Multi-level Pattern Recognition in Wireless Sensor Networks for Structural Health Monitoring

Asad Khan, Zubair Baig, Matt Swift

Faculty of Information Technology
Monash University, Frankston 3199, Victoria, Australia,
e-mail: {Asad.Khan, Zubair.Baig, Matt.Swift}@infotech.monash.edu.au

Abstract Wireless Sensor Networks (WSNs) are emerging as a significant platform for the deployment of applications to monitor the health of structures. We propose the design of a multi-level mechanism for recognition of patterns in WSNs, depicting the health of structures. Our scheme reduces the overhead incurred owing to large-scale data transfers between the sensors and the base station by in-network pattern analysis, performed in real-time within the WSNs. The second layer of the pattern recognition process is done at the base station of the sensor network using adaptive mesh refinement and local data interpolation over triangulated surfaces. In-network processing provides for the fusion of data to minimise network traffic, and to conserve the energy resources of resource-constrained sensors. The fusion of sensory data such as stress fields of a bridge, or a logistical entity, at the base station, provides the means to form holistic patterns over domains which would otherwise be impossible to observe from an individual sensor perspective.

Key words Pattern recognition – associative memory – parallel processing – wireless sensor networks – finite element method

1 Introduction

Wireless Sensor Networks (WSNs) are characterised as a collection of tiny devices called sensor nodes placed in harsh and inaccessible environments
for sensing and reporting purposes. Sensor nodes are inexpensive to manufacture, and are therefore deployable on a large scale. They have limited on-chip energy resources available for all their sensing, processing, storage, and communication requirements [1][3][17]. Recent advances in microelectromechanical (MEMS) technology, particularly, ones which make use of wireless communications, open up new possibilities in the use of these devices in a coordinated manner. Sensor nodes may be programmed to form self-organising networks to create a tangible level of computation ability within the WSN setup. They have the capability to detect and measure a range of environmental variables e.g. temperature, motion, barometric pressure, and various forms of force fields. Sensory data may be fused to form a compact response for delivery to the remote monitoring system, which may be a base station with significantly more capabilities for large-scale computation and communication operations.

WSNs provide a very significant platform for monitoring the health of structures. Such structures may include buildings, offshore oil drilling facilities, and aircrafts. Sensor nodes report crucial data that may have an impact on the structural integrity. The manual observation of hard to access areas on the structural surface is not only very expensive, but at times also infeasible. For instance a large proportion of costs associated with aircraft corrosion prevention and control are incurred during the manual inspections of hard to access areas [18]. Sensor nodes deployed within or on such structures are responsible for the monitoring and reporting of critical structural status details. Such an approach ascertains diagnosis or prognosis of the structural health. It is therefore crucial to ensure that all operating sensor nodes are active, and have enough energy reserves for their respective lifetimes, since energy-depleted nodes within a structure are generally inaccessible for replacement purposes. It is therefore imperative to maximise the lifetimes of sensor nodes within the structure by reducing the overhead on these devices. The energy consumption associated with data communication operations by a sensor node is several orders of magnitude more expensive as compared to data storage and processing operations [10]. In order to increase the lifetimes of sensor nodes, it is therefore imperative to reduce the overall number of messages communicated within the network.

In [24], a multi-level decision fusion approach for distributed sensor networks using several data classification techniques has been proposed. The scheme combines the perceptions of local sensors with the damage scenarios in a knowledge database for generating an index defining the confidence of damage occurrence. Several levels of data fusion increase the complexity of the entire system, and add to the overhead associated with increased number of messages communicated between the sensor and actuator nodes operating at various layers of the scheme.

In this paper, we use the Graph Neuron (GN) [11][12][14][13], as a pattern recognition application for light-weight devices, to memorise and subsequently recall distinct patterns of structural behaviour imbedded within sensor nodes. The GN application overlay on the sensor nodes does comparisons
of sensed data with locally stored sub-patterns. Subsequently the application initiates communication with peer sensor nodes in order to generate an output decision depicting either a recall of a stored pattern, or memorisation of a previously unobserved pattern. The GN application therefore provides an in-network pattern recognition approach towards structural health monitoring by reducing the overhead of communicating multiple messages between various levels of a hierarchy, typical of a data fusion-based scheme.

This paper proposes a multi-level pattern recognition approach for the efficient monitoring of structural health. A major advantage of our approach is that the critical states are detected before they actually become critical for the health of the structure. Initial data analysis (i.e. pattern recognition) is carried out in real-time by the GN application, which operates on the sensor nodes. A second-level of data analysis is done at the base station, using adaptive mesh refinement and local data interpolation over triangulated surfaces techniques. This level of filtering requires the visualisation of sensory data in the form of holistic patterns, which may otherwise be unobservable from the perspective of a single sensor node. The base station, being central to all network operations, provides for visualization of holistic patterns. It also has more processing capabilities as compared to a typical sensor node, for such resource-demanding operations. The base station does not store each perceivable critical state of a structure, as the generation and storage of a large database with all possible critical states is computationally infeasible, and very time consuming. In addition, the computational overhead associated with analysis and comparisons of sensor readings with large databases is generally infeasible on light weight devices. However, our scheme does provide an efficient mechanism to reduce the overall volume of data generated by the sensor nodes, for communication to the base station, with the means of the GN application overlay (i.e. real-time responses). The proposed approach is particularly suitable for scenarios such as off-shore oil platforms or bridges, where long term structural damage is expected over a significant period of time. In such cases, the GN application ensures that the overall lifetimes of sensor nodes are extended as a result of efficient energy usage.

2 Wireless Sensor Networks (WSN)

In contrast with the conventional computer architecture, where computing and networking resource are hardly a consideration in the software design, the very limited architecture of tiny devices requires a minimalist approach to be adopted for such designs [4].

2.1 WSN Characteristics

A Wireless Sensor Network (WSN), as seen from Figure 1, is defined as a collection of battery-powered tiny devices called sensor nodes reporting
their sensory readings to a centralised device with several orders of magnitude more capabilities, called the base station [1]. WSNs are deployed for several health monitoring applications such as building structure and logistical entity monitoring. Sensor nodes have an on-board processor, wireless communication capability, sensing module, and memory. Table I enlists the specifications of the Berkeley Mica Mote sensor node [16].

Table 1 Berkeley Mica Mote Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8-bit 4 MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>128 KB Flash and 4KB RAM</td>
</tr>
<tr>
<td>Communication</td>
<td>916 MHz 40 Kbps Radio</td>
</tr>
<tr>
<td>Power</td>
<td>2 AA Batteries</td>
</tr>
</tbody>
</table>

Due to the light-weight nature of sensor nodes, as seen from Table I, any algorithm that may entice heavy usage of computation, communication or storage resources of a sensor node may lead to quick exhaustion of the limited battery power available to a sensor node. The GN as a light-weight pattern recognition application thus provides an efficient solution for in-network analysis of observed sensor data.

The limited energy and computational resources of sensors imply that data processing and transmission must be kept to a minimum in order to conserve energy reserves, and to not overload the sensor nodes. WSNs tend to use several data aggregation techniques to reduce the volume of traffic to be communicated to the base station. Data Aggregation is the process which allows vast amounts of data to be communicated in an efficient manner through the use of data centric routing protocols and effective middleware. Data is intelligently processed by aggregation nodes to reduce the volume of data generated, and for removal of redundant/repetitive data [8][9]. Pro-
Protocols such as SPIN, LEACH, PEGASIS, SMECN and others, provide data aggregation solutions for WSNs.

For the purpose of this paper we assume that the data aggregation and energy minimising routing schemes discussed above are already available within the WSN and thus do not influence the data aggregation and energy minimisation resulting from the pattern recognition scheme.

The sensor nodes operating within the network are programmed to host the Graph Neuron (GN) pattern recognition application. GN performs level-one of decision fusion by reducing the volume of data generated for transfer to the base station. This data volume reduction at the source-end reduces the communication bandwidth usage associated with large data transfers between the sensor nodes and the base station.

2.2 Pattern Recognition within a WSN

Adaptive finite element analysis [2][26], provide the means to accurately predict the behaviour of a range of electromechanical and structural systems, under the influence of the anticipated load conditions. These analysis are heavily relied upon in complex engineering designs. Wireless sensor networks could derive meaningful information by correlating the input patterns, gathered in-situ, with the patterns calculated by a finite element analysis using the latent associative memory of the network. Considering the associative memory must be implemented within a network with very limited computational resources, the governing algorithm must be modified to suit the limitations of the operating environment. This would generally entail replacement of complex sequential algorithms with parallel/decentralised ones. Also some drop in the accuracy of pattern recognition may occur, which could be offset by further processing at the remote system’s end (WSN base station). In this paper a very simple graph-based pattern recognition algorithm, hence forth referred to as the Graph Neuron (GN), is presented for use within the WSN. The proposed algorithm is not intended as an improvement to the highly sophisticated pattern recognition methods which are already available in the literature; focus instead is on reducing the computational requirements within resource-restricted networks, and providing responses in real-time.

The material stresses, computed using finite element analysis, were employed as the patterns for testing the proposed algorithm. Though the algorithm may be used for any other system capable of being modeled using a numerical, or an empirical method. A six element GN array was used to distinguish between the three significant stress states of a thin plate subjected to an in-plane load. The array design was later extended to include the detection of stress gradation within each of the monitored states.
2.3 A Simple Graph-based approach for Wireless Sensor Networks

The GN approach models the parallelism available within the naturally occurring Associative Memory (AM) systems, and by-passes the deficiencies present in some of the other contemporary approaches. It may also be noted that the GN algorithm is an inclusive technology; it may be extended to include spatiotemporal encoded neurons, and any of the evolutionary optimisation techniques such as genetic algorithms.

The GN is implemented as a self-organising [20] (ad hoc) virtual network of processing nodes. Each node executes the same copy of a very simple AM algorithm, and in doing so it provides a natural framework for supporting parallelism. The algorithm is best suited for immensely parallel systems such as wireless sensors based networks.

The overall topology of the array is shown in Figure 2.

The proposed architecture draws upon the model proposed by Stapp [22] where,

A grid of points in the brain is represented as a set of vectors \((x_i)\),

\[
(i \in [1, N]) \land (\forall x_i, \exists F_j(x_i) \mid j \in [1, M] \land M \approx 10)
\]

For all valid pairs \((i, j)\) at a temporal point \(t\),

\[
F^t_j(x_i) \in [-L, +L]
\]

where,

- \(F^t_j(x_i)\) is a set of Fields
- \(L\) is a large integer.

There is also a grid of temporal values \(t_n, n \in [1, T]\). The description of the classical system at any time \(t_n\) is therefore given by specifying for each pair of value \((i, j)\) with \(i \in [1, N]\) and \(j \in [1, M]\), some value of \(F^t_j(x_i) \in [-L, +L]\).

We would consequently need, in order to specify this classical system at one time \(t_n\), \(N \times M\) registers, each of which is able to hold an integer in the range \([-L, +L]\).

The GN data representation follows a very similar model to the one proposed by Stapp. In this regard, an implementation of the GN algorithm shows how information may be discretely stored within the network, by
simply manipulating the adjacency information held within the nodes of recursively connected arrays.

3 The Graph Neuron

Graph Neuron is a scalable associative memory network, capable of handling concurrent streams of inputs for processing and matching with stored historical data [11].

3.1 Overview

The Graph Neuron algorithm changes the emphasis from high speed sequential CPU processing, to parallel network centric processing; allowing the use of a very large number of parallel processors within recursively connected domains [11] of Peer-to-Peer networks [15][20][25]. The algorithm is thus capable of providing a near instantaneous response time to the inputs, using relatively low-performance processors.

The proposed technique is expected to work rapidly through very large sets of multi-dimensional sensory inputs, and to provide results with a reasonable degree of accuracy. This is possible owing to a very wide range of input information e.g. comprising 1, 2, 3, and 4 dimensional data points, all present within the same signaling element, being concurrently presented and processed by the network without any loss of efficiency. The processing requirements in this regard may be progressively reduced, by simply increasing the number of nodes within the network, to the extent where the algorithm may be implemented within the hardware of the sensors (acting as the nodes within the WSN).

3.2 GN Data Representation

The information presented to a GN is in the form of (value, position) pairs; representing a data point in a two dimensional space. For multi-dimensional patterns the number of values per position would increase in order to represent the additional information - the underlying principle would however remain the same.

The GN array converts the spatial/temporal patterns into a graph-like representation, and then compares the edges of the graphs for memorisation and recall operations. The advantage of having a graph-like representation is that it provides a mechanism for placing the spatial/temporal information in a context. Therefore, not only can we compare the individual data points but we may also compare the order in which these occur. The drawback to this approach is in the excessive number of comparisons required for matching a stored pattern with an incoming sequence. The search domain will increase in correlation with the number of stored patterns. For
an \( n \) vertices graph, the maximum computational complexity of the matching process is of the order of \( O(2^n) \) [23] [6]. However, this impediment only exists because of the nature of the contemporary computer architecture; which converts purely parallel operations into a sequential form and then emulates these operations in a pseudo-parallel mode using elaborate scheduling algorithms. On the other hand, the proposed algorithm utilises the real parallelism present within the network.

The inter-processor message-passing is implemented using the communicating sequential process (CSP) model put forth by Hoare [7].

The data representation for a GN may be summarised as follows: An input pattern vector \( P{} \) is represented as a set of \( p(value, position) \) pairs. These inputs are mapped on to a virtual array of processors by using the adjacency characteristic of the input. For example, alphabets and numbers would have their inherent adjacency characteristics. Similarly images would have the frequency bands, intensity, and spatial coordinates, as the adjacency characteristics per pixel.

For an input domain \( R \), the GN array represents all possible combinations of \( P{} \in R \). Therefore, each GN node is initialised with a distinct pair \( p \) from the input domain \( R \).

Each GN keeps a record of the number of times it encounters a matching input pair; within its bias vector. Each element of the bias\( \{ \} \) comprises a list of the adjacent GNs relating to a matched input pair. The bias\( \{ \} \) counter is incremented for each new pair matched by the GN. A new pair is defined as the one which has a different set of adjacent GNs to the existing elements of the bias\( \{ \} \).

In order for this method to work successfully, we need to have a priori knowledge regarding the size of the input data domain, or alternatively we may choose our own limits and define the reality within those bounds. For instance, by defining an input domain which comprises all the characters in a natural language, and the number of characters in the longest word occurring in that language, it would be sufficient to represent any word from the language. Alternatively we could set our own limits for discretisation of continuous input domains, for this purpose.

3.3 The Parallelism within the Representation

A Graph Neuron (GN) array may be created where each GN is initialised to a (value, position) pair \( p \) for every possible position and value within the input domain. The incoming data pairs simply get mapped to their appropriate locations within the array. For instance, a four lettered word with a choice of two alphabets, say \( X \) and \( O \) for each of the four positions, would require eight GNs for representing every conceivable combination. It’s easy to show that the total number of possible combinations in this case would be \( 2^4 = 16 \). This effectively means that we are assigning a separate search domain for each set of the possible values of the alphabets, and
Fig. 3 An eight node GN array is in the process of storing patterns P1 (RED), P2 (BLUE), P3 (BLACK), and P4 (GREEN). (Note: The colouring scheme for interconnects is separate from the scheme used for the patterns.)

therefore halving the search domain in this case. For example, if we get a letter X in the first position of the word, then letter O can never occur at this position for this particular word and vice versa. The halved (adjacency) search domains are processed concurrently, thus the total time for the search is that for one half of the domain.

The number of elements in the bias\{\} increase with the number of patterns being presented to the array. However, the number of bias\{\} elements does not increase in proportion to the number of stored patterns, since pairs with the same set of adjacent GNs are treated as recalls (and thus do not get stored). The store operation requires an increment in the bias\{\} index counter.

The process of searching through the bias\{\} entries within each GN takes place concurrently. This map and search process is broadly illustrated in Figure 3. The Figure outlines the process of storing patterns (P1, P2, P3, and P4) on an array comprising 8 GNs (labeled as N1, N2, ..., N8). Each pattern is comprised of 4 pairs, where the values may alternate between X and O for each of the four positions.

Assuming P1 is mapped first in this instance, each GN would record the responses from the other GNs to form its port sequence list of the adjacent GNs, and would allocate an entry within their respective bias\{\} arrays for these pairs (the GNs are adjacent if their position differs by 1 in this example). When N1 encounters p(X, 1), it will store the port number,
6, in its $bias\{}$ for $N_6$. $N_6$ will store the port numbers 1 and 3, for $N_1$ and $N_3$, in its $bias\{}$ after encountering $p(O, 2)$. The process gets repeated for the encounters with the remaining pairs in the pattern i.e. $p(X, 3)$, and $p(X, 4)$. The entire process is repeated each time for storing $P_2$, $P_3$, and $P_4$. The $bias\{}$ entries for each of the GN are shown in Figure 3. The following section summarizes the GN algorithm.

### 3.4 The Graph Neuron Algorithm

All operating GN instances have exactly the same logic and code. These are implemented as copies of a self contained message-passing application. Each instance of the application is initialised to a distinct $p(val, pos)$ and port values. Hence the GN array keeps all possible values and all possible positions, for a particular data domain $R$, mapped as unique $p(val, pos)$ pairs on each GN.

The patterns are presented as sets of $p(val, pos)$ pairs to the array. Adjacencies are calculated independently by each GN within the array as part of the store/recall operations. A GN on receipt of a $p(val, pos)$ pair, checks with all other GNs for adjacent values and notes the port sequence for that particular pair. The GN then compares the previously stored port sequences within the $bias\{}$ and returns a high bias if a match is found, otherwise the sequence is added as a new element to the $bias\{}$ (partial matches may result in low confidence bias matches, however this function has not been implemented yet).

Only a single value may be found at a particular position within the array. Knowing the adjacent GN’s number, is sufficient to determine the pair it has been programmed to respond to.

### 3.4.1 The Input Operation

Incoming stimuli (the whole pattern/sequence) should be sensed by every GN (akin to an Ethernet broadcast on a Local Area Network). Only the GNs with matching values should initiate action. Doing this would require interfacing the array to the spiking neurons or a form of sensory mechanism e.g. a WSN. Alternatively, the use of a data link layer level multicasting protocol may be considered.

### 3.4.2 Pattern Store and Recall Operations

Assuming that such an input mechanism is in place, each GN listens on the port that matches its own unique identity number to store or recall. There is no order as to how a pattern gets distributed amongst the GNs. The commit to memory operation is done on 1st-come-1st serve basis. Each GN communicates with the other GNs to identify its adjacent GNs (closest neighbours). The commit to memory operation is only performed if there is no recall within the GN. Hence for each input pair, a GN checks with its neighbours, to decide whether to treat the incoming pair as a store or as a recall operation.
3.4.3 The GN PDU An input to the array is in the form of Protocol Data Units (PDUs) which comprise the pattern. Each pattern in turn comprises a set of value and position pairs. The structure of a PDU is shown in Figure 4, where pos could be a timing relationship, or it could be a vector in its own right comprising contextual values associated with each val.

In the current implementation, the vals and pos pair determine the contact port and the direction of search; using the adjacency characteristics of a two-dimensional array. It is important to note that the data type is only for human consumption. As far as the array is concerned, the data type has no bearing on its store and recall operations. The array only deals with the internal representations, associated with the inputs, in terms of its connectivity with the other nodes within the array. The connectivity information is kept within the bias{} vector.

4 Application of GN on a Wireless Sensor Network

The existing implementation of the GN is based on the standard internetworking protocols i.e. TCP/IP which allow interoperability with various types of WSNs. The amount of memory and non-volatile storage space requirements are nominal for the software implementation. However these need to be further reduced to meet the on-board storage and processing capacities of the WSN nodes. In this regard the number of patterns stored within the network must be limited to reduce the number of entries within each of the bias{} vectors. Keeping the bias entries to a minimum will also reduce the processing load per GN.

The increase in the number of entries primarily depends upon the data set being used for storing a pattern. In this regard a pattern comprising a set of four alphabet characters (A, B, C, D), would entail a total number of $4^4 = 256$ possible combinations. A GN array of 4x4 = 16 nodes would
be required to cater for all possible patterns. Figure 5 shows the maximum bias entries which would occur after an arbitrary set of string patterns comprising $P_1 : ABBD, P_2 : ACCB, P_3 : BACA,$ and $P_4 : ABCD,$ has been exposed for the first time to the array. It is also important to point out that a GN array exposed to its full capacity, would return a positive response to any pattern (from the pattern domain); rendering the purpose of pattern recognition meaningless. Hence having fewer patterns to discern amongst provides a valid use of the array. Alternatively the pattern recognition domain of the array must always be greater than the pattern domain itself. The condition of having far fewer patterns as compared to the total capacity of the array dovetails into the requirement of storing fewer patterns due to the hardware limitations of the WSN.

The GN algorithm has the inherent ability to form an ad hoc network. This is done by connecting to the appropriate GN ports when the array is simulated within a single computer (i.e. the host address remains the same for all GN processes). However the node addresses would all be different within a WSN. Hence the process of forming an ad hoc network would need to be addressed. The formation of an ad hoc network in order for the GNs to communicate with their nearest neighbours, as defined under the GN algorithm’s data adjacency considerations, may be done in two different ways: either the node (host) addresses could be appended with the port number entries with each of the bias vectors, or a more elaborate scheme needs to be implemented. In this regard, a very basic distributed hash table (DHT) [19] based ad hoc networking scheme would be a feasible alternative.

4.1 In-network Pattern Pre-selection with GN

An arbitrary L-shaped plate with in-plane loading is investigated for in-network pattern pre-selection with the GN algorithm. This process forms
the first level within the multi-level decision fusion approach being presented in this paper. It is assumed that each of these plates would be embedded with a WSN as shown in Figure 6.

Complex shapes can thus be formed using these simple L-shaped plates. The embedded WSN can measure strain, stress, displacement, or any other parameter of importance in the design of this continuum. These parameters may be assumed as vectors orthogonal to the plane of the WSN. The in-plane stresses have been selected as the orthogonal vector for this study. Two stress states, out of the six possible states under the horizontal and vertical load conditions, were arbitrarily selected to demonstrate the in-network pattern recognition capability of the application. It is assumed that these two stress patterns are highly detrimental to the continuum, and need to be watched for their occurrence, to be detected in real time. These patterns may also result for non-critical stress states, however the final determination of the pattern detected by the WSN would be done outside the network, where greater computational resources can be made available for interpolating stress readings obtained from a relatively coarse grained WSN.

4.2 Identification of Stress States by a Simple GN Array

A relatively simple finite element discretisation was used to program a GN based network in order to demonstrate that the stress distributions, within a structural element, may be successfully identified. In this regard the three main stress distributions, $\sigma_x$, $\sigma_y$, and $\sigma_{xy}$, were induced by a single in-plane load acting in the horizontal direction on a thin L-shaped plate. The process was repeated with the same load, but this time acting in the vertical direction. The stress distributions computed using this finite element model are shown in Figures 8 and 9, respectively. The stress patterns $\sigma_z$, the longitudinal stress pattern for the load acting in the horizontal direction,
Fig. 7 A text string comprising characters ‘X’ and ‘O’ being mapped to the appropriate nodes within the array using the input pairs \( p_1(X, 1), p_2(O, 2), p_3(X, 3) \).

Fig. 8 \( \sigma_x, \sigma_y, \) and \( \sigma_{xy} \) for the horizontal load condition

and \( \sigma_{xy} \), the shear stress pattern for the load acting in the vertical direction, were selected at random to test the pattern recognition application.

It should be pointed out that the stress states, shown in the Figures 8 and 9, were produced entirely by the finite element analysis, and therefore the GN had no influence over the quality or the make-up of the input patterns. In order to test the algorithm, the array was exposed to the \( \sigma_x \) stress
Upon all the stress states being presented to the array, the array was expected to respond unequivocally to the patterns it had been exposed to previously. Other patterns were expected to invoke only partial recall responses. It was assumed that a finite element would either be in a state of positive stress state (tension) or it would be in a negative stress state (compression). Hence a binary set of characters, X and O, was sufficient to define the stress values within the patterns in this case. The position for each stress value was denoted by the corresponding mesh element in the discretisation.

Since the mesh comprised of six elements; a network of $2 \times 6 = 12$ GNs was set up to cater for each element in the finite element mesh. By setting the number of GNs as an integer multiple of the number of elements in the finite element mesh, it was possible to define an unequivocal response from the array. Exactly six GNs had to agree before a response could be accepted in this case. By maintaining this relationship it was also possible to differentiate between the six stress states. The network input patterns corresponding to these stress states, are shown in Table 2. Stress patterns $P_1$ and $P_6$, were first introduced to the array for storage within the GN array. Later all six patterns were sequentially presented to the array. The array storage and recall responses are shown in Figure 10. It may be seen from the Figure that $P_1$ and $P_6$, marked with the asterisk, set six of the GNs within
Table 2 The stress state patterns corresponding to the horizontal and the vertical load conditions

<table>
<thead>
<tr>
<th>Stress Pattern (P)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 = \sigma_x ) 100H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>( P_2 = \sigma_y ) 100H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>( P_3 = \sigma_{xy} ) 100H</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( P_4 = \sigma_x ) 100V</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( P_5 = \sigma_y ) 100V</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( P_6 = \sigma_{xy} ) 100V</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
</tbody>
</table>

Fig. 10 The pattern storage and the pattern recall outputs relating to the six stress states.

the array to the recall status. It may also be noted that no memorisation activity occurs within the array, when these patterns are presented again. Other patterns evoke mixed responses from the network. Figure 10 shows that stress states may be detected by employing a fully distributed pattern recognition algorithm within a WSN. It may however be noted that at this stage the responses from the network are limited to detecting regions of positive and negative stresses alone.

5 Second Level Data Fusion with Adaptive Mesh Refinement

The patterns picked up by the WSN through in-network processing only represent binary level variations within the patterns. It is quite possible the pattern pre-selected at Level 1 may not be the critical pattern. The other problem with these patterns is that they represent a very coarse sample comprising relatively few readings. Therefore there is a need for comparison of WSN node readings (within each pre-selected sample) with the critical patterns. Adaptive mesh refinement provides a well adjusted finite element
mesh, which can be used for interpolating the values at a finer scale. The interpolated values could then be readily compared with the critical pattern values. This would ascertain whether the critical pattern was indeed encountered within one or more WSNs.

5.1 Adaptive Finite Element Analysis

The state of the mechanical, structural, and electrical components may be effectively modeled using this well established numerical technique. For this study, a stress-strain based finite element model was selected to distinguish between the various stress states. To achieve a finite element solution close to the actual response of the material, it is assumed that if the continuum is idealised using a large number of elements, such that the size of each individual element is very small in comparison to the dimensions of the continuum, then the actual element stresses will tend to be constant over each of the elements, and the solution may be regarded as accurate for all practical purposes. However doing so would require the use of a very fine mesh. In order to avoid the computational cost in this regard, an adaptive refinement of the mesh is carried out. The process of adaptive mesh refinement \cite{2}\cite{26}\cite{21}, is briefly outlined in the following sections for providing the readers with some background information, relating to the method used for the pattern generation.

5.1.1 Smoothed stresses from nodal averaging

In the process of nodal averaging, the element stresses $\sigma$ from all the elements common to the node are averaged and this is repeated for all the nodes of the mesh. The vector $\sigma$ consists of the element stresses $\{\sigma_{xx}, \sigma_{yy}, \sigma_{xy}\}^T$. Then for each element a smoothed value, $\hat{\sigma}$, is determined by averaging the three nodal stress values. This can be expressed as

$$\sigma_n = \sum_{nn} \sum_{ns} \frac{\sigma}{|\sigma|}$$

$$\hat{\sigma} = \sum_{ne} \frac{\sigma_n^3}{3}$$

where $nn$ is the number of nodes in the mesh, $ns$ the number of elements connecting to a node and $ne$ the number of elements in the mesh.

5.1.2 Equations of adaptive remeshing $h$-refinement

The standard finite element problem may be summarised by the following equation:

$$Ku - f = 0$$

where
\[ K = \int_{\Omega} B^T DB \]  \hspace{1cm} (6)

\( f \) = applied load vector.
\( u \) = nodal displacement vector.

In adaptive calculations the displacement error is:

\[ e = u - \hat{u} \]  \hspace{1cm} (7)

The error in the stresses:

\[ e_\sigma = \sigma - \hat{\sigma} \]  \hspace{1cm} (8)

where \( \sigma \) and \( \hat{\sigma} \) are the element stresses from finite element analysis and nodal averaging, respectively. The error in the strain may be determined from the error in the stresses:

\[ e_\varepsilon = D^{-1} e_\sigma \]  \hspace{1cm} (9)

The error energy norm is defined to be

\[ \| e \| = \left( \int_{\Omega} (Be)^T D (Be) d\Omega \right)^{\frac{1}{2}} \]  \hspace{1cm} (10)

Substituting equation (9) in equation (10) we obtain the following expression for the energy norm:

\[ \| e \| = \left( \int_{\Omega} e_\sigma^T D^{-1} e_\sigma d\Omega \right)^{\frac{1}{2}} \]  \hspace{1cm} (11)

The total energy norm is given as

\[ \| u \| = \left( \int_{\Omega} \hat{\sigma}^T D^{-1} \hat{\sigma} d\Omega \right)^{\frac{1}{2}} \]  \hspace{1cm} (12)

All these norms have been defined for the whole domain. In practice the norms for each individual element of the mesh are calculated and summed:

\[ \| e \| = \sum_{i=1}^{ne} \| e \|_i^2 \]  \hspace{1cm} (13)

An adaptivity control parameter \( \eta \) is defined to quantify percentage error

\[ \eta = \frac{\| e \|}{\| u \|} \times 100 \]  \hspace{1cm} (14)

If \( \eta \) is the limit on error then, while

\[ \eta > \eta_0 \]  \hspace{1cm} (15)
the mesh element refinement parameter is defined as

$$\xi_i = \frac{\|e\|_i}{e_m}$$  \hspace{1cm} (16)

where

$$e_m = \eta \left( \frac{\|u\|^2}{ne} \right)^{\frac{1}{2}}$$  \hspace{1cm} (17)

The mesh is refined according to

$$h_{\text{new}} = \frac{h_i}{\xi_i}$$  \hspace{1cm} (18)

until

$$\eta \leq \bar{\eta}$$  \hspace{1cm} (19)

where $h_i$ and $h_{\text{new}}$ are the previous, and the new element sizes, determined from the adaptive analysis. By iteratively refining the mesh, as describe above, the true material response to an external force field may be quite accurately estimated. Adaptive finite element analysis is extensively used to predict the behaviour of large building structures, the modeling of fluid flows, and the estimation of electrical, magnetic, and thermal fields. Applying the adaptive mesh refinement to the L-shaped plate provides a precise estimate of one of the critical stress patterns defined earlier.
Figure 11 provides a detailed distribution of stresses over the plate. However, the values obtained from the WSN, would in theory, only provide the readings at the centre of each of the six WSN nodes embedded in the plate. The values at other points within the WSN, would need to be estimated values using some form of an interpolation scheme which can refine the values measured by the sparse set of WSN nodes to a continuum.

5.2 Inverse distance weighted least squares interpolation

We propose Franke and Nielson’s Method II interpolants [5], for mapping coarse 2-dimensional meshes to 3-dimensional surfaces. A set of orthogonal vectors defined at the surface of a coarse background mesh, can thus be used in conjunction with an adaptively refined mesh, to determine a realistic surface through this interpolation.

\[
W_i(x, y) = b_i^2(3 - 2b_i) + 3b_i^2b_jb_k
\left\{ b_j \left( \|e_i\|^2 + \|e_k\|^2 - \|e_j\|^2 \right) + b_k \left( \|e_i\|^2 + \|e_j\|^2 - \|e_k\|^2 \right) \right\}
\]

where \(b_i, b_j\) and \(b_k\) are the barycentric (area) co-ordinates of the point \((x, y)\) with respect to the triangle \(T_{ijk}\) and \(\|e_n\|, n = i, j, k\) represents the length of the edge opposite \(V_n, n = i, j, k\). The final interpolant is given by

\[
G[f](x, y) = W_i(x, y)Q_i(x, y) + W_j(x, y)Q_j(x, y) + W_k(x, y)Q_k(x, y)
\]

The function \(Q\) is defined as

\[
Q_k(x, y) = z_k + \pi_{k2}(x - x_k) + \pi_{k3}(y - y_k)
\]

\[
+ \pi_{k4}(x - x_k)^2 + \pi_{k5}(x - x_k)(y - y_k) + \pi_{k6}(y - y_k)^2
\]

and the solution of the following minimisation problem is required:

\[
\min_{a_{k2}, \ldots, a_{k6}} \sum_{i=1 \atop i \neq k}^{N} \left( \frac{z_k + a_{k2}(x_i - x_k) + \ldots + a_{k6}(y_i - y_k)^2 - z_i}{\rho_i(x_k, y_k)} \right)^2
\]

where

\[
\frac{1}{\rho_i} = \frac{(R_q - d_i)_{+}}{R_q d_i}
\]

\(d_i\) is the distance from \(i\)th node of the coarse mesh to the point represented by \((x_k, y_k)\). The radius of influence \(R_q\) determines the accuracy and efficiency of the method and is taken as
Fig. 12 Example: Initial coarse mesh comprising 21 nodes and 28 elements

\[ R_q = \frac{D}{2} \sqrt{\frac{N_q}{N}} \]  \hspace{1cm} (25)

where \( D = \max_{i,j} d_i(x_i, y_i) \).

The initial value of \( N_q \) was taken as 0.55\( D \). However this may have to be increased two to three times depending on the density and the gradation of the mesh in order to obtain correct interpolation of the three dimensional surface.

**Implementation**

The initial coarse mesh shown in Figure 12, comprising 21 nodes and 28 elements, was restrained at the four corners, and then a distributed out-of-plane load was applied to the central portion of the mesh. The membrane was analysed using a dynamic relaxation based non-linear finite element analysis scheme. The displaced shape after the analysis is shown in Figure 13.
Adaptive finite element calculations were performed using the stress resultants obtained from the finite element analysis. The domain was re-meshed into two dimensional form, using the mesh parameters computed within the adaptivity module. The re-meshed two dimensional domain is shown in Figure 14.

Using the Franke and Nielson’s inverse distance weighted least square interpolation, the two dimensional re-meshed domain was mapped over the three dimensional geometry represented in Figure 13. The adaptively generated three dimensional surface is shown in Figure 15.

6 Discussion

The initial mapping of the pairs, $p$, to the GNs requires that each GN must contact every other GN, at the appropriate locations, within the array. The process requires a high degree of connectivity between the GNs, or the availability of adequate bandwidth if shared connections are being used. The communications among the GNs, with adjacent pairs, take place in parallel. The communication in the previous step would also occur in parallel, if
Fig. 16 A 3-dimensional GN array for sensing a range $R_1, \ldots, R_n$, where $n = 3$, for a 2-dimensional finite element domain with a stress gradation.

A spiking neuron interface, or a multicast protocol, was used to input the information.

Each new pattern sequence results in an increase in storage within some of the GNs. Thus the search domain does not increase proportionately whilst the total storage keeps increasing until it’s exhausted for the particular topology. The bias storage algorithm at present compares a GN’s adjacent sub-patterns before entering these in its bias array. This substantially reduces the storage requirement, as previously encountered sub-patterns do not get stored again. However a loss of information occurs owing to the sub-patterns not being associated with their respective (stored) patterns. The risk of two slightly dissimilar patterns being viewed as the same by the array increases in the presence of a large number of matching sub-patterns within the pattern domain. This may also be seen in the process, where similar patterns tend to interfere with each other at the time of a recall. The level of interference between the stored patterns is generally proportional to the number of stored patterns. Hence the interference is expected to increase with an increase in the number of patterns stored for reference within the pattern domain. A possible way of reducing this problem would be to limit the sub-pattern matching to the individual patterns; whereby a separate bias entry would be required for each sub-pattern found in a different stored pattern. However, such an approach would lead to a greater storage and processing requirement. Hence the current scheme, though more prone to interference and having lesser capacity for storing reference patterns, would be better suited for WSNs comprised of nodes with very limited processing and storage capacities.

The current application of the GN for stress state detection may be applied to virtually any sensor environment, where it’s important to detect and possibly avert certain internal states. For instance, a space structure may be pre-programmed to detect certain conditions, which must be avoided during its operational life. Similarly a complex plant, or a mechanical equip-
ment, may be monitored at the component level with respect to the critical states. Each of the fire bricks of a heat shield could in theory be monitoring its state against the thermal and structural stress distributions. Doing so would require that the nodes within the WSN are fine grained enough to be embedded in the actual component at the manufacturing stage.

7 Conclusions

The ability to discern between various stress states of a material object, using a basic pattern recognition network, has been demonstrated in this paper, to prove that an inanimate object may be enabled to detect significant changes to its internal state, in a manner similar to our sense of touch. The pattern recognition operations are carried out at two stages of a hierarchy. The GN application performs the initial pattern recognition operation within the sensor network. The more sophisticated adaptive mesh refinement, and local data interpolation over triangulated surface operations, are performed by the more powerful base station. The GN application does in-network fusion of decisions, to generate minimalistic responses for communication to the base station. The results show that a basic pattern recognition capability may be easily introduced within a network, using a completely distributed software architecture. The distributed nature of the algorithm allows a natural form of parallel processing to occur, which offsets the constraint limiting the use of a graph-based approach.

The investigation shows that the proposed approach has the potential to be developed into a novel application, where material objects could sense their internal states autonomously using the limited computational resources available within their embedded sensor networks, and possibly take remedial actions.

References

5. R. Franke and G. Nielson. Smooth interpolation of large sets of scattered
dust. In *Proceedings of the ACM/IEEE Intl. Conf. on Mobile Computing and
and aggregation in wireless sensor networks. In *Proceedings of the IEEE
International Conference on Networking (ICON’02)*, Atlanta, Georgia, August
10. Hui Kang and Xiaolin Li. Power-aware sensor selection in wireless sensor net-
works. In *In Proceedings of the 5th International Conference on Information
Processing in Sensor Networks (IPSN’06)*, 2006.
11. A. I. Khan. A peer-to-peer associative memory network for intelligent infor-
mation systems. In *Proceedings of the Thirteenth Australasian Conference on
Information Systems*, volume 1, 2002.
12. A. I. Khan, M. Isreb, and R. S. Spindler. A parallel distributed application for
the wireless sensor network. In *Proceedings of the 7th International Conference
on High Performance Computing and Grid in Asia Pacific Region*, 20-22 July,
Tokyo, Japan, 2004.
13. A. I. Khan, M. Isreb, and R. S. Spindler. A parallel distributed applica-
tion of the wireless sensor network. In *Proc. of the 7th Int’l Conf. on High
14. A. I. Khan and P. Mihailescu. Parallel pattern recognition computations
within a wireless sensor network. In *Proceedings of the 17th International
15. A. I. Khan and R. S. Spindler. A blueprint for building serverless applications
16. Philip Levis and David Culler. Mate: A tiny virtual machine for sensor net-
works. In *In Proceedings of the 10th International Conference on Architec-
tural Support for Programming Languages and Operating Systems (ASPLOS
X)*, 2002.
17. S. Lindsey and C.S. Raghavendra. Pegasis: Power-efficient gathering in sensor
information systems. In *Aerospace Conference Proceedings*, volume 3, pages
D. Price. Establishing a physical basis for the in-situ monitoring of airframe
Tri-Services Conference (TSCC05)*, 2005.
19. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location
20. S. Senaratna and A. I. Khan. Autonomous clustering algorithms for serverless
peer-to-peer (p2p) systems. In *Proceedings of the 6th International Conference
on High Performance Computing in Asia-Pacific Region*, Bangalore, India,
volume 2, 2002.


