n required to predict s_n accurately. By a *system having depth n*, we mean that any symbol s_k requiring no more than the preceding n symbols, $s_{k-n}, \ldots, s_{k-2}, s_{k-1}$, to predict it can be predicted with certainty. For example, if the system has depth ≥ 8, then having learned the two sequences *xabcbaahz* and *axbcbaahy*, the terminal z and y are infallibly predicted.

- *Quality 1: Depth.* The TSP should be capable of minimum depth $n \gg 1$. n should be a design parameter. Note that the depth n, for complex sequences, implies the ability to count up to n repetitions of the same symbol, which is a strong requirement.

- *Quality 2: Temporal flexibility.* The TSP should tolerate temporal scaling and warp during storage, addressing, and retrieval.[1] *Scaling* refers to the absolute time units; *warp* is the nonuniformity of intersymbol spacing. However, it should be possible to retrieve the relative presentation length of the symbols if that information is stored.

- *Quality 3: Equivalence class flexibility.* The TSP should be capable of flexibly adjusting the "radius" of the equivalence classes of stored sequences. Thus, similar sequences, learned as distinct, may be treated as equivalent without altering the stored data. This quality implicitly permits limited errors such as missing or added symbols to be either tolerated or not tolerated, depending on the objective during retrieval.

- *Quality 4: Minimum storage.* We define the absolute storage requirement as the total number of weights needed to store a particular collection of sequences. The total number of weights required depends on the number of common subsequences in the data. In the present context, the required storage capacity is $N_{\text{sym}} * N_t$, where N_{sym} is the number of symbols in the alphabet and N_t is the number of symbol transitions that must be distinguished. Obviously, we want to minimize this requirement.

- *Quality 5: Sequential content addressability.* The TSP should provide for content addressability with a sequentially supplied address.

[1] The terms *retrieval* and *recognition* are used synonymously here.

2.2 Implications of desired qualities

What exactly are the properties that the desired qualities and our applications objectives imply about the TSP or network in general terms? A careful analysis of this question yields instructive insights on the subject of the TSP.

First, in order to simplify the addressing algorithm, it is very desirable to have only one cell (or neuron) per symbol of the alphabet. This choice defeats the potential problem of having to manage multiple active paths while searching for a unique responsive sequence.

Second, to be able to distinguish sequences such as *cbbbh* from *cbbbbg*, *each transition must produce a distinct internal state*, for example, the transition from the second to the third b must result in a different internal state from that produced by the transition from the third to the fourth b. This implies that the one cell identified with the symbol b must be able to resolve the difference, however it may be represented, between ν and $\nu + 1$ previous occurrences of the symbol up to some maximum depth, n. (See [3] for a thorough discussion.) But, third, to satisfy temporal flexibility (quality 2), the ability of a cell to distinguish between the νth and $(\nu + 1)$th previous occurrences of a symbol must not be rigidly dependent on the absolute or relative time scale.

Finally, the desire for equivalence class resolution (quality 3) can be satisfied by an acceptance window diameter that is dynamically adjustable *at retrieval time* so that the transition from the νth to the $(\nu + 1)$th previous occurrence of a symbol can be either resolved or ignored according to a global control parameter.

Given these crystallized requirements for the TSP, the merits of various proposed systems can be judged for breadth of capabilities.

We will show that the TSP proposed here does satisfy all of these requirements with some limitations on time flexibility (quality 2) and on addressing of internal subsequences (quality 5).

2.3 Previous work on sequence recognition by networks

For an excellent summary article on temporal sequence processing see [4] The conclusion regarding simple recurrent network architectures (SRNs) [5, 6] is that they are extremely limited in depth, to the degree that they are impractical for many applications. [7] concludes that there is no straightforward way for these networks to exhibit the property of counting. Our own experiments support this conclusion. The limitation on SRNs is centered on the fact that there is only one vector to represent all past contexts, which severely limits depth. In addition, there is no known design formula that relates achievable depth to architecture parameters.

The recurrent cascade correlation network (RCC) in [8] is another interesting approach to TSP that is actively constructive, as is its parent, the cascade correlation network. It learns a Reber grammar efficiently and processes the test set with a very low error rate. A major limitation of RCC is

that its time delays are fixed, which limits its usefulness with time-variable inputs (see quality 2 of section 2.1).

The ability to train for depth and for variable intersymbol delay is afforded by the "gamma delay" arrangement explored in detail in [9]. It is difficult to compare their approach with the present work since the gamma delay assumes numerical signals, for which algebraic operations are defined, whereas we are treating the sequences as *nominal* symbols.

[10] also presents an interesting mechanism for retaining and distributing the memory of past events in their TEMPO 2 model. They make use of an adaptive gaussian kernel to capture history by distributing the "trace" of a symbol in time. The reaction-diffusion method of history retention used here is essentially a superset of the diffusion idea, since the reaction process adds to the flexibility which can be exploited.

The temporal sequence processing algorithm studied in [11] addresses many of the same goals as the current effort. They achieve the depth objective by assuming an element with multiple terminals. Terminal 1 stores the state of the most recent input to the element, terminal 2 the second most recent external input to the element, and so on. Thus, by providing an n-terminal element representing symbol s_k, with each terminal corresponding to a distinct history of the sequence prior to s_k, a depth of n can be accurately stored. This design meets all of the desired qualities specified in section 2.1 except, possibly, for some limitations on quality 3. Minor properties of the design that are less than ideal are the fact that a maximum depth must be specified in advance and the biological basis for multiple distinct terminals is not supported. But the design in [11] squarely faces the need for sufficient storage to represent the complexity required, which is the primary problem with SRNs and most other attempts. It overcomes or ameliorates its need for prespecified maximum depth by providing a mechanism for *chunking*, that is, grouping subsequences iteratively into higher-level representations. We will return to a comparative discussion of the present reaction-diffusion method with the approach of [11] in section 5.

3. The biological basis: Reaction-diffusion

A well-studied biological experiment consists of the surgical removal of an internal segment of a cockroach tibia followed by regrafting of the distal and proximal parts, as illustrated in Figure 3 [12]. If the original tibia consisted of a sequence of similar but not identical segments numbered 123456789, and the segments 4567 are removed, it is observed that after one or two moults the internal segment is regenerated in its original order. This experiment implies the existence of memory of the tibia segment sequence and a controlled growth process. How is this to be explained? A quantitative explanation for this as well as myriad other pattern growth processes (e.g., zebra stripes, leaf capillary patterns) was set forth some 40 years ago [1] in the form of a set of partial differential equations that describe the self-stabilized increase and decay of growth-stimulating "morphogens" in a *reaction-diffusion* process.

Here is an example system.

Reaction-Diffusion Equations

$$\frac{\delta g_p(z)}{\delta t} = \frac{\epsilon g_p(z)}{r} - \alpha g_p(z) + D_g \left(\frac{\delta^2 g_p(z)}{\delta z^2} \right) \tag{1}$$

$$\frac{\delta r(z)}{\delta t} = \gamma \sum_p g_p^2(z) - \beta r(z) + D_r(\frac{\delta^2 r(z)}{\delta z^2}) \tag{2}$$

where g_p represents the concentration of the pth morphogen that excites the growth of the pth segment of the tibia, r is the concentration of the common or global reactant, and the coefficients are constants, the Ds being diffusion coefficients.

To give a sketch of the biological meaning, assume that segment 8 emits morphogen g_7. If $r = 1$, $\delta g_7/\delta t$ will grow by positive feedback (term 1 of equation 1) for some time, and the diffusion term (containing D_g) will diffuse the chemical into the segment 8–segment 3 interface, stimulating growth of segment 7 material. At distant locations, the reactant r will diffuse and suppress growth of segment 7. Segment 7 material at the interface with segment 3 triggers the emission of g_6, which causes segment 6 growth. Notice that the reactant concentration is generated in proportion to the sum of squares of the morphogens present, thus suppressing and stabilizing the process through the appearance of r in the denominator of the first term in equation 1. Thus the reaction-diffusion equations permit calculation of the growth, diffusion, and decay of each morphogen in turn, which effectively carries the history of previous activity forward in time while distributing the information spatially by diffusion. This is the natural biological process that we will simulate as the basis for the intercommunication of cells (elements) and the distribution of sequence information.

Figure 3 illustrates the experiment [13, 14]. A cockroach tibia is originally constructed as a sequence of similar but not identical segments numbered 123456789 in proximodistal order. If segments 4567 are surgically removed and the remaining portions of the tibia grafted together to form the sequence 12389, after one or two moults (M) the missing elements are found to regenerate. Further experimental evidence shows that growth morphogens are emitted sequentially starting from the distal segment, 8, inducing growth in the order 7654.

The existence of this biological sequence reproduction, being a form of sequential memory, attracted our interest in using an analogous process in the storage and retrieval of symbol sequences. The reaction-diffusion temporal sequence processor (ReDi TSP) emerged from this observation. The primary benefit of this starting point is a well-defined, stable system for communicating conditions from one cell by a means other than discrete conductive paths (e.g., axons or wires).

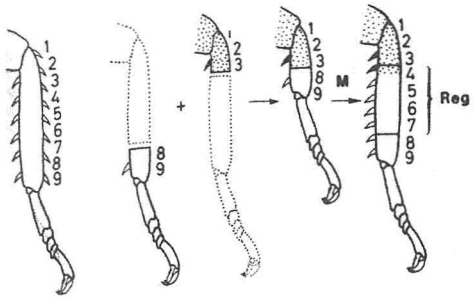

Figure 3: An experiment that demonstrates the regeneration of sequential segments of cockroach tibia. Correctly ordered regeneration of segments is explained by the reaction-diffusion model. (From [12].)

3.1 The computational model

Architecturally, the ReDi TSP model consists of cells randomly distributed in two- or three-dimensional space (Figure 4). We will assume a two-dimensional geometry, without loss of generality, for presentation simplicity. Cells (Figure 4, upper left) are fixed in position. Each of the N cells is uniquely labeled G_i, $i = 1, \ldots, N$, which is also the label of the output that the cell emits when activated.

The cells are interconnected through a diffusion-supporting medium. At any time t, and at any position (x, y), the internal "signal" is a vector

$$\mathbf{G}(x, y, t) = (G_1, G_2, \ldots, G_k, \ldots, G_K)$$

where K is the total number of distinct symbols,[2] $\mathbf{G}$ is the symbol concentration vector at location (x, y), and G_k is the concentration of the kth symbol at (x, y).

Each cell has a number of weight registers V_i, $i = 1, \ldots, P$. The length of each V register is K real numbers.

A cell can be activated by two means. First, the cell may act as a radial basis function cell in response to the local value of the $\mathbf{G}$ vector. If

$$AC = \min_{j=1}^{P} \|V_j - G(x, y)\| < \sigma_s$$

the cell is *internally* excited by level AC. The excited cells form a competitive network resulting in a single winner, namely, the cell j^*, which satisfies $\min_{j=1}^{N}(AC_j)$, being the most strongly excited cell.

Second, a cell C_M may also be excited by the external input directly, by having

$$\mathbf{Z}_M = \mathbf{S}_i$$

[2]Analogous to morphogens in the biological tissue growth model.

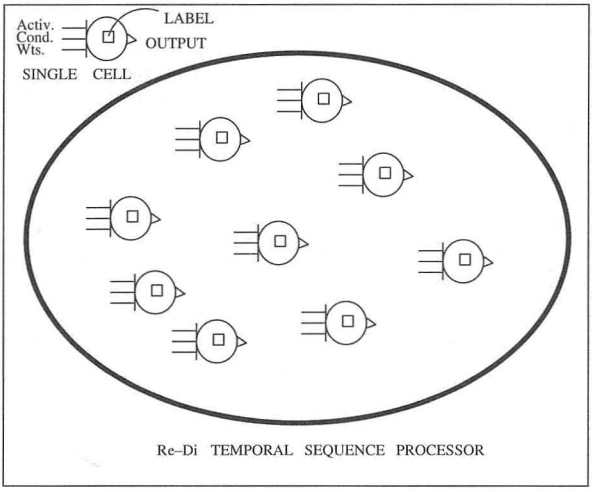

Figure 4: Architecture of the ReDi TSP. Identical cells are distributed randomly in a three-dimensional (or two-dimensional) volume supporting diffusion.

where S_i is the input symbol and $\mathbf{Z}_M$ is the external input weight vector of cell M. In this case we are assuming nominal (as opposed to linear) input symbols so that the input symbols are an orthogonal set.

The single winning cell, C_{j*}, emits a unit impulse of its label symbol, G_{j*}, after which the symbols react and diffuse according to the reaction-diffusion equations.

Let us briefly touch on the principal aspects of operation of the TSP. In general terms, the external input path with weights Z_M provides the means for learning the sequences originally and for guiding the sequence of activations in guided sequence retrieval operation.

The weights, $V_{M,j}$, specify the P different internal conditions when cell C_M corresponds to the next sequential input symbol. The V registers, after training, contain the memory of all learned transitions, including context, back to a depth determined by σ_s. Setting $\sigma_s = 0$ produces theoretically infinite depth or eidetic memory, limited only by P, the number of V registers, since every unique context of a symbol is learned. As σ_s grows, similar sequence contexts are treated as equivalent, which reduces depth but also reduces the storage capacity required. The states stored in the V registers are the sole source of sequential operation during free sequence retrieval. The fuller discussion of how the desired qualities of the TSP are achieved will be presented after the complete algorithms are described.

4. Sequence storage and recognition

4.1 Training the ReDi TSP to store sequences

A training or storage sequence $s_1 s_2 s_3 \ldots$ is presented at integral time points.[3] The input symbols are maximally sparse and therefore form an orthogonal set.

The number of cells is $N \geq K$, where K is the number of distinct symbols in the input symbol alphabet. P is the number of V registers per cell. The V registers are the weight vectors that can respond to internal symbol vectors in the diffusion-supporting medium.

Initially all V registers are in an unassigned state. In order to simplify the algorithm statement, without loss of generality, assume cells are uniquely assigned to each symbol of the alphabet and labeled C_{s_i}. The cell assignment is accomplished by setting the external weight vector, $\mathbf{Z}_i = s_i$.

The Storage (Training) Algorithm

1. Set $i = 1$.
2. Apply s_i. The winning (and active) cell will be C_{s_i}, abbreviated AC (active cell).
3. For the AC,
   If
   $$\min_{i=1}^{P} \|\mathbf{V}_i - \mathbf{G}(x, y)\| < \sigma_s$$
   the current transition (from s_{i-1} to s_i) is already known.

   Else recruit an unused V register, V_j, in cell C_{s_i} and set (one-shot learn)
   $$\mathbf{V}_j = \mathbf{G}(x, y).$$
4. Emit a unit impulse of symbol G_{s_i}.
5. Evaluate the reaction-diffusion equations for one symbol step, growing and diffusing all extant symbol concentrations. (A constant number of integration steps are used per symbol step depending on the dimensions of the cell array.)
6. Increment i and go to step 1 (until input sequence ends).

During storage, each symbol in the applied sequence causes the emission of a "symbol fluid," which diffuses spatially. The successive symbols build a vector $\mathbf{G}$, which varies with both time and space. History of the sequence

[3]The time points can be set by the appearance of input symbols or treated as absolute time units.

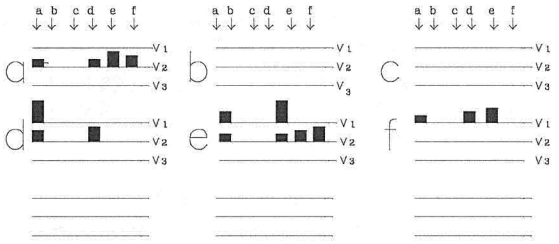

Figure 5: How the TSP works. Nine cells with three V registers are shown. Bar graphs represent the symbol concentration after the sequence *addefea* is learned. An activated cell traps the current symbol concentration in one of its V registers. For example, the first d traps only some G_a in its V_1 register. Each register represents one vector that can activate (fire) the cell.

is distributed throughout the volume, where it can be associated to the successor symbols. Later, during testing to detect the existence of a stored sequence, the sequence of emission and reaction-diffusion[4] that occurred during training will be recapitulated, exactly or approximately (depending on σ_s), by guided sequence retrieval.

4.2 An example of the training algorithm in use

An example of the stored V vectors that would exist after a sequence is stored is shown in Figure 5. The sequence *addefea...*, is stored.

When a first occurs, the environment is blank, so a activates Cell "a" (through the external Z inputs, not shown), which records 0s, the current local symbol concentration, in its first V register.

When the first d occurs, the symbol a has diffused to the vicinity of cell "d," which becomes activated from the external inputs through its Z weight vector. Being the AC, it stores the local symbol vector, $\mathbf{G}$, in its V_1 register, consisting only of some a symbol concentration.

When the second d occurs, cell "d" is reactivated and again stores the local $\mathbf{G}$, which contains some a and some d symbol concentrations. Cell "d" then emits a unit impulse of d symbol.

Upon reaching the second occurrence of a in the input sequence, the current symbol concentration at the x, y position of cell "a" is stored in the second V register of the cell, which then retains the state. The second V register of cell "a" shows the successively smaller values for e, f, d, and a, which are approximately proportional (in this case) to the recency of occurrence of the symbols in the input sequence.

[4]Note that symbol concentrations may grow as well as decay, depending on values of the free parameters in equations (1) and (2). This allows more degrees of freedom than diffusion alone.

The actual magnitudes of the symbols in the symbol vector, $\mathbf{G}(x, y)$, depend on the relative position of cells in the volume as well as on the constants used in the reaction-diffusion equations. The magnitudes of symbol concentrations do not even have to decay monotonically; it is essential only that the same subsequence produce the same concentrations on repeated appearance.

Storage with $\sigma_s = 0$ is analogous to eidetic memory and requires a different V register for every unique symbol transition. This condition is very profligate with memory usage. For increasing σ_s, however, an increasing number of transitions are treated as equivalent by using an existing V vector that is close enough to a subsequence already seen. In this case, a new V vector is not required, and this reduces storage space.

We have chosen to use a one-shot training algorithm here. Incremental training, typically used in neural networks, could be used to average out variations in timing of the symbols, but we will not pursue that avenue in the current investigation.

4.3 Demonstration of depth and counting quality

The depth and counting ability of the TSP, referred to in section 2.1, was demonstrated in a simple experiment simulating only the storage algorithm. This property was demonstrated by monitoring the growth of V register usage while storing a sequence with repetitions of a single symbol. The experiment consisted of storing the sequence *ddddddd*... and noting the symbol number, n, where the storage algorithm first did not invoke a new V register to store the state. This would correspond to a depth of $n - 1$ as well as the ability to predict the different successor symbol for the sequences $cd^{n-1}e$ and $cd^n f$, for example.

This experiment was performed with various values of tolerance, σ_s. The results are tabulated below. Unless otherwise noted, experiments were run with $D_g = 0.3$, $D_r = 0.3$, $\alpha = \beta = 0.3$, $\epsilon = 0.2$, and $\gamma = 0.1$ in equations 1 and 2.

Counting Test

σ_s	n (largest depth)
0.05	3
0.04	4
0.02	6
0.01	10
0	∞

When the recognition (or retrieval) algorithm, discussed later, uses the same value of σ, the storage process is recapitulated deterministically, so we can conclude that the depth during retrieval will be the same as that found by this experiment.

Metaphorically, larger values of σ_s correspond to paying less attention to the precise count. As the needed count or depth increases, the penalty is an increase in the required storage capacity, the total number of occupied V registers in the TSP.

4.4 The network in the embedded sequence recognition mode

Suppose many short sequences are stored in a ReDi TSP (as in Figure 1). Let a long argument sequence be presented beginning at its kth symbol, s_k, for the purpose of detecting whether the argument subsequence is stored in the network. This is guided sequence retrieval; we are using the known external sequence to guide the search for its stored equivalent.

It is assumed that (1) we want to search for the entire stored (short) string and (2) the period of symbol presentation is the same as during the storage operation. σ_T is the tolerance measure during test operations, setting the radius of acceptance of matches between the stored and argument sequences.

The algorithm is as follows.

Guided Sequence Retrieval

1. Set the argument string pointer, $k = 1$. Clear all symbol concentrations, $\mathbf{G}(x, y)$, to zero.
2. Using the external input s_k, activate cell C_{s_k} (whose label is G_{s_k}). If none, terminate with failure.
   If there exists a V register of C_{s_k} such that
   $$\sum_i \mid (G_i(x, y) - V_i) \mid \leq \sigma_T$$
   then C_{s_k} becomes the new AC and the transition $s_{k-1} \rightarrow s_k$ in the argument sequence is a *known* (previously encountered) transition. Emit one unit of s_k at the location of C_{s_k}, and perform one symbol step of the reaction-diffusion equations. Else, if no cell satisfies inequality 2, then the external sequence is known only from s_1 to s_{k-1}. Terminate the test with unknown sequence.
3. If a *known* transition occurred, increment k and return to step 2, iterating until an end mark of the stored sequence is encountered, implying a known sequence.

The foregoing algorithm of the test phase recapitulates the training steps in the sense that the symbol conditions present at storage are revivified during the *successful* matching of the argument sequence to an originally stored sequence. Notice, however, that we are permitting a different activation tolerance measure, σ_T, than we used during training. As σ_T rises from 0, we are requiring increasingly less strict matching of the symbol concentrations with the original conditions during storage of the sequence. The interesting feature here is that the precision of matching the external sequence to the internal sequence is controllable during recognition.

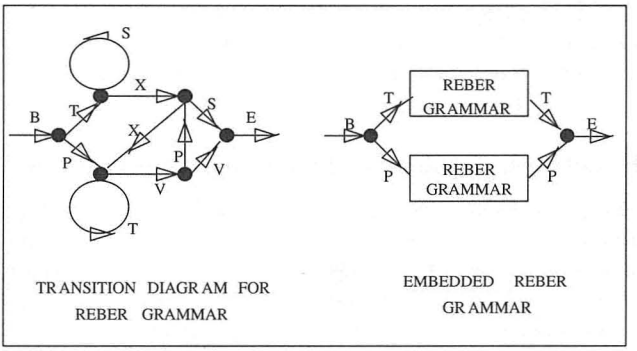

Figure 6: Embedded reber grammar. Only the penultimate symbol is predictable, given the second symbol.

4.5 An embedded sequence recognition experiment

Embedded sequence recognition provides a testbed to demonstrate how equivalence class flexibility and storage efficiency are traded off.

A finite-state Reber grammar has been strengthened as a test for depth and time-series prediction in recurrent neural nets. The generated sequences all have the second and penultimate symbols in deterministic relationship, generating sequences such as *BTXPSSTTTXTE* or *BPVTPSPE* (Figure 6) [8]. All interior sequence positions, other than the penultimate, correspond to transitions having equal probability of either of two symbols.

The average sentence length generated is about 10 symbols, with the maximum of about 30 symbols in a sample of 500 sentences.

A ReDi TSP with 25 cells in a 5×5 grid was defined. Five hundred strings were generated and applied tρ the TSP in the training or storage mode. Since the grammar uses only 7 symbols, there were 18 unused cells.

Varying the storage tolerance measure, σ_s, demonstrates the range of generalization possible during storage. In the limit as $\sigma_s \to 0$, every transition with a unique history is stored as a unique transition, consuming one V register. The first appearance of a distinct transition is signaled as an *unknown* transition, which requires recruitment of a V register. As σ_s increases, some transitions are treated as equivalent by the system. For example, when $\sigma = 0.05$, *bpptttvvpe* is stored as equivalent to *bpptvvpe*. Higher values of σ_s result in generalization or enlargement of equivalence class, eventually to the point that every transition is context-free.

Similarly, increased values of activation tolerance during retrieval, σ_T, permit more tolerance in the affirmative decision that a particular transition with its (possibly) extensive history is permissible.

The accuracy as a function of σ_T is plotted in Figure 7 for σ_s fixed at 0.0001. Cases where $\sigma_s > \sigma_T$ have relatively little meaning.

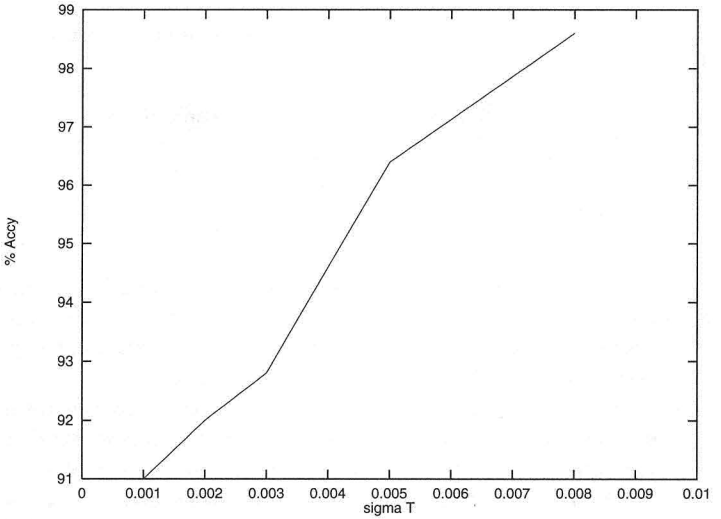

Figure 7: Accuracy of predicting embedded Reber grammar sentences as a function of σ_T, the test phase tolerance ($\sigma_s = 0.0001$).

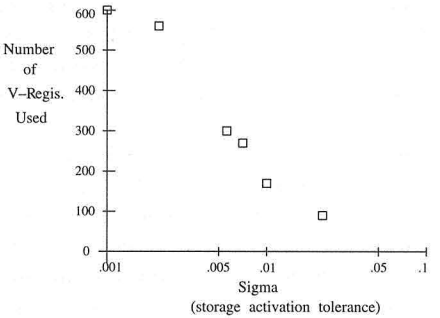

Figure 8: Number of activation condition registers (V registers) used as a function of σ_s. Symbol transitions totaled 4492.

To retrieve a sequence, a randomly generated Reber sentence is presented symbol-sequentially, and matching is attempted by the guided sequence retrieval algorithm. The retrieval is counted as accurate only if the penultimate symbol transition in the sentence is found to be a *known* transition.[5] The number of activation condition registers (V registers) required as a function of σ_s is displayed in Figure 8. This number is a function only of storage tolerance, σ_s, not retrieval tolerance.

[5]The *a priori* probability is 0.50.

Table 1: False positive rate (percent) for recognition of embedded Reber grammar sentences.

σ_s	$\sigma_T = 0.0001$	$\sigma_T = 0.001$	$\sigma_T = 0.005$
0.0001	16.4	25.8	75.6
0.001	15.8	25.8	75.6
0.005	8.0	7.0	71.0

Under the condition that practically every distinct transition in the training set is known ($\sigma_s = 0.0001$), the retrieval accuracy for 500 newly generated strings was found to vary from 91 percent with $\sigma_T = 0.001$ to 98.6 percent with $\sigma_T = 0.008$, a relatively relaxed matching requirement.

The storage efficiency is suggested by the fact that for the total 4501 symbol transitions in the 500 stored sequences, the required number of V registers ranged from 872 with $\sigma_s = 0.0001$ to 476 with $\sigma_s = 0.005$.

To study the ability of the TSP to reject false sentences, we trained the TSP as before, storing embedded Reber sentences containing the T-T and P-P deterministic correlation between the second and penultimate symbols. For the GSR or recognition stage, however, the embedded Reber grammar source was reversed *in the penultimate symbol only*, guaranteeing that *none* of the test sequences had been seen completely during training but that all sequences were *grammatically* identical up to the penultimate symbol.

Upon performing the experiment with this modification, all recognized sentences (false *knowns*) corresponded to false positives as indicated in Table 1.

When σ_T is small, the narrow acceptance window during retrieval easily blocks acceptance of most nonstored cases, lowering the false-positive rate to the 8 to 16 percent range. With a wider acceptance window, however, the false-positive rate soars. These results suggest a good compromise in the vicinity of $\sigma_s = \sigma_T = 0.001$, although this choice would interact with the free constants used in the reaction-diffusion equations.

4.6 Demonstration of tolerance to warp

In the preceding tests, one step per symbol was used during both storage and testing. This condition permits the symbol concentrations appearing during GSR steps to reproduce exactly the same sequence of symbol concentrations, $\mathbf{G(t)}$, as the network experienced during storage, assuming the tolerances σ_s and σ_T are equal. The desirability of tolerating intersymbol time variations or warp (noted as quality 2 in section 2) is obvious, particularly if the original source of the symbols is a biological one with its typical random property.

Consider the effects of warp in the ReDi TSP system. Let the spacing between symbol s_n and s_{n+1} be Δt during training. Next consider the introduction of 50 percent warp during recognition by GSR. After matching s_n, the test for s_{n+1} occurs at $1.5\Delta t$ seconds later. The additional $0.5\Delta t$ seconds

Table 2: Percentage accuracy in predicting the penultimate symbol in an embedded Reber sentence when warp is introduced before the critical symbol. All sentences were stored with $\sigma_s = 0.0001$. σ_T is the test (or retrieval) tolerance. The term *100% warp* means that the time between the test symbol and its predecessor is twice as long as during storage. When retrieval is attempted at wider tolerances, for example, $\sigma_T \geq 0.005$, the accuracy is still excellent up to 200% warp.

σ_T	No Warp	100% Warp	200% Warp
0.001	91.0	69.6	47.8
0.003	92.8	77.6	63.6
0.005	96.4	95.0	93.2
0.008	98.6	98.8	99.0
0.01	99.6	99.6	99.8

will result in a difference of $\Delta\mathbf{G}$, between the current $\mathbf{G}$ and the value stored for the transition $s_n \rightarrow s_{n+1}$ in the same context. The magnitude of the error $\Delta\mathbf{G}$ will depend on the free parameters in the reaction-diffusion equations, which are not linear. But $\|\Delta\mathbf{G}\|$ will increase, in general, with warp. When $\|\Delta\mathbf{G}\| > \sigma_T$, the warp will have exceeded the ability of the TSP to recognize the transition $s_n \rightarrow s_{n+1}$ as valid.

An experiment was conducted to measure the relationship of the error to warp. The sequences generated by the embedded Reber grammar generator were stored using one step per symbol.[6] During retrieval, however, two or three times as many symbol steps were applied for the transition to the penultimate symbol, effecting a warp of 100 to 200 percent between the $(n-2)$th and $(n-1)$th symbols. The correct or incorrect recognition (*known* or *unknown*) of the $(n-1)$th symbol was the error criterion. The retrieval error rate was measured as a function of the tolerance, σ_T (see Table 2). As σ_T increases, we are permitting an increasing tolerance for acceptance of the comparison of a stored sequence with the external sequence.

The results show the error to be rather high when the retrieval tolerance is tight ($\sigma_T \leq 0.001$). Relaxation of σ_T largely overcomes the warp but decreases depth and the corresponding resolution of sentence distinctions. Somewhat greater warp tolerance might be obtained with the use of slower learning rather than one-shot learning. This is one of many trade-offs that can be juggled by the choice of storage and test tolerances.

4.7 Sequentially addressed sequence memory

We defined the SASM mode in section 2 as the case in which the stored sequences are relatively long and the content address is sequentially presented. The responsive sequence(s) are then retrieved. Addressing follows

[6]Four integrations per symbol step were used. This permitted each symbol's effect to reach anywhere in a 5×5 array of cells.

the guided sequence retrieval (GSR) algorithm, leading the TSP to recapitulate the training steps and set up the symbol concentrations that occurred during training. Subsequent retrieval of the responsive sequence is performed by free sequence retrieval (FSR).

In FSR, there is no external symbol input. At the end of GSR (the addressing operation), the symbol concentration, $\mathbf{G}(x, y)$, has been established and the processor then follows FSR, which is a sequence of cell activations determined solely by global-maximum activation (global WTA), analogous to a falling domino pattern.

The FSR algorithm follows. Its basic simplicity is complicated by the rather stringent demands we will place on it by choice of task, which will be discussed shortly.

Free Sequence Retrieval Algorithm

1. Find the winning cell (most highly activated globally), $C_i \epsilon \mathbf{C}$, corresponding to
$$\forall \mathbf{C} : \min_{k=1}^{N}(\|(\mathbf{V}_k - \mathbf{G}(x, y)\|) = MIN$$
where k extends over all V registers in all cells and MIN is the global minimum.
2. *Tentative path A*: Emit a unit of symbol s_{C_i} at (x, y) and Diffuse one symbol step. Find the most highly activated cell, C_{ja}, and the corresponding minimum activation, MIN_A.
3. *Tentative path B*: Diffuse one step (without an Emit). Find the most highly activated cell, C_{jb}, and the corresponding minimum activation, MIN_B.
4. Select path A if $MIN_A < MIN_B$ and $MIN_A < \sigma_T$.
Otherwise, select path B if $MIN_B < MIN_A$ and $MIN_B < \sigma_T$. Otherwise, terminate with *no path.*
5. Actualize the steps of the winning path, that is:
Designate C_{ja} or C_{jb} as the AC. If path A, actualize the Emit + Diffuse one step; else actualize only the Diffuse step.
6. Return to step 1.

4.7.1 Selection of a data type

There are several questions which deserve to be distinguished because they present different problems.

- *Addressing*: Can addressing utilize either initial or internal subsequences?
- *Sequence complexity*: Is symbol duration stored and can it be retrieved? Does the presence of substrings that are common within stored strings cause any difficulty for retrieval?

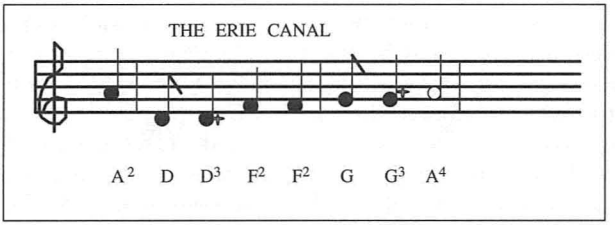

Figure 9: Example of musical melody encoding. A superscript represents time length in eighth notes. F^2F^2 is an example of a "reattacked" note. One eighth note corresponds to a fixed number of integration steps (usually four) of the reaction-diffusion equations.

To study all of these questions, we looked for a data type for which all cases can appear. Such a data type is the musical score of simple melodies.

First, a system for encoding a single melody of a music score was devised (see Figure 9). The music unit time was chosen to be an eighth note of the reaction-diffusion equations. For each eighth note, four integration steps of the reaction-diffusion equations were performed. Four steps ensures that an emitted pulse of symbol will travel to every location in the TSP structure so that any next symbol can link with its predecessor. The appearance of a base symbol (e.g., $C, D, F\#$) was coded as an activation of the single cell of the corresponding symbol (e.g., A, B, C#). Relative time length in eighth notes is denoted by a superscript.[7]

To distinguish held notes from reattacked notes notationally, we will write C^2C to represent a C quarternote followed by a C eighthnote, and C^3 represents a dotted quarternote or, more precisely, a C tone held for 3 units of time.

At the storage/training algorithm, level, the two sequences C^2C and C^3 are interpreted differently. The former means "Emit C and Diffuse two steps, then Emit C and Diffuse one step," whereas the latter means "Emit C and Diffuse three steps." "Diffuse one step" corresponds to four integrations of the reaction-diffusion equations in the following experiments.

An example of a music score to notation mapping is shown in Figure 9.

4.8 Experiments with SASM

Test 1: Distinguishing s^k from $sss\cdots s$ (k times)

For the first test the following strings were stored using $\sigma_s = 0.0001$. The parameters used were $D_g = 0.4$, $D_r = 0.3$, $\alpha = 0.3$, $\beta = 0.3$, $\epsilon = 0.2$, and $\gamma = 0.1$.

```
A B C D | C B A E
E B C D | F F F G
G B C D | F^3 D
```

[7]For a single eighth note, superscript 1 defaults to blank.

The sequences were addressed (by GSR) up to the |, and the remainder of the sequences were retrieved by the FSR algorithm. The objective was to test the ability of the TSP to distinguish the FFF case from the F^3 case based on the initial symbol of the address, E or G.

The retrieved strings were exactly as stored. The sequence of **G** vectors experienced during retrieval was exactly the same as occurred during storage since σ_s was small enough to give a depth of more than 4, the distance from the last unique symbol to the point where the sequences diverge.

In this case, not only does the difference among the initial four symbols of each sequence predict the fifth symbol correctly, but the more difficult task of distinguishing FFF from F^3 is achieved.

Note that there is no problem due to the common (BCD) subsequences here or, in general, as long as the effect of the first symbol is not confounded by the depth being too small (i.e., by σ_s being too small).

Test 2: Variable length symbols in the address

Suppose the same number of appearances of a symbol occurs in two addresses, but the time lengths of symbol appearance are different. Can the TSP avoid confusion? To test this, we store the following two sequences.

```
A B C F F^2 F | E D C
A B E F F F F | G F E
```

Again, using $\sigma_s = 0.0001$, which is small enough to ensure that the context of each transition is uniquely represented, there was no problem at all in retrieving the correct sequence following the | by FSR. All parameters were as in test 1 except that $D_g = 0.3$ proved to be a better choice.

Test 3: Sequences with long common subsequences

Finally, we stored sequences having common subsequences to test the ability of the TSP to avoid confounding the sequence during retrieval. The stored sequences, with the address sections shown, are

```
F  A^4 | B D E F G B C D^5
F^2 D  | E F G A^3 B^4
F^3 C  | D E F G A^2 B^3
B^2 C^3 E^2 D E F   | C^3 B^2 A C
B^2 C^3 E^2 D E C^4 | B^2 C A
B^3 C^3 E^2 D E C^2 | E^4
```

The common subsequences were in the free retrieved section in some cases and in the address section in other cases. Using the same parameters as before, with the exception that $D_g = 0.2$, all sequences could be retrieved exactly—meaning both the symbols (tones) and their duration.

4.8.1 Comments on the FSR algorithm

We can now appropriately discuss the complication in the FSR algorithm involving the need to test two possible actions at each step (see steps 2 and 3

in the FSR algorithm, section 4.7). If one permitted only sentences having a single symbol per step, the simple rule of the emission of a unit of a symbol on each transition would be adequate. The need for the test arises from the fact that we want to distinguish cases of "held" notes versus "reattacked" notes. For example, in the subsequences $\ldots BC^4B\ldots$ and $\ldots BCC^2CB\ldots$, each step through the region of Cs requires testing whether the active cell supports a path corresponding to (Emit + Diffuse) = "reattacked note" or a path corresponding to (Diffuse only) = "held note." If the sequences were restricted to only an "attacked note" for each symbol transition, the test of alternative possibilities seen in the FSR algorithm would be unnecessary. But with the allowance of successive appearances of the same symbol either held or reattacked, the way to distinguish the preferred path is by testing the two possibilities corresponding to path A and path B in the FSR algorithm.

Also note that the TSP does not support addressing that begins at an *internal* subsequence of a *stored string*. The reason for this limitation is that an approximate value of the **G** vector at the internal symbol where addressing begins would have to be known in order to continue the FSR correctly. For addressing with an initial substring, the addressing always begins with $\mathbf{G} = 0$ and the GSR develops the complex distribution of symbol concentrations.

5. Discussion

We have proposed and studied a temporal sequence processing system that uses the reaction-diffusion process to provide interconnection of all cells and convey memory of past events.

The cells are provided with a multiplicity of inputs that are capable of activation due to the symbol concentration at the location of the cell, producing a cell like a multi-input radial basis function cell. Reaction-diffusion provides for controlled growth and decay processes, rather than just decay and diffusion, for the concentrations, which adds another dimension of possibilities to the sequence representations contained in the TSP and is a unique feature. Other designs having similar objectives [10, 11] limit themselves to monotonic decay of memory of past events.

The experimental study of the ReDi TSP was keyed to five qualities, which are, briefly, depth, flexibility of equivalence class representation, warp tolerance, minimum storage, and content addressability. Various combinations of these qualities are preferred for various applications.

The present system permits any depth desired but at an increasing cost in storage capacity for increasing depth. A unique feature of the system is that it allows flexible class equivalence at retrieval time. Once the sequences are stored with a specific degree of uniqueness of the transitions, they may be retrieved (or recognized) using the same or a lesser degree of stringency in symbol matching.

Storage efficiency, in relative terms, is controllable by a single parameter, σ_s, which sets the radius in symbol concentration space within which sequence transitions are recognized and stored as unique. Thus, storage

efficiency and its inverse, depth, are easily controlled by selection of the parameter σ_s.

The system shows a moderate amount of warp tolerance, but this is achieved at the cost of accuracy in identifying the stored sequence.

Two principal tasks were examined for the purpose of evaluation of the system: embedded sequence recognition (ESR) and sequentially addressed sequential memory (SASM). In ESR, stored short sequences are examined for their occurrence in a longer argument string. Both tasks are accomplished by combinations of the more fundamental algorithms, guided and free sequence retrieval (GSR and FSR).

For the ESR problem, the external sequence guides the inquiry (addressing), which is, operationally, guided sequence retrieval. This algorithm was tested using an embedded Reber grammar to generate short sequences for storage. Other sequences were applied as arguments that, although not necessarily physically longer, provided an equivalent test to the case of an extensive argument string. Successful events were those that correctly reached and predicted the penultimate symbol. This test was fully successful, reaching accuracies up to 98 percent for particular values of σ_s and σ_T. A test of false-positive responses showed that the accuracies attributed to true-positive cases were largely valid.

For the SASM problem, the external sequences are short and the internal sequences are long. The external sequence is used as an address, applied by the GSR algorithm, and the remainder of the internally stored string is retrieved by FSR in a method analogous to following the lowest energy path from the point where the address ends. We confined ourselves to addressing from the *initial subsequences of the stored sequences only*; addressing beginning internal to the stored sequences is not possible, in general, with the TSP.[8]

The TSP has no problem at all performing the SASM problem for ordinary complex sequences that are stored and retrieved with one symbol per time step. In order to demonstrate the greater capabilities of the system, we introduced and stored simple melody-like sequences requiring that both the symbols and their time duration be correctly retrieved. This experiment was also fully successful, although it required an unwanted complication in the FSR algorithm in the form of a test-and-choose procedure to correctly identify the "held" versus the "reattacked" case when the same symbol recurs.

Comparing the TSP to the method presented in [11], their approach appears to be more tolerant to warp than ours. Their approach requires preliminary knowledge of the necessary depth before commencing storage of sequences. The feature of the TSP method that allows adjustment of recognition tolerance during the recognition phase is not supported, as far as we can determine, in the method in [11]. The important property of "chunking," however, is one we have not yet addressed, although we do not see any inherent difficulty in extending the TSP to perform chunking.

[8] A modified system that permits addressing from internal sequences has been studied by us and will be presented elsewhere.

References

[1] Turing, Alan, "The Chemical Basis of Morphogenesis," *Phil. Trans. of the Royal Soc. of London, Ser. B*, **237** (1952) 37–72.

[2] Ray, S.R. & Gyaw, T.A., "Universal Multichannel Signal Interpretation Using Connectionist Architecture," in *Proceedings of Artificial Neural Networks in Engineering Conf., V.4*, (ASME Press, New York, 1994).

[3] Cleeremans, Axel, *Mechanisms of Implicit Learning* (MIT Press, Cambridge, MA, 1993).

[4] Mozer, Michael C., "Neural Net Architectures for Temporal Sequence Processing," in *Predicting the Future and Understanding the Past*, edited by A. Weigend & N. Gershenfeld (Addison-Wesley, Redwood City, CA, 1993).

[5] Elman, Jeffrey, "Finding Structure in Time," *Cognitive Science*, **14** (1990) 179–211.

[6] Jordan, M.I., "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine," in *Proceedings of 8th Annual Conference of the Cognitive Science Society*, 1987, 531–546.

[7] Cummings, F., "Representation of Temporal Patterns in Recurrent Neural Networks," in *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1993, 377–382.

[8] Fahlmann, S.E., "The Recurrent Cascade-correlation Architecture," in *Advances in Neural Information Processing Systems*; NIPS-3, Proceedings of the 1990 Conference, held at Denver on November 26–29, 1990, edited by Lippmann, Moody, and Touretzky (Morgan Kaufmann, Palo Alto, CA, 1990).

[9] deVries, B. and Principe, J.C., "The Gamma Model–A New Neural Net Model for Temporal Processing," *Neural Networks*, **5** (1992) 565–576.

[10] Bodenhausen, Ulrich and Waibel, Alex, "Learning the Architecture of Neural Networks for Speech Recognition," *IEEE Proceedings of the ICASSP*, **1** (1991) 117–124.

[11] Wang, DeLiang and Arbib, Michael, "Timing and Chunking in Processing Temporal Order," *IEEE Transcripts on SMC*, **23** (1993) 993–1009.

[12] Meinhardt, Hans, *Models of Biological Pattern Formation* (Academic Press, 1982).

[13] Bohn, H., Interkalare Regeneration und segmentale Gradienten bei den Extremitaten von Leucophaea-Larven (Blattaria). III. Die Herkunft des interkalaren Regenerats. *Wilhelm Roux Arch.*, **167** (1971) 209–221.

[14] French, V., "Leg Regeneration in the Cockroach, Blattella germanica I. Regeneration from a congruent tibial graft/host junction," *J. Embryol. exp. Morph.*, **179**, (1976) 57–76.