Neural, Parallel & Scientific Computations 3 (1995) 1-14

ON MULTI-PROPERTY SET OPERATIONS USING NEURAL NETWORKS AND SYSTOLIC ARRAYS

Margaritis K.G.[1,2], Goulianas K.[1], Adamopoulos M.[1], Tsouros K.K.[1] and Evans D.J.[2]

[1]Informatics Centre, University of Macedonia,
Thessaloniki 54621, Greece.

[2]Parallel Algorithms Research Centre, University of Technology,
Loughborough Leics., LE11 3TU, U.K.

Abstract

This paper presents artificial neural networks and systolic architectures that detect multi-property relations satisfied by the elements of a set, that is: elements having some common properties; satisfying at least one of some given properties; not satisfying all given properties; not satisfying any of some properties. The operations discussed are initially performed by means of a feedforward artificial neural network which encodes in its interconnection matrices the values of the properties of the set members. Then the systolic implementation of the neural network is addressed. Finally the ability of modifying, adding or deleting elements or properties is discussed.
Keywords - neural networks, systolic algorithms, set operations

1. INTRODUCTION

Constructing multi-property relations between the elements of one or more sets is an important component of many computing problems, including various applications of searching, sorting and matching problems in large data or knowledge bases. Performance is especially crucial when real time response is required and several parallel algorithms as well as special purpose architectures have been proposed (see for example Akl, 1989; Herath, 1992). In this paper the utilization of artificial neural networks and systolic arrays is discussed for the implementation of real time searching techniques which are directly implementable onto high performance neurocomputing systems or more general VLSI processor array architectures (Kung, 1987; Kung, 1993; Przytula & Prasanna, 1993).

The basic problem can be stated as follows. Given a set $\Omega = \{\omega_1,\omega_2,...,\omega_m\}$ with m elements, where each element ω_j ($1\le j\le m$) has k generic properties P_i ($1\le i\le k$) and each generic property is a set having m elements, i.e. the actual values of the property for each ω_j ($1\le j\le m$). We seek to detect the elements of Ω that satisfy some relations between q ($1\le q\le k$) given property values. Let us define the property matrix $\wp$ of the property values as follows :

Received February 17, 1994

$$\wp = \begin{bmatrix} P_{11} & P_{12} \ldots P_{1m} \\ P_{21} & P_{22} \ldots P_{2m} \\ \ldots & \ldots \ldots \ldots \\ P_{k1} & P_{k2} \ldots P_{km} \end{bmatrix}$$

where P_{ij}, $(1 \le i \le k)$, and $(1 \le j \le m)$, will be the actual value of generic property P_i for the element ω_j. For example, if the m members in a database have the k properties $(P_1, P_2, \ldots, P_k)$ = (P#,PNAME,...,PTEL), then the matrix $\wp$ of the values of the above properties for the m members of the database would be :

$$\wp = \begin{bmatrix} P\#_1 & P\#_2 & \ldots & P\#_m \\ PNAME_1 & PNAME_2 & \ldots & PNAME_m \\ \ldots & \ldots & \ldots & \ldots \\ PTEL_1 & PTEL_2 & \ldots & PTEL_m \end{bmatrix}$$

where column j, $(1 \le j \le m)$, contains the values of the k properties of member j, while line i, $(1 \le i \le k)$, contains the m values of property P_i.

Definition 1: The mapping

$$f_a : (\omega_j : \omega_j \in \Omega,\ 1 \le j \le m) \xrightarrow{f_a} (P_{1j}, P_{2j}, \ldots, P_{kj}) \qquad (1)$$

is a relation between each element $\omega_j \in \Omega$ and the values of the k properties of that element (i.e. column j of the matrix $\wp$).

Definition 2: The mapping

$$f_b : (P_i : P_{ij} \in \wp,\ 1 \le i \le k,\ 1 \le j \le m) \xrightarrow{f_b} (P_{i1}, P_{i2}, \ldots, P_{im}) \qquad (2)$$

is a relation between each generic property P_i and the actual values of the propery for the elements of Ω (i.e. row i of the matrix $\wp$).

Definition 3 (Intersection): The mapping

$$f_c : (P_{ij} : P_{ij} \in \wp,\ 1 \le i \le k,\ 1 \le j \le m) \xrightarrow{f_c} R_{ij} = \{\omega_r : P_{ir} = P_{ij},\ 1 \le r \le m\} \qquad (3)$$

is a relation between the value of property P_i for element $\omega_j \in \Omega$ to the set $R_{ij} \subseteq \Omega$, which contains the elements $\omega_r \in \Omega$ $(1 \le r \le m)$ that have the same value for the above property, thus creating the matrix $\Re$, with elements R_{ij}, $(1 \le i \le k)$ and $(1 \le j \le m)$, with $R_{ij} \subseteq \Omega$:

$$\Re = \begin{bmatrix} R_{11} & R_{12} \ldots R_{1m} \\ R_{21} & R_{22} \ldots R_{2m} \\ \ldots & \ldots \ldots \ldots \\ R_{k1} & R_{k2} \ldots R_{km} \end{bmatrix}$$

The element R_{ij} can be seen as the result of a single property intersection operation that, for a given input value P_{ij} of a generic property P_i produces as its result a subset which contains all elements of Ω that have the same value for the generic property P_i (see also

Evans et al, 1990). The multi-property extension of the intersection operation produces as result the subset that contains all elements that satisfy a given set of kl input property values (1≤kl≤k). Thus, if the set of input property values is $P_{i_1}, P_{i_2}, \ldots, P_{i_{kl}}$, where subscripts i_p, ($1 \le i_p \le k$, p=1,2,..., kl≤k), is a random subset of {1,2, ..,k}, then the multi-property intersection can be defined as $\bigcap_{p=1}^{kl \le k} R_{i_p,j}$, j=1,2,...,m.

Definition 4 (Union): The mapping f_c can be used for the definition of the multi-property union as an operation that produces as its result a subset which contains all elements of Ω that fulfil at least one of the given property values. Thus, for a set of input property values is $P_{i_1}, P_{i_2}, \ldots, P_{i_{kl}}$, we should select the columns j of matrix $\Re$ such that $\exists\, i_p$, ($1 \le i_p \le k$, p = 1, 2, ..., kl ≤ k) : $\omega_{i_p} \in R_{i_p,j}$. Thus, the multi-property union can be defined as $\bigcup_{p=1}^{kl \le k} R_{i_p,j}$.

Definition 5 (Complement): The complement of a multi-propery intersection is defined as $\Omega - \bigcap_{p=1}^{kl \le k} R_{i_p,j}$ and it can be interpreted as the subset of Ω with elements that do not satisfy all the input property values. Similarly the complement of a multi-property union is defined as $\Omega - \bigcup_{p=1}^{kl \le k} R_{i_p,j}$ and it can be interpreted as the subset of Ω that do not satisfy any of the input property values.

2. NEURAL NETWORK FOR MULTI-PROPERTY INTERSECTION

Initially the case of multi-property intersection is discussed. The neural network used is a simple feedforward network (see Simpson, 1990; Wasserman, 1989; Kung, 1993 for general introduction), consisting of three layers: layer 0 is the input layer with k blocks of n neurons in each block, where n is the number of bits required for encoding the value of each property in bipolar form. Layer 1 consists of k blocks of m neurons in each block, where m is the number of the elements of the set, with block p of layer 0 fully connected with block p of layer 1. Layer 2, which is the output layer, consists of m neurons connected only with the corresponding neurons in each block of neurons at Layer 1.

2.1. Interconnection weights

We assign random values (0,1) to the connections between the neurons of layer 0 and layer 1, so that a three-dimensional synaptic matrix $W^{0\to1}$ of size (kxnxm) is defined. The interconnection weights between neurons of layers 1 and 2 are valid, i.e. they have weight equal to 1 only if they obey the interconnection structure discussed before. For the synaptic matrix $W^{1\to2}$ defined as a three-dimensional matrix of size (kxmxm) it can be said that $w_{pij}^{1\to2} = 1$ if and only if i=j, for (1≤p≤k), (1≤i≤m) and (1≤j≤m).

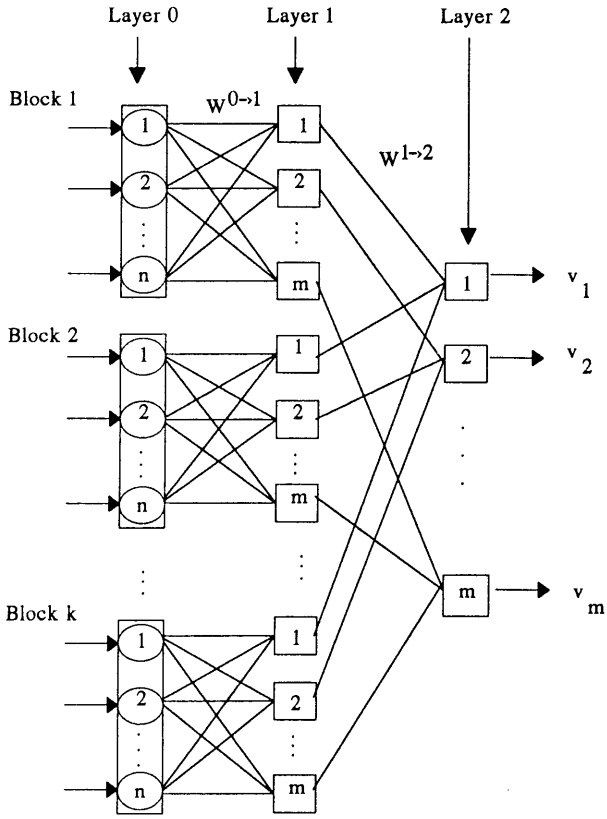

Figure 1. Neural Network for Intersection

2.2. Activation functions

For the neurons at layer 1 we use as activation function the Hard Delimiter function with threshold $\theta = n - 1$:

$$f_1(u) = \begin{cases} 1, & u > \theta = n-1 \\ 0, & u \le \theta = n-1 \end{cases} \tag{4}$$

For the neurons at layer 2 we use as activation function the Hard Delimiter function with threshold $\theta = kl - 1$, where kl is the number of the common properties we search for :

$$f_2(u) = \begin{cases} 1, & u > \theta = kl-1 \\ 0, & u \le \theta = kl-1 \end{cases} \tag{5}$$

2.3. Training (Encoding)

Initially the values of the k properties are encoded in the interconnections between layers 0 and 1. The synaptic matrix $W^{0\to1}$ is of size (kxnxm) so that each one of the k sub-matrices with size (nxm), defined as $W_q^{0\to1}$ $(1 \le q \le k)$, keeps the values of property P_q in bipolar format. More specifically the encoding has the form

$$w_{qij}^{0\to 1} = P_{qj}^{i}, \quad q = 1,2,..,k, \quad i = 1,2,..,n, \quad j = 1,2,..,m \tag{6}$$

where P_{qj}^{i} is the i^{th} bipolar digit of the j^{th} value of property P_q. The above encoding can be seen as a simple training procedure which is performed by presenting the values of the properties for each member ω_j of the set ($1 \le j \le m$), at the input layer 0 and using the Kohonen learning (Kohonen, 1984)

$$w_{qij}^{0\to 1} = w_{qij}^{0\to 1} + \alpha(P_{qj}^{i} - w_{qij}^{0\to 1}), \quad q = 1,2,..,k, \quad i = 1,2,..,n \tag{7}$$

where $\alpha = 1$ is the learning coefficient since only one input vector is to be associated with each neuron at layer 1, and the class is known (value j of property P_q is to be stored in the weights between all n neurons of block q of layer 0 and neuron j of block q of layer 1). Notice that layer 1 is trained with only one calculation per weight, by presenting the k values of properties for each one of the m members of the set each time, so that a total of m training phases are required.

2.3. Network Operation (Recall)

During the recall phase the values of the k1 properties in question are presented to the input layer of the network in bipolar form, so that block q of layer 0 accepts the required value for the q^{th} property. More specifically, if we denote the input pattern as a two-dimensional array X of size (kxn), we can say that :

$$x_{qi} = \begin{cases} X_q^i, & \text{if we search for a value of property } P_q \\ 0, & \text{otherwise} \end{cases}, \quad q=1,2,..,k, \quad i=1,2,..,n \tag{8}$$

where X_q^i, ($1 \le q \le k$, $1 \le i \le n$), is the i^{th} bipolar digit of the value in question for the q^{th} property. Further, if $Q = \{i_p : 1 \le i_p \le k, \ p = 1,2,..,k1 \le k\} \subseteq \{1,2,..,k\}$, a subset with the order of the properties in question, we will have non-zero inputs only to those i_p blocks of neurons in the input layer 0. The input to neuron j, ($1 \le j \le m$), of any block q, ($1 \le q \le k$) of layer 1 will be :

$$u_{qj} = \sum_{i=1}^{n} w_{qij}^{0\to 1} \, x_{qi} = \begin{cases} \sum_{i=1}^{n} P_{qj}^{i} X_q^i, & \text{if we search for property } P_q, \ 1 \le q \le k, \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where $P_{q,j}^{i}$ is the stored value of property P_q for ω_j. Further, if $q = i_p \in Q$ the output of that neuron will be $v_{i_p,j} = f_1(u_{i_p,j})$. Therefore, the output will be $v_{i_p,j} = 1$, only for $u_{i_p,j} = n$, that is only if $P_{i_p,j}^{i} = X_{i_p}^{i}$. Thus, at block i_p of layer 1, the neurons which correspond to the members of Ω that satisfy the search value X_{i_p} of the generic property P_{i_p} (i.e. the members of the subset $R_{i_p,j}$) will be activated. The input $u_{i_p,j}$ to every other neuron j' of block i_p at layer 1, ($j' \notin R_{i_p,j}$) that does not satisfy this property, will be $u_{i_p,j'} < n$, so that the output of these neurons will be $v_{i_p,j'} = 0$.

The output of layer 1 forms a two-dimensional array V of size (kxm) which serves as input to layer 2 through the interconnection matrix $W^{1\to2}$. Thus the input to each neuron j of layer 2, ($1\le j\le m$) will be $u'_j = \sum_{q=1}^{k} w^{1\to2}_{q,j,j}\, v_{q,j} = \sum_{q=1}^{k} v_{qj} = \sum_{p=1}^{kl} v_{i_p,j} \le kl$, since the interconnection weights are valid (their value is 1) and $v_{q,j} = 0$, when $q \notin P_1$. The output of that neuron will be $v'_j = f_2(u'_j)$. Therefore, the output will be $v'_j = 1$, only if $u'_j = kl$, or element j satisfy all the kl properties, i.e. only if $\omega_j \in \bigcap_{p=1}^{kl\le k} R_{i_p,j}$, $1 \le kl \le k$, $1 \le i_p \le k$. The input at every other neuron j' of layer 2 that does not satisfy all the kl properties will be $u'_j < kl$, and its output $v'_j = 0$.

3. OTHER MULTI-PROPERTY RELATIONS

3.1. Multi-property union

The network used for implementing the multi-property union is the same as in Fig. 1, except for the activation function f_2, where we use the Hard Delimiter Function with threshold $\theta = 0$. Thus, during the recall phase the values of the kl properties in question are presented to the input layer of the network in bipolar form, as discussed at section 2. So, at block i_p of layer 1, the neurons which correspond to the members of Ω that satisfy property $P_{i_p} = X_{i_p}$, (i.e. the members of the subset $R_{i_p,j}$) will be activated. The input $u_{i_p,j}$ to every other neuron j' of block i_p at layer 1, ($j' \notin R_{i_p,j}$) that does not satisfy this property, will be $u_{i_p,j'} < n$, so that the output of these neurons will be $v_{i_p,j} = 0$. The output of layer 1 forms a two-dimensional array V of size (kxm) which serves as input to layer 2 through the interconnection matrix $W^{1\to2}$. Thus, the input to each neuron j of layer 2, ($1\le j\le m$) will be $u'_j = \sum_{q=1}^{k} w^{1\to2}_{q,j,j}\, v_{q,j} = \sum_{q=1}^{k} v_{q,j} = \sum_{p=1}^{kl} v_{i_p,j} \le kl$, since the interconnection weights are valid (their value is 1) and $v_{q,j} = 0$, when $q \notin P_1$. The output of that neuron will be $v'_j = f_2(u'_j)$. Therefore, the output will be $v'_j = 1$, only if $u'_j > 0$, or element j satisfy at least one of the kl properties, i.e. only if $\omega_j \in \bigcup_{p=1}^{kl\le k} R_{i_p,j}$ $1 \le kl \le k$, $1 \le i_p \le k$. The input at every other neuron j' of layer 2 that do not satisfy at least one of the kl presented properties will be $u'_j = 0$, and its output $v'_j = 0$.

3.2. Complement of Multi-property Intersection

The neural network of Fig. 1 is again used, with the following differences in the interconnection weights and the activation functions. The interconnection weights between neuron j, ($1\le j\le m$) at block p, ($1\le p\le k$) of layer 1, and neuron j, ($1\le j\le m$) of layer 2 will be -1 instead of 1: $w^{1\to2}_{q,j,j} = -1$, while the remaining connections are zero. For the neurons at layer 2 we use as Activation Function the Hard Delimiter Function, with $\theta = -kl$, where kl is the number of properties we search for :

$$f_2(u) = \begin{cases} 1, u > -kl \\ 0, \ u \leq -kl \end{cases} \quad (10)$$

Thus, at each block of layer 1 only those neurons that correspon to the members that satisfy the property P_q will be activated, while the output of all other nodes will be zero. At layer 2, the weighted sum for the neurons that satisfy all the properties will become, via the negative weights, -kl, where kl is the number of the properties we search for, and will produce a zero. The rest of the neurons, which belong to the complement of the intersection will produce the unity, which is the desired result.

3.3. Complement of Multi-property Union

The network for the Complement of the Intersection of Properties is used, with the only difference that in the activation function for layer 2 the threshold is -1 instead of -kl :

$$f_2(u) = \begin{cases} 1, u > -1 \\ 0, \ u \leq -1 \end{cases} \quad (11)$$

Layer 1 works as before while at layer 2, the weighted sum for the nodes that satisfy at least one up to kl properties, after passing via the negative weights, will range from -1 to -kl, where kl is the number of the properties we search for, and will produce a zero. The remaining neurons, which belong to the complement of the union will produce the unity, and the output of layer 2 will give the required subset.

4. SYSTOLIC ARRAY IMPLEMENTATION

The systolic implementation for searching for single-property relations is discussed in Margaritis & Evans, 1992 and Margaritis et al, 1992. The extension to multi-property relations is discussed by means of the example of the multi-property intersection, based on the bit-level rectangular systolic array of Fig. 2.

4.1. Mapping of neural network onto systolic array

The basic building block of the systolic implementation of a neural network is a matrix vector multiplication systolic array which corresponds to the feedforward recall operation of one network layer. Starting from layer 1 we can see that it consists of k blocks, each one performing a single-pass full matrix vector multiplication of size (nxm), i.e. it compares the input value of property X_q ($1 \leq q \leq k$) with the corresponding property values of the m set elements in a bit-wise fashion. Thus, a number of k rows of linear arrays, one for each property, is required to accommodate the input of matrix X of size (kxn). The choice of the linear array type is discused in Margaritis & Evans, 1992: herein the ring-like linear array is used augmented with an output mechanism so that the result can be produced systolically through the vertical channels. The result of each linear array (i.e. a block of layer 1) is a row vector V_q (1xm) so that finally the output matrix V of size (kxm) is formed. Layer 2 performs a dimensionality reduction through a summation of each column of matrix V thus producing as result a row vector U of size (1xm), denoting the cardinality of the set elements the satisfy the multi-property intersection chosen. As it is shown in Fig. 2 this operation can be performed by means of vertical summation so that the output vector U is produced through a pipeline at the bottom of the array.

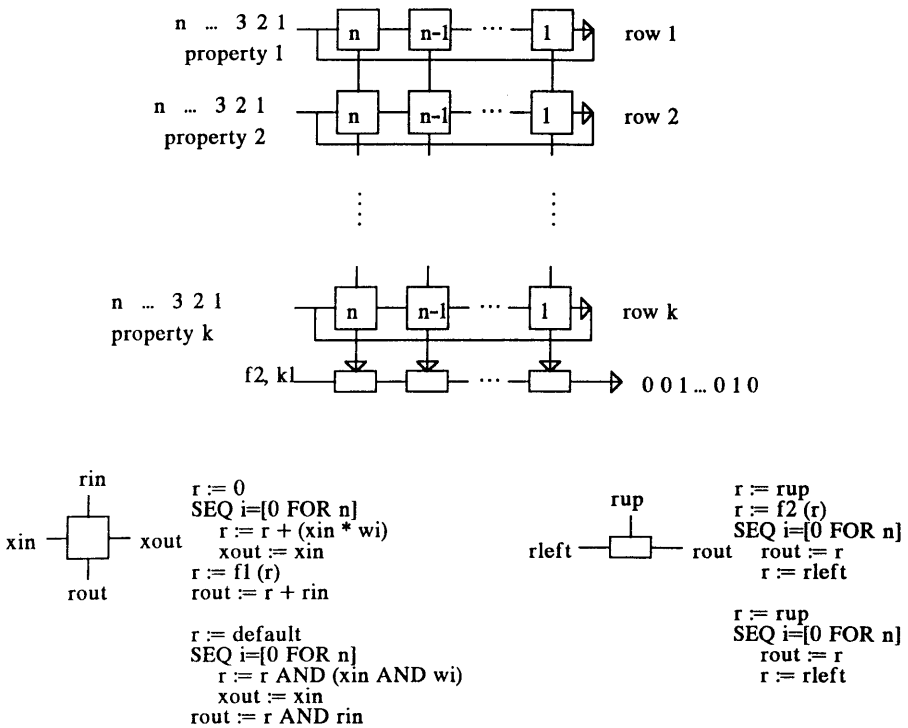

Figure 2. Bit-level Systolic Array for Intersection

4.2. Recall phase array operation

Initially we discuss the recall phase in more detail. The q^{th} row linear array, ($1 \le q \le k$), accepts as input property vector $X_q = x_{q1}, x_{q2}, ..., x_{qn}$ in bipolar format. The input is delayed by q-1 time steps, thus forming a skewed input wavefront for matrix X, as shown in Fig.2. After n steps the property reaches the end of the array and it is circulated back through the ring-like connection. The actual computation starts as soon x_{q1} reaches cell 1. Each cell keeps in local memory the values of the corresponding property for a set element in bipolar format. Thus, in n steps the j^{th} cell ($1 \le j \le m$) performs a bit-level inner product step operation $u_{qj} = \sum_{i=1}^{n} w_{qij}^{0 \to 1} x_{qi}$ followed by the non linear neuron function $v_{qj} = f_1(u_{qj})$. After n steps the result is passed down towards the q+1 array, which completes its operation with one step delay. The two results are added, thus performing part of the operation of layer 2, i.e. $u'_j = \sum_{q=1}^{k} v_{qj}$. The final sum is passed to the bottom row linear array which performs the non-linear function of layer 2, that is $v'_j = f_2(u'_j)$. The output of the bottom row linear array is a string of 1's or 0's with the position of 1's in the string denoting the position of a valid set element. Notice that the array processes n set elements at a time. Notice, however that the processing of the next group of elements overlaps exactly with the previous group. Thus, after an initial input delay of 2n+k steps the results are produced in a continuous string in m steps, so that the total computation time is 2n+k+m bit-level inner product steps. The area required is the computation is (k+1)n bit-level cells.

The number of properties that are taking place in the computation as well as the type of multi-property relation that is being sought is important in order to determine

the form and the thresholds of function f_2. This information enters the bottom row array together with the first set of input data and travels systolically through the array so that the appropriate choice of thresholds is made. Notice that in the case of the complement operations the valid interconnection weights between layers 1 and 2 are equal to -1. Thus, for those operations f_2 should be applied to -u instead of u.

The bit-serial inner product step discussed is actually a bit-level comparison of a given input property with the corresponding property value of a set element (see Margaritis et al, 1992). This comparison-based operation is detailed in the cell descriptions of Fig. 2. Therein two alternative cell descriptions are presented for the systolic array for multi-property intersection. The first description is based on the multiplication and summation concept, while the second is based on boolean operations. Thus, for the second alternative the computation can be described as follows. In n steps the j^{th} cell ($1 \le j \le m$) performs a bit-level logical operation $u_{q,j} = \bigcap_{i=1}^{n} (w_{qij}^{0 \to 1} \bigcap x_{qi})$. After n steps the result is passed down towards the $q+1^{th}$ array, which completes its operation with one step delay. The two results are logicaly combined, thus performing part of the boolean operation of layer 2, i.e. $u'_j = \bigcap_{q=1}^{k} v_{qj}$. The final result is passed to the bottom row linear array which simply forms the output data string which is a stream of boolean 1's or 0's with the position of 1's denoting the set elements satisfying the chosen multi-property intersection. It is evident that the second alternative is more economical to implement. In order to implement other multi-property relations (union or complements), in the first alternative we have to modify the neuron functions f_1 and f_2, as explained in section 3. If the second alternative is opted then the boolean operations should be modified accordingly. Thus, we should have

rout := r OR rin,	for multi-property union
rout := (NOT r) OR rin	for intersection complement
rout := (NOT r) AND rin	for union complement

Notice that the input and stored property values should be in similar boolean format. Further, the type of multi-property relation searching must be known to all cells. Thus, the input of the property values should be accompanied by a flag indicating the operation to be performed. Finally the initial values should change accordingly (i.e. logical 1 for AND and logical 0 for OR).

4.3. Synaptic matrix allocation and timing schedule

Each cell of the rectangular array, except the bottom row linear array, keeps in its local memory the property values of some set elements in bipolar or boolean format. The exact allocation scheme is shown below, as well as the array timing schedule, i.e. time step that these elements are accessed during the array operation. Fig. 3 depicts the allocation and operation of the q^{th} row linear array. After (q-1)+n initial steps the q^{th} input property value bits are ciculated round the array, as shown at the bottom of Fig. 3. At time step (q-1)+n the actual processing starts, so that the q^{th} property values of n set elements are being processed simultaneously, for time steps (q-1)+n to (q-1)+2n-1. This is repeated for μ passes, where $\mu = \lceil m/n \rceil$, where each pass is depicted with a block in Fig. 3.

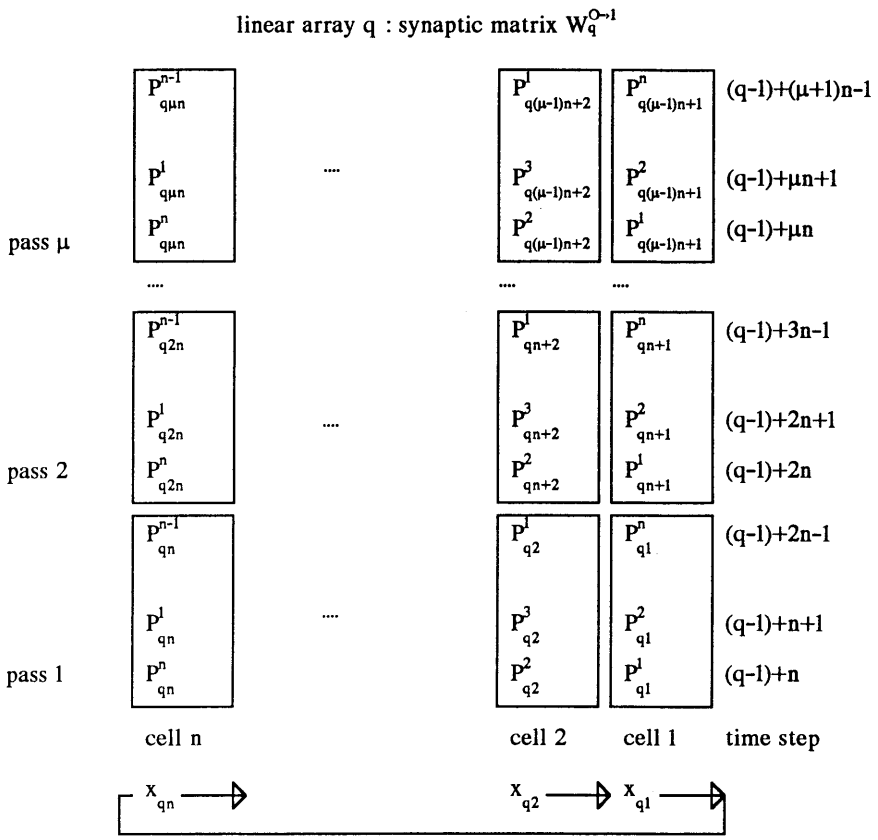

Figure 3. Synaptic matrix allocation and timing schedule

In Fig. 3 it is assumed, without loss of generality, that m is divided exactly with n; in the general case there would be μn-m unoccupied locations at the top block, which coresponds to pass μ. Therefore, a given synaptic matrix element $w_{qij}^{0\to1} = P_{qj}^{i}$, (1≤q≤k), (1≤i ≤n) and (1≤j≤m), is stored at the bit-level memory location with coordinates

linear array (row)	q
cell (column)	j MOD n
pass	j DIV n
memory location	(j DIV n)n + ((i-j+1) MOD n).

The timing schedule of the same synaptic matrix element can defined in a similar way and it is shown in Fig.3. The computation at the q^{th} row linear array is completed at step (q-1)+(μ+1)n-1, so that the computation of the k^{th} row is terminated at step (k-1)+(μ+1)n-1, or approximately k+m+n, and the final result is produced after 2n+k+m bit-level steps.

4.4. Training phase array operation

The training phase consists of essentially loading the elements of the synaptic matrix $W^{0\to1}$ into the appropriate memory locations. Notice that the other synaptic matrices are effectively fixed matrices which are mapped directly onto the neural network or the systolic array structure, so that no training phase is required. In our case

the training phase is reduced to simple encoding, i.e. accessing the appropriate memory location, using the memory allocation strategy described, in order to encode the corresponding bit of the property value in question. In order to load values to the memory locations of a systolic system two techniques are used, depending on the actual realisation of the systolic array. If the systolic array is realised in a massively parallel or distributed memory neurocomputing system then the memory access is performed through the memory management system. On the other hand a hardware oriented realisation implies the usage of the same communication links for both the recall and the training phase (Kung, 1993; Przytula & Prasanna, 1993). In the former case it is not possible to have exact time requirements for the training phase. In the latter case the training phase requires (m+1)n bit-level time steps for initial loading all property values.

5. MODIFICATION OF SETS AND PROPERTIES

The modification of a set element consists of the modification of some properties of the element., i.e. the modification of entry P_{qj} ofmatrix $\wp$ for given q_p ($1 \le q_p \le k$) and j_p ($1 \le j_p \le m$). This modification can be realised by means of the encoding method presented in section 2, that is $w^{0\to 1}_{q_p,i,j_p} = P^i_{q_p,j_p}$, $i = 1,2,..,n$. It is assumed that isolated access to interconnection weights is possible, for example through a separate memory management sysem. Otherwise the normal (recall) operation of the systolic array should terminate and a local or global training procedure should be followed. A local training procedure would re-encode only the modified value while a global training scheme would refresh the whole memory of the system.

5.1. Adding a new element

If a new element is added to Ω, then it will contain m+1 elements and in matrix $\wp$ of the values of the properties another column will be added, containing the values of the k properties for element ω_{m+1}. The interconnection weight between neuron i, ($1 \le i \le n$) at block q, ($1 \le q \le k$) in the input layer 0, and neuron m+1 at layer 1 will be the i^{th} digit of the value of property $P_{q,m+1}$: $w^{0\to 1}_{q,i,m+1} = P^i_{q,m+1}$, $i = 1,2,...,n$, while the rest of the connections are equal to zero. This corresponds to the 'modification' of all properties of a set element. The interconnection weight between neuron m+1 at layer 1 and neuron m+1 at layer 2 will be $w^{1\to 2}_{m+1,m+1} = 1$, while the rest of the weights are zero.

In terms of the systolic array implementation the addition of a new set element requires the encoding of all of its properties into the appropriate memory locations of the linear array rows by means of the storage allocation scheme described in the previous sections with j=m+1. However, if $m = \mu n$, then the addition of a new element implies the addition of a new pass, since $\lceil (m+1)/n \rceil = \mu+1$ and therefore $(m+1)$ MOD $n = 1$. Notice that there are no other changes required for accommodating the interconnection weights of higher layers. Thus, the area requirements remain the same, the memory requirements are increased by kn bit-level locations and the time requirements are increased by one step.

5.2. Adding a new generic property

If a new generic property (k+1) is added, it will contain m elements, and a new row should be added in matrix $\wp$ of the values of the properties. In the neural network of Fig. 1, we add another block (block k+1) of n neurons at layer 0, and a block of m neurons at layer 1, which should be fully connected with each neuron of block k+1 at layer 0. The interconnection weight between neuron i, ($1 \le i \le n$) at line k + 1, in the input

layer 0, and neuron j, ($1 \le j \le m$) at line k+1 of layer 1 will be the i^{th} digit of the actual value of property P_{k+1}: $w^{0\to1}_{k+1,i,j} = P^{i}_{k+1,j}$, i = 1,2,..,n, while the remaining connections are zero. The interconnection weights between neuron j of block k+1 at layer 1 and neuron j at layer 2 will be $w^{1\to2}_{k+1,j,j} = 1$ while the remaining connections are zero.

For the systolic array implementation the addition of a new property requires the insertion of a new linear array, that is a new row, between the k^{th} row and the bottom row linear array. The memory locations of this row will be encoded by means of the storage allocation scheme of section 4, with q=k+1. Thus, the area requirements are increased by kn bit-level cells, the memory is increased by kn^2 (or approximately km) locations and the time is increased by one step.

5.3. Deleting an element or a generic property

If element j_p ($1 \le j_p \le m$) of Ω is deleted then the set will contain m-1 elements and column j_p of matrix $\wp$ should be disabled. This corresponds to the disabling of the j_p^{th} neuron of each block of layer 1, as well as the j_p^{th} neuron of layer 2, that is $w^{0\to1}_{q,i,j_p} = 0$ and $w^{1\to2}_{q,j_p,j_p} = 0$ for q=1,2,..,k, i=1,2,...,n.

If a generic property q_p ($1 \le q_p \le k$) is deleted this means that row q_p of matrix $\wp$ should be disabled. This corresponds to the disabling of the q_p^{th} block of layer 0, as well as the interconnections of this block to layer 1, that is $w^{0\to1}_{q_p,i,j} = 0$ and $w^{1\to2}_{q_p,j,j} = 0$ j=1,2,..,m, i=1,2,...,n.

These modifications can be applied onto the systolic array as follows. The deletion of generic property q_p corresponds to the disabling of row q_p of the array. This disabling can be realised by modifying the cell computations so that they do not interfere with the calculations of the remaining rows. Thus the cells of row q_p simply transfer the incoming result towards the next row without any modification. Thus, in the cell definitions of Fig. 2 we should have

rout := rin for a disabled generic property.

In order to disable the j_p set element we should clear the memory locations that correspond to its property values. Using the storage allocation scheme of section 4 and for $j = j_p$, all appropriate memory locations can be put to 0.

6. CONCLUSIONS

This paper presents artificial neural networks and systolic architectures that detect multi-property relations satisfied by the elements of a set., that is elements having some common properties; satisfying at least one of some given properties; not satisfying all given properties; not satisfying any of some properties. The operations discussed are initially performed by means of a multilayer feedforward artificial neural network which encodes in its interconnection matrices the values of the properties of the set members. Then the systolic implementation of the neural network is addressed. Finally the ability of modifying, adding or deleting elements or properties is discussed.

Further research directions to be followed include the realisation of more complex combinations of set operations, the use of logical and other (e.g. don't care) operators, as well as the extension to fuzzy set operations. Possible applications of interest include structured data and knowledge bases, as well as free text searching and retrieval.

REFERENCES

Akl, S.G. (1989). *The design and analysis of parallel algorithms.* Prentice Hall.

Evans, D.J., Tsouros, K.K., Adamopoulos M. & Kortesis S. (1990). Neural network for searching sets of properties. *Parallel Computing,* v.16, pp. 279-286.

Herath, J. (ed.) (1992). Computer architectures for intelligent systems, *IEEE Computer Magazine Special Issue,* v.25, no.5.

Kohonen, T. (1988). *Self-organization and associative memory,* 2nd ed. Springer-Verlag.

Kung, S.Y. (1987). *VLSI processor arrays.* Prentice Hall.

Kung, S.Y. (1993). *Digital neurocomputing.* Prentice Hall.

Margaritis, K.G. & Evans D.J. (1992). Systolic implementation of neural networks. *Parallel Computing,* v.18, pp. 325-334.

Margaritis, K.G., Adamopoulos, M., Tsouros K.K. & Evans D.J. (1992). Systolic and optical implementation of neural network for searching sets of properties. *Proc. Conf. Parallel Computing '91.* Elsevier Science Publ,. pp. 229-240.

Przytula, K.W. & Prasanna, V.K. (eds.) (1993). *Parallel digital implementations of neural networks.* Prentice Hall.

Simpson, P.K. (1990). *Artificial neural systems.* Pergamon Press.

Wasserman, P.D. (1989). *Neural computing, theory and practice.* Van Nostrand Reinhold.