Middle-East Journal of Scientific Research 20 (5): 617-629, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.20.05.11574

A New Simulation of Distributed Mutual Exclusion on Neural Networks

Peyman Bayat

Mathematics and Informatics Research Group, ACECR,
Tarbiat Modares University, Tehran, Iran

Abstract: In a distributed system, process synchronization is an important agenda. One of the major duties for process synchronization is mutual exclusion. In new algorithm, opposite the past algorithms fairness happens. This paper presents a new approach of the race models involving distributed mutual exclusion. Further, concrete applications of these models did not involve variability in the accumulator size or were based on a specific distribution. We show that the distributions of time stamp, time action and the other effective parameters predicted by the neural network competitive models can be solved analytically this problem that be happens in the critical sections. The model can be manipulated and simulated to predict the effects of reward on Hamming and Hopfield's models curves and speed-accuracy decomposition. In other hand, the major contribution of this paper is the implementation of a learning rule that enables networks based on a race model to learn stimulus-response associations. The model described here can be seen as a reduction of information system and is compatible with a priority learning system. Also, we will consider the non-linear behavior of the competitive models and as a result use this property in distributed systems. Finally, it is possible to use the neural networks as a distributed system pattern, to optimization of fault tolerance, reliability and accessibility related to mutual exclusion and critical section. Thus in the new approach fault tolerance will ascend and centralize and distributed algorithms can use this and based algorithm will be more reliable.

Key words: Distributed Mutual Exclusion · Neural networks · Competition · Simulation

INTRODUCTION

THE mutual exclusion problem states that only a single process can be allowed access to a protected resource, also termed as a critical section (CS), at any time [1]. Mutual exclusion is a form of synchronization and is one of the most fundamental paradigms in computing systems. Mutual exclusion has been widely studied in distributed systems where processes communicate by asynchronous message passing and a comprehensive survey is given in [2, 3].

The aim of this article is to expand our knowledge of the race models by showing that there exist similarities between the competitive models in neural networks and happens that in distributed systems.

In the other hand, fairness is a very important criterion for solutions to most real-life resource contention problems. The commonly accepted definition of fairness in the context of mutual exclusion is that requests for access to the CS are satisfied in the order of their time stamps. Of all the distributed mutual exclusion algorithms in the literature, only the non-token-based algorithms of Lamport [4] and Ricart-Agrawala [5] (RA) are fair in the sense described above. Singhal's heuristic algorithm [6] guarantees some degree of fairness but is not fair in the sense described above. A lower priority request can execute CS before a higher priority request if the higher priority request is delayed. The algorithm has different criteria for fairness. It favors sites which have executed their CSs least frequently and discourages sites which have executed CSs heavily. This does not take into account the causality relation that exists between two requests and hence, does not conform to the sense of fairness described by Lamport's clock. Singhal's dynamic information structure algorithm [7] attempts to be fair, but does not satisfy the fairness criterion. The algorithm uses the concept of Lamport's clock and the causality relationship, but it also allows a low priority request to execute CS before a high priority request if the high priority request is on the way or delayed (process that has

Corresponding Author: Peyman Bayat, Mathematics and Informatics Research Group, ACECR,
Tarbiat Modares University, Tehran, Iran.

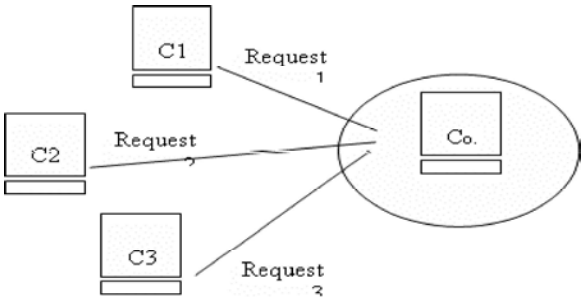

Fig. 1: Multi clients requesting critical section that coordinate with Coordinator.

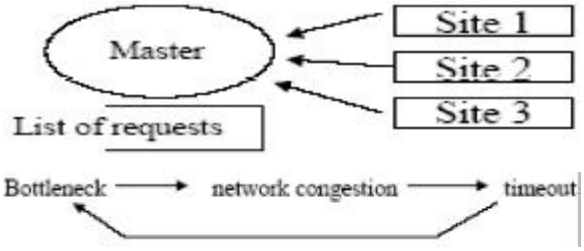

Fig. 2: Distributed Systems and Web Applications.

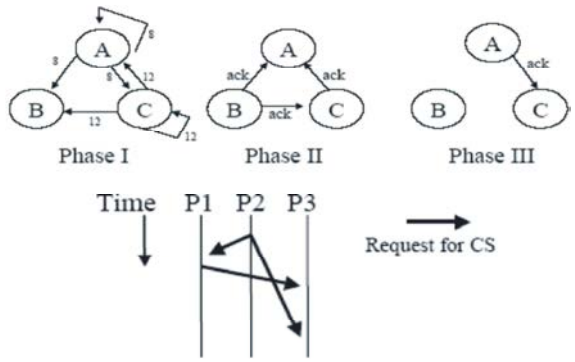

Fig. 3: Distributed System Algorithm.

made a higher priority request is not in the request set of the process that has made the low priority request). The proposed algorithm in this paper uses the fairness criteria given by Lamport and improves on RA, which is the best known algorithm that guarantees fairness in the same sense. Also, it will be different to the routine of past algorithms. Because in those algorithms, just time stamp, singly is the parameter that makes decision to entrance the processes to the critical sections.

Due to the absence of global time in a distributed system, timestamps are assigned to messages according to Lamport's clocks [4]. In the context of mutual exclusion, Lamport's clocks are operated as follows: Each process maintains a scalar clock with an initial value of 0. Each time a process wants to access the CS, it assigns that request a Timestamp which is one more than the value of the clock.

The process sends the time stamped request to other processes to determine whether it can access the CS. Each time a process receives a time stamped request from another process seeking permission to access the CS [8], the process updates its clock to the maximum of its current value and the time stamp of the request. So we have the bellow categories [9, 10]:

Centralized Algorithms: In this bunch of algorithms, a central process or machine services others. One of the well-known algorithms is Agrawala and El Abbadi [11], which has three steps to synchronize processes in entering and releasing CS. This central process is called coordinator and each process should be allowed from this coordinator to use a shared resource. At the end they should inform coordinator in time of leaving CS. This method is simple to implement but has centralization's own problems.

At the centralized approach, the coordinator holds a list of processes that they are requesting the CS; grants requests in some order (random, FIFO, etc).

Distributed Algorithms: In this series of algorithms, resource dedication and retrieval is performed in distributed manner. In other words all of the available processes decide about whom enters in CS [12, 13]. Of course, in this way there will be more communication rate and fault-tolerance can be low in the cases of crashing a process [14].

The main idea is that lowest timestamp wins. Also, the asking and permission messages are combined into asks, which are only send out after a proc used the CS (if it has smaller timestamp). For example:

- Node A req CS with ts=8, Node C with ts=12.
- B acks both A&C
- C acks A (smaller ts)
- A uses CS, then sends ack to C.

In a distributed system (for example centralize algorithm) [11], every client or process that wants the critical section must be races to other processes. In the other hand, at neural networks as a distributed system this race is between cells or neurons and because the neural network is a perfect system (without crash and fault) [15], it is a good pattern for simulation and apply the result of simulation in distributed systems.

Correspondance of Neural Networks and Distributed Systems: As we know, neural network of human body is free crash and the models of neural networks are proof.

So if we adapt a distributed system and a neural network, as a result we can have a reliable and fault-tolerance system [16].

In this simulation, each of the process simulate with a message of a neuron, because they are the elements that will create the dual systems. Processes play a fundamental role in distributed systems as they form a basic for communication between different machines. An important topic for distributed systems is the migration of code between different machines. And a software agent is a special kind of process, which operates as an autonomous unit, but it is capable of corporation with other agents. Characteristics of a process can same to a neuron, because each of the neurons is intelligence and can work as a system. It has input, processing and output [13, 17, 18]. Some of the neurons can make decision unity and some of them are only transmitter the data across the neural network. In distributed system, for example in centralized algorithm, coordinator is a transmission process, that it can transmitter every process number into the queue of critical section when the process get the entrance permit. Also, in other hand at a distributed system, each process can has a request and can work (by get itself response from the system). So each process is a system because it has input, processing and output. Output of each process can uses with itself or the other processes [20].

Resourses: At the first, the resources are shared to the processes and the neurons (at two systems) and so they can use them. Secondly, critical section is exists at each system. In a neural network competitive models created and constructed for race between the neurons reach to the resources; and in a distributed system a resource as a critical section shows the competitively. Distributed system's systematic view definition is: a collection of elements that they collaborate and they have a unique goal. Also, a think or a command in the human mind is a goal of the collaboration of neurons. So a neural network is a system [21].

Trancparency: "A distributed system is a collection of independent computers that appears to its users as a single coherent system" [22]. In a neural network each cell operates separately and it is independent to the other cells. Thus, behavior of network is output of local behavior of cells. This characteristic causes the local error at the neural networks covered to final output. Too collaboration of cells causes the local errors to be reform. As a result of this property can mention to higher fault-tolerance or robustness at whole of the neural network system. In other hand, this is transparency property that mentioned at distributed systems.

Learning Operation: Learning operation that happened at neural networks is similar to the algorithms that we use at distributed systems. In fact, we learn to the distributed system with algorithms that they run on the whole of the system.

Client-server Model: As we know, a distributed system is included the clients and servers and each of them are including the processes. The neurons are same; because some of the neurons are to sense (clients) and the others are motive (servers). The other category of neurons is communication neurons that same of network equipments (cables, routers and etc.). Communication in neural network is same to a distributed system. Because, processes can communicate the data across the network that create infrastructure of distributed system and in a neural network synapses and some of the other neurons can create this infrastructure [23].

Parallel Processing: Also parallelization happens at two systems. In neural networks, all of cells that are in a one level can do operation in parallel. And the other hand, a distributed system is a suitable infrastructure for parallel processing. (In each situation shared memory and distributed memory.).

Neural network is a self organize system and so we can seem that as a control system with feedback [24, 25]. Feedback caused the system is stable [26]. As the Fig. 4 shows, continusely the system do compare of output and target and so error will be minimized. When we adapted the neural network and a distributed system, we have a stable system. Thus, the errors will be decries. Make minimum error in each system will help to reliability on that system. In other hand, decries of the errors causes the system will be fault-tolerant.

Hardware and Software: Distributed system hardware is loosely coupled. And at a neural network, if a part of network erased the reminder part (parts) will works. So hardware in a distributed system is same of a part of neural network. Also distributed system software is tightly coupled. So each part of the software is important. At a neural network thinking, is as software on the system. So if a part of software destroys, the exactly goal of the system will not reached. In other word, output of the system has a problem. In fact a neural network is not a true distributed system but it is a distributed system [22].

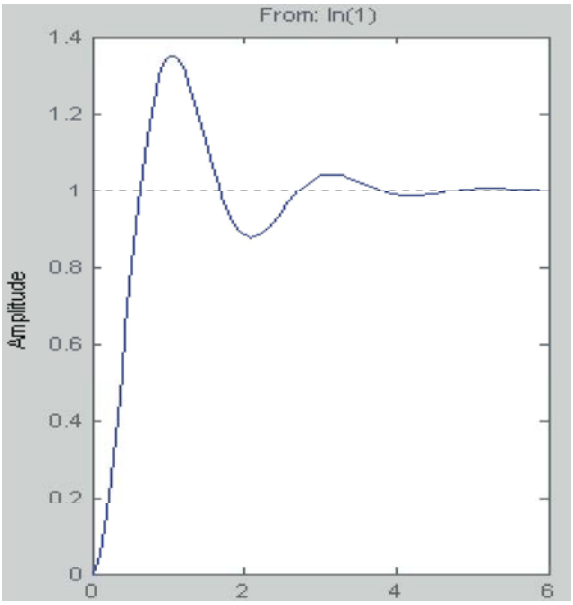

Fig. 4: Stability in feedback of neural network

MATERIALS AND METHODS

Competitive Learning: In competitive learning, as the name applies, the out put neurons of a neural network compete among themselves to become active (fired). Whereas in a neural network based on Hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time. It is this feature that may be used to classify a set of input patterns. There are three basic elements to a competitive learning rule:

- A set of neurons that are all the same except for some randomly distributed synaptic weights and which therefore respond differently to a given set of input patterns.
- A limit imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subnet of inputs, such that only one output neuron, or only one neuron per group, is active (i.e. "on") at a time. The neuron that wins the competition is called a winner-takes-all neuron.

Accordingly the individual neurons of the network learn to specialize on ensembles of similar patterns; in so doing they become feature detectors for different classes of input patterns.

In the simplest form of competitive learning, the neural network has a single layer of output neurons, each of which is fully connected to the input nodes. The network may include feedback connections among the neurons, as indicated in fig. In the network architecture described herein, the feedback connections perform lateral inhibition, whit each neuron tending to inhibit the neuron to which it is laterally connection. In contrast, the feedback synaptic connections in the network of fig. are all excitatory.

For a neuron k to be the winning neuron, its induced local field v_k for a specified input pattern $\mathbf{x}$ must be the largest among all the neurons in the network. The output signal Yk of winning neuron k is set to one; the output signals of all the neurons that lose the competition are set equal to zero. We thus write:

$y_k = 1$ if $v_k > v_j$ for all $j, j \neq k$

$y_k = 0$ Otherwise.

where the induced local field v_k represents the combined action of all the forward and feedback inputs to neurons k.

Let w_{kj} denote the synaptic weigh connecting input node j to neuron k. suppose that each neuron is allotted a fixed amount of synaptic weight (i.e., all synaptic weights are positive), which is distributed among its input nodes; that is,

$$\sum_j w_{kj} = 1 \text{ for all } k$$

A neuron then learns by shifting synaptic weights from is inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, each input node of that neuron relinquish some proportion of its synaptic weight relinquished is then distributed equally among the active input nodes. According to the standard competitive learning rule, the change Δw_{kj} applied to synaptic weight w_{kj} is defined by

$\Delta w_{kj} = \eta(x_j - w_{kj})$ if neuron k wins

$\Delta w_{kj} = 0$ if neuron k loses the competition

where n is the learning-rate parameter. This rule has the overall effect of moving the synaptic weight vector w_k of winning neuron k toward the input pattern x. It is assumed that all neurons in the network are constrained to have the same Euclidean length (neuron), as shown by

$$\sum_j w_{kj}^2 = 1 \text{ for all } k$$

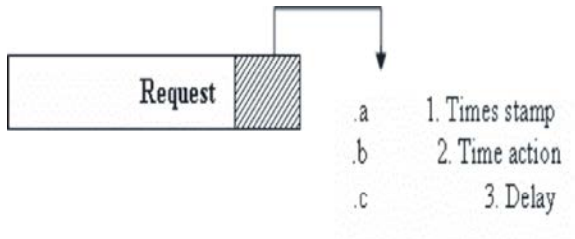

Fig. 5: Priority and Weight on three importance parameters.

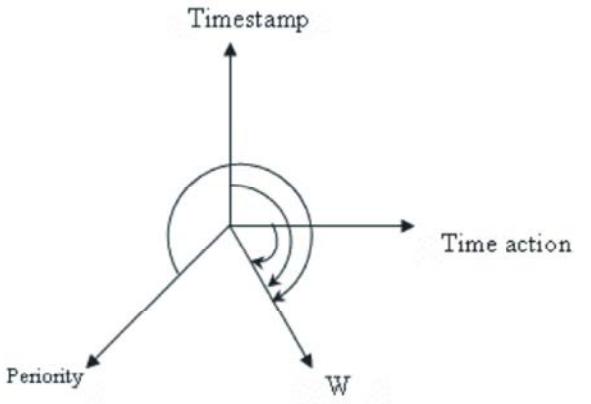

Fig. 6: Three dimension for race.

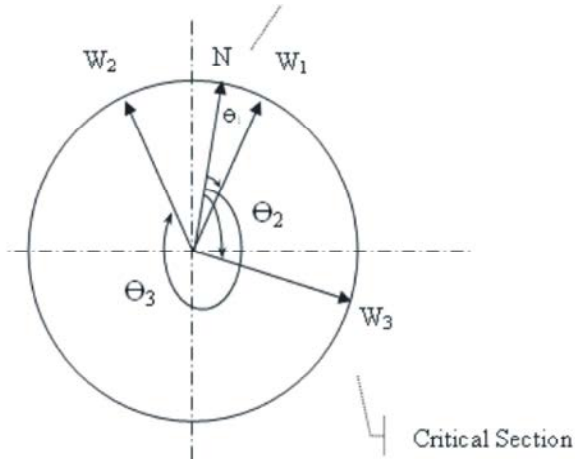

Fig. 7. Selecting an overcoming vector (request).

When the synaptic weights are properly scaled they from a set of vectors that fall on the same N- dimensional unit sphere.

Definition: One of the models of neural networks, especially for racing model is hamming [27].

This model that is too self-organized has three layers: feed back layer, race layer (racing between cells and define one cell as an overcome cell) and output layer that defines prototyping and comparing with every cell [28, 29].

Reaction the Hamming Vector: Now we assume these three important parameters are three dimensions of Hamming modeling. In the first step of this simulation according to this issue the weight vectors define in this environment.

After creation the weight vector we have:

$$\vec{w} = a_i.\vec{x} + b_j.\vec{y} + c_k.\vec{z}$$

RACE: After this stage all of the other vectors that are the results of all requests, created and race is starting. According to the Hamming low, every process that liker than the reference vector, it will be overcoming and get critical section.

So we have a buttery shape represents the circle. Because every request changes to a vector:

Request 1 (Process C1):

$$w_1 = a_{1i}.\vec{x} + b_{1j}.\vec{y} + c_{1k}.\vec{z}$$

Request 2 (Process C2):

$$w_2 = a_{2i}.\vec{x} + b_{2j}.\vec{y} + c_{2k}.\vec{z}$$

Request 3 (Process C3):

$$w_3 = a_{3i}.\vec{x} + b_{3j}.\vec{y} + c_{3k}.\vec{z}$$

That a is Priority, b is Time Action and c is Time Stamp.

In the above picture, N is Prototype or reference vector and as we see the order of wining of the vectors are: at first, w_1 second, w_2 and finally w_3. So after simulation, first request that will be enter to the Critical Section is the process equivalent w_1, next is w_2 and the end will be w_3.

Calculations for Argument Normal Vector and Other Vectors: In this section, present the formulas about calculation that related to vectors. According to below picture we have:

For calculate θ argument:

$$\cos\theta = \frac{\vec{N}.\vec{W}_i}{|N|.|W_i|} = \left(\frac{a \times a_i + b \times b_i + c \times c_i}{\sqrt{a^2 + b^2 + c^2}\sqrt{a_i^2 + b_i^2 + c_i^2}}\right)$$

And then:

$$\theta = ArcCos\left(\frac{a \times a_i + b \times b_i + c \times c_i}{\sqrt{a^2 + b^2 + c^2}\sqrt{a_i^2 + b_i^2 + c_i^2}}\right)$$

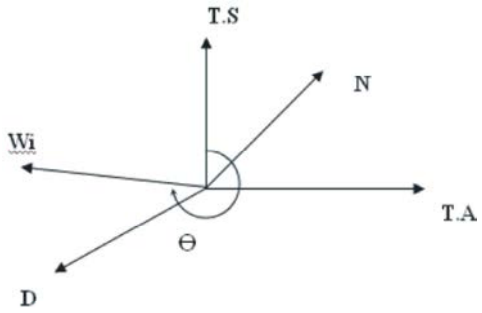

Fig. 8: T.A. (Time Action) T.S. (Time Stamp) and D (Delay).

For Θ : 1 to n do

Sort (Θi) : Min ... . Max

Input sort (Θi) ⟶ *Add*

Queue – reply

Θ2 << Θ1 << ... << Θi

Fig. 9: An algorithm for queuing.

Finding Shortest Θ and Overcome Cell: An algorithm will be present for define the overcome cells. Each vector (cell) that has shortest argument it is overcoming. And other cells (processes) are waiting in a queue that wills show: Of course the other approaches can uses for queuing.

Dead Units: These units are dead cells or in other word cells that are very far form other cells. They haven't chance of involving to other cells in races.

Advantages: One of the more important advantages is that all of the distributed algorithms only time stamp is a parameter to value judgment, but with this simulation other important parameter too have value judgment and we can learn to system that what is the effect of each parameter (Time Stamp, Time Action and Priority). Priority is a parameter that related to different systems will different and administrator can define and change it on the different situation of system.

In other hand neural networks are perfect distributed systems by characteristics for example fault tolerance, reliability, scalability, fairness and etc.

So when coincide that networks to distributed systems, we can optimize all of distributed systems parameters and present the solutions to have better characteristics.

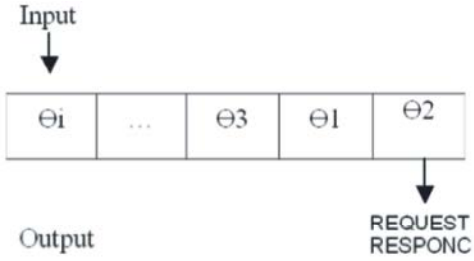

Fig. 10: A queue for sorting requests.

Self Organize Map: In this part we continue our presentation of self-organizing maps. These networks are based on competitive learning; the out put neurons of the network compete among themselves to be activated or field, with the result that only one output neuron, or one neuron per group, is on at any one time. In a Self-Organizing Map, the neurons are placed at the nodes of a lattice that is usually one-or two- dimensional. Higher-dimensional maps are also possible but not as common. The neurons become selectively tuned to various input patterns (stimuli) or classes of input patterns in the course of a competitive learning process. The locations of the neurons so tuned (i.e., the winning neurons) become ordered with respect to each ordered in such a way that a meaningful coordinate system for different input features is created over the lattice. A SOM is therefore characterized by the formation of a topologic map of the input patterns in which the spatial locations (i.e. coordinates) of the neurons in the lattice are indicative of intrinsic statistical features contained in the input patterns, hence the name "Self-Organize Map". As a neural model, the SOM provides a bridge between two levels of adaptation:

- Adaptation rules formulated at the microscopic level of a single neuron.
- Formation of experimentally better and physically accessible patterns of feature selectively at the microscopic level of neural layers.

Because a SOM is inherently nonlinear, it may thus be viewed as a nonlinear generalization of principal components analysis.

The development of SOM as a neural model is motivated by a distinct feature of the human brain: The brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computational maps. In particular, sensory inputs such at tactile are mapped onto different areas of the cerebral cortex in a topologically ordered manner. Thus the computational map constitutes a basic building block in the information-processing infrastructure of the

nervous system. A computational map is defined by an array of neurons representing slightly differently tuned processors or filters, which operate on the sensory information-bearing signals in parallel. Consequently, the neurons transform input signals into the place-coded probability distribution that represents the computed values of parameters by sites of maximum relative activity within the map. The information so derived is of such a form that it can be readily accessed by higher-order processors using relatively simple connection schemes.

Two Basic-mapping Models: Anyone who examines a human brain cannot help but be impressed by the extent to which the brain is dominated by cerebral cortex. The brain is almost completely enveloped by the cerebral cortex, which obscures the other parts. For sheer complexity, the cerebral cortex probably exceeds any other known structure in the universe. What is equally impressive is the way in which different sensory inputs (motor, soma to sensory, visual, auditory, etc. are mapped onto corresponding areas of the cerebral cortex in an orderly fashion. The use of computational maps offers the following properties:

- At each stage of representation, each incoming pieces of information is kept in its proper context.
- Neurons, dealing with closely related pieces of information, are close together so that they can interact via short synaptic connections.

Our interest lies in building artificial topologic maps that learn through self-organizing in a neuron biologically inspired manner. In this context, the one important point that emerges from the very brief discussion of computational maps in the brain is the principle of topographic map formation, which may be stated as:

> *The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature of data drawn from the input space.*

This principle has provided the neuron biological motivation for two basically different feature-mapping models described herein.

Self-Organizing Map: The principal goal of the SOM is to transform an incoming signal pattern of arbitrary dimension into a one- or two- dimensional discrete map and to perform this transformation adaptively in a topologically ordered fashion. Each neuron in the lattice of SOM is fully connection to all the source nodes in the input layer. This network represents a feed forward structure with a signal computational layer consisting of neurons arranged in rows and column [30]. Each input pattern presented to the network typically consists of a localized region or "spot" of activity against a quit background. The localization and nature of such a spot usually varies from one realization of the input pattern to another. All the neurons in the network should therefore be exposed to a sufficient number of different realizations of the input pattern to ensure that the self-organization process has a chance to mature properly. The algorithm responsible for the formation of the self-organizing map proceeds first by initializing the synaptic weights in the network. This can be done by assigning them small values picked from a random number generator, in so doing, no prior order is imposed on the feature map. Once the network has been properly initialized, there are three essential processes involved in the formation of the self organizing map, as summarized three:

Competition: For each input pattern, the neurons in the network compute their respective values of a discriminate function. This discriminate function provides the basis for competition among the neurons. The particular neuron with the largest value of discriminate function is declared winner of the competition. Let m denote the dimension of the input (data) space. Let an input pattern (vector) selected at random from the input space be denoted by $x = [x_1,x_2,...,x_m]^T$ The synaptic weight vector of each neuron in the network has the same dimension as the input space. Let the synaptic weight vector of neuron j be denoted by $w_j = [w_{j1},w_{j2},...,w_{jm}]$ j = 1, 2…l where l is the total number of neurons in the network. To find the best match of the input vector **x** with the synaptic weight vectors w_j, compare the inner products $w_j^T x$ for j=1, 2…l and select the largest this assumes that the same threshold is applied to all the neurons; the threshold is the negative of bias. Thus, by selecting the neuron with the largest inner product $w_j^T x$, we will have in effect determined the location where the topological neighborhood of excited neurons is to be centered. The best matching criterion, based on maximizing the inner product $w_j^T x$, is mathematically equivalent to minimizing the Euclidean distance between the vectors x and w_j. If we use the index $i(x)$ to identify the neuron that best matches the input vector x, we may then determine $i(x)$ by applying the condition $I(x) = \arg.\min||x–w_j||, j = 1, 2… l$ which sums up the essence of the competition

process among the neurons. The particular neuron i that satisfies this condition is called the best-matching or winning neuron for the input vector x. In other word, a continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network. Depending on the application of interest, the response of the network could be either the index of the winning neuron (i.e., its position in the lattice), or the synaptic weight vector that is closest to the input vector in a Euclidean sense.

Corporation: The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among such neighboring neurons. Let $h_{j,i}$ denote the topological neighborhood centered on winning neuron i and encompassing a set of excited (cooperating) neurons, a typical one of which is denoted by j. Let $d_{i,j}$ denote the lateral distance between winning neuron i and excited neuron j. then we may assume that the topological neighborhood $h_{j,i}$ is a unimodal function of the lateral distance $d_{j,i}$, such that it satisfied two distinct requirements:

- The topological neighborhood $h_{j,i}$ is symmetric about the maximum point defined by $d_{i,j} = 0$; in other words, it attains its maximum value at the winning neuron i for which the distance $d_{j,i}$ is zero.
- The amplitude of the topological neighborhood $h_{j,i}$ decrease monotonically whit increasing lateral distance $d_{j,i}$, decaying to zero for $d_{j,i} \to \infty$; this is a necessary condition for convergence. A typical choice of $h_{j,i}$ that satisfies these requirements is the Gaussian function:

$$h_{j,i(x)} = \exp(-\frac{d_{j,i}^2}{2\sigma^2})$$

- Synaptic Adaptation: this last mechanism enables the excited neurons to increase their individual values of the discriminate function in relation to the input pattern through suitable adjustment applied to their synaptic weights. The adjustments made are such that the response of the winning neuron to the subsequent application of a similar input pattern is enhanced.

Properties of the Feature Map: Once the SOM algorithm has converged, the feature map computed by the algorithm displays important statistical characteristics of the input space.

To begin with, let x denote a spatially continues input (data) space, the topology of which is defined by the metric relationship of the vectors $x \in X$. Let A denote a spatially discrete output space, the topology of which is endowed by arranging a set of neurons as the computation nodes of a lattice. Let ϕ denote a nonlinear transformation called a feature map, which maps the input space X onto the output space A, as shown by

$$\phi : X \to A$$

For example, in a neurobiological context, the input space X may represent the coordination set of somatosensory receptors distributed densely over the entire body surface. Correspondingly, the output space A represents the set of neurons located in that layer of the cerebral cortex to which the somatosensory receptors are confined.

Given an input vector x, the SOM algorithm proceeds by first identifying a best matching or winning neuron $i(x)$ in the output space A, in accordance with the feature map ϕ. The synaptic weight vector w_i of neuron $i(x)$ may then be viewed as a pointer for that neuron into the input space X; that is, the synaptic elements of vector w_i may be viewed as the coordinates of the image of neuron i projected in the input space.

Kohonen Map Model: As mentioned the self-organizing feature map or kohonen map, has a vector of input neurons connected to a two-dimensional grid of output neurons. Output nodes are extensively interconnected with many local connections. Continues input values are applied to a neural network and it is trained with unsupervised learning. The learning algorithm requires, for each neuron, a definition of a neighborhood that slowly decreases in size with time. It is based on checking the most active neuron and updating its weights.

A parallel algorithm of the Kohonen map is desired because of the large cost of determining the Euclidean distance for each neuron in order to find the winning one and the large number of iteration required to reach a stable state.

There have been some works on the parallelizing Kohonen map. Some of the works are parallel algorithms based on the training parallelism. The former work shows that if the number of training patterns assigned to a processor is small, it appears to have similar performance to the exact algorithm. However, because of the effect of training parallelism that average the change of weight, the self-organizing process occasionally fails, as in the back-propagation model.

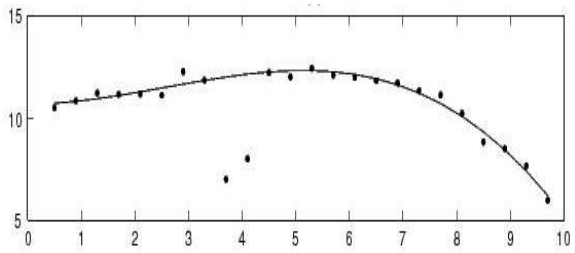

Fig. 11: Linear and non-linear behaviors.

In this model, neurons in the input and output layers are divided equally among the processors and each processor is responsible for updating the weights connecting the input neurons with the corresponding set of output neurons. The analytic speedup ratios of its parallel algorithm on the ring and mesh are obtained via simulation. However, they did not mention the exact algorithm to find the winning cell and strategy to update the weights of connections on the multiprocessor system.

Simulation

Feature Selection: One of properties of SOM is feature selection, which means: Given data from an input space with a nonlinear distribution, the SOM is able to select a set of best features for approximating the underlying distribution.

This property brings to mind the idea of principal components analysis. In Fig. 11 shows a two dimensional distribution of zero-mean data points resulting from a linear input-output mapping corrupted by additive noise. In such a situation, principal components analysis works perfectly fine: It tells us that the best description of the "linear" distribution in Fig. 11 is defined by a straight line (i.e., one-dimensional "hyperplane") that passes through the origin and runs parallel to the related vector associated with the largest value of the correlation matrix of the data. Consider next the situation described in Fig. 11, which is the result of a nonlinear input-output mapping corrupted by additive noise of zero mean. In this second situation, it is impossible for the straight-line approximation computed from principal components analysis to provide an acceptable description of the data. On the other hand, the use of a SOM built on a one dimensional lattice of neurons is able to overcome this approximation is illustrated in Fig. 11. This Fig. 11 shows linear and non-linear behaviors for the curve. The curve is linear at 0 to 5 and it is non-linear at 5 to 10.

As we know when inputs enter at a FIFO queue and as inputs, the output(s) create we will have a linear function. But usually inherent of the neural networks are based on non-linear [21]. This behavior is caused by that neural networks are not following to the FIFO queues.

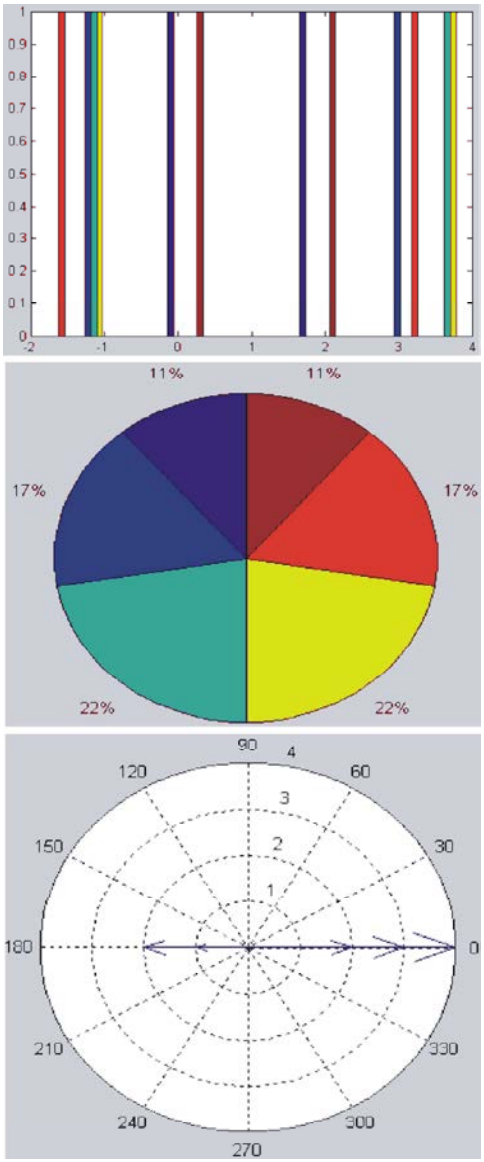

Fig. 12: Different kind of input showing.

For example, input data distribution can following to the Markova chain or the other queuing.

In precise terms we may state the self-organizing feature maps provide a discrete approximation of the so-called principal curves or principal surface and may therefore be viewed as a nonlinear generalization of principal components analysis.

Implementation of Simulation: According to our issues, inputs with their weights perform to the SOM neural network and each effective parameter has a spatially weight. So in MATLAB software we do simulation. As shows in Fig. 12 the input vector defined and with this vector neural network will train.

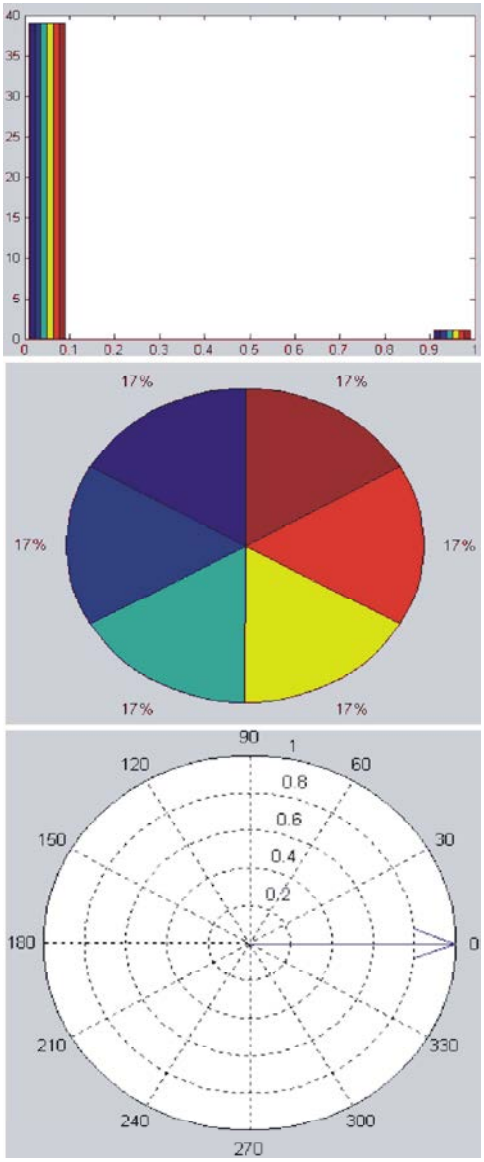

Fig. 13: Different kind of output showing.

Also, about output after competitive and effect of the weights a win input will select.

In fact, after the racing the win cell can go to the critical section and get it.

And after its work finished, the next win can go to that section. Accordingly, for make a decision to entrance the critical section of a distributed system several parameters can effective.

Impelimentation of Neural Networks: Neurocomputers: The neuroncomputers refer to the all implementation methods designed to optimize the computation of artificial neural networks [31]. The general-purpose neurocomputer is a generalized and programmable neurocomputer for emulating a range of neural network models and the spatial-purpose neurocomputer is a specialized neural network hardware implementation dedicated to a specific neural network model and therefore potentially has a very high performance. To solve a real-word or real-time application problem, the special-purpose neurocomputer may be an attractive method. However, when developing a new neural network model or testing the new application, the general-purpose neurocomputer is more attractive because of its flexibility. Furthermore, it can also used in the real-word application if its underlying architecture is a parallel computer powerful enough to fulfill the computational requirement.

Multilayered Neural Network on Distributed Memory Multiprocessor: As a parallelization and implementation method, presents a way to parallelize the back propagation model on a distributed memory multiprocessor. It is based on the spatial parallelism in which the neurons on each layer are partitioned into p disjoint sets and each set is mapped on a processor of a p-processor system.

Mapping Neural Network on a Distributed Memory Multiprocessor: The parallelism used in our model is a kind of spatial parallelism, in which a multilayered network is vertically partitioned into p sub-networks and each sub-network is mapped on a processor of the p-processor distributed memory multiprocessor (DMM) [32].

If a processor maintains only the output or the input weight values, an excessive inter- processor communication is required in either the forward or backward execution phase of the back propagation algorithm. Therefore, to simulate the sub-network as independently as possible, each processor maintains in its local memory the activation values, the error values and the input and output weight vectors of the assigned neurons. Since an input weight value of layer l is the output weight value of layer l, the same value is stored in two processors. Though this partitioning scheme results in the duplication of weight values, it avoids the complex communication requirement during he execution of the distributed back propagation algorithm.

DISCUSSION

In this paper we present a generic, descriptive form of the race model, as applied to forced choice data from a perceptual identification task. Similar to applications of

signal detection theory, we use the race model as a tool for describing accuracy and latency data under the assumption that the response alternatives accrue information in distributed system.

Other researchers have used Hamming model to describe different type of race, but such applications are few. More specifically, the Hamming race model has been applied in several domains, including perceptual identification. Our assumption that finish times are distributed is unique, providing an interpretation in terms of a race between separate pools of racers for each response alternative. Because the neural networks itself is the distribution that results from a race process, we are essentially assuming a "race-race" process, placing into a final competition the fastest racers from separate pools of racers.

Feuture Work: Presented ideas can develop to racing learning and can say to the system by set the free parameters. Future work will contrast this interpretation of these data with that provided by more traditional multi learning models, such as time tamp, time action and delay. A random walk would not describe the data in terms of the offsetting effects of the target versus the foil, but might, for instance, describe the data in terms of changes in the decision boundaries or starting point bias. In order to distinguish between these closely related explanations, we are currently employing other experimental techniques [33].

Consideration of the best-fitting parameters revealed that priming one of the alternatives affected the other alternative in an opposite manner. This suggests a "rich get richer" interaction that could be realized through various mechanisms such as lateral inhibition or capacity limitations. It is not clear at this time whether such interactions imply processing dependence between the finish time distributions, but it is crucial to answer this question since it may be the main distinction between a race process and a random walk process. Indeed, a race between perfectly negatively correlated racers is identical to a random walk. However, it is important to realize that observing opposite effects upon the race parameters does not imply a negative processing dependency. For instance, it may be that priming affects the properties of the two racers in an opposite manner, but, nevertheless, the race proceeds in an independent fashion once those properties have been established.

CONCLUSION

A seldom addressed but important hard to solve problem in mutual exclusion conflict research is whether it is possible to use algorithmically [34] (no artificially) data process.

Our purposed in this article is to outline one simulation, based approach to finding an answer. We will dynamically modeled conflict using an approach drawn from the area of artificial neural network modeling. Since we had no need to discuss the original biological type of neural network, herein we dropped the phrase "artificial". In this paper we focused on conflict between processes in distributed systems that is same to neural network; however the technique of neural network modeling is applicable to race conflict as well and we indicated very briefly how that application might work [35]. First the concept of neural networks and indicated its possible application to our modeling topic. Then we discussed what data are appropriate and, of those, which are presently available for use; too discussed our proposed modeling approach and confront some limits to achieving a fully dynamic model.

As some of the applications, we can mention to medical applications [36, 37], web engineering [38], Distributed multi thread systems [39], Group Mutual Exclusion [40], web applications [41], Sensor Networks [42, 43], Distributed Control Systems (DCSs) [44, 45, 46], Networking [47] and etc.

REFERENCES

1. Rahul, Agarwal Scott and D. Stoller, 2006. Run Time Detection of Potential Deadlocks for Programs with Locks, Semaphores and Condition Variables, PADTADIV, July 17, Portland, Maine, USA, pp: 1-10.
2. Chang, Y.I., 1996. A Simulation Study on Distributed Mutual Exclusion, J. Parallel and Distributed Computing, 33: 107-121.
3. Singhal, M., 1993. A Taxonomy of Distributed Mutual Exclusion, J.Parallel and Distributed Computing, 18(1): 94-101.
4. Lamport, L., 1978. Time, Clocks and the Ordering of Events in Distributed Systems, Comm. ACM, 21(7): 558-565.
5. Ricart, G. and A.K. Agrawala, 1981. An Optimal Algorithm for MutualExclusion in Computer Networks, Comm. ACM, 24(1): 9-17.

6. Singhal, M., 1989. A Heuristically Aided Algorithm for MutualExclusion in Distributed Systems, IEEE Trans. Computers, 38(5): 651-662.
7. Singhal, M., 1992. A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed Systems, IEEE Trans. Parallel and Distributed Systems, 3(1): 121-125.
8. Martin Raußen, 2005. Deadlocks and Dihomotopy in Mutual Exclusion Models, Dagstuhl Seminar Proceedings 04351, Spatial Representation: Discrete vs. Continuous Computational Models.
9. Tanenbaum, A.S., 1995. Distributed Operating Systems, Prentice-Hall International, Inc.
10. Tanenbaum, A.S. and M.V. Steen, 2002. Distributed Systems Principles and Paradigms, Prentice-Hall International, Inc.
11. Agrawal, D. and E.l. A. Abbadi, 1991. An Efficient and Fault-Tolerant Solution of Distributed Mutual Exclusion, ACM Trans. on Computer Systems, 9: 1-20.
12. Cao, G. and M. Singhal, 2000. Distributed Fault-Tolerant Channel Allocation for Cellular Networks, IEEE Journal of Selected Areas in Communications, 18(7): 1326-1337.
13. Simon, Haykin., 1999.Neural networks A Comprehensive Foundation" Second Edition, Prentice Hall.
14. Senouci, M., A. Liazid and D. Benhamamouch, 2007. Towards an Exclusion Mutual Tolerant Algorithm to Failures, Journal of Computer Science, 3(1): 43-46.
15. Branko Soucek and The IRIS Group, "Neural and Intelligence Systems Integration", Sixth Generation Computer Technology Series, John Willy & SONS Inc.
16. Olaf Sporns, Dante R. Chialvo, Marcus Kaiser and Claus C. Hilgetag, 2004. Organization, development and function of complex brain networks, TRENDS in Cognitive Sciences, 8(9): 418-425.
17. Neelakanda Perambur. S., 1999. Information Theoretic Aspects of Neural Networks", First Edition, CRC Press LCC.
18. Toshihide Imaruoka, Jun Saiki and Satoru Miyauchi, 2005. Maintaining coherence of dynamic objects requires coordination of neural systems extended from anterior frontal to posterior parietal brain cortices, NeuroImage, 26: 277- 284.
19. Weihua Song, Vir V. Phoha, 2004.
20. Neural Network-Based Reputation Model in a Distributed System, Proceedings of the IEEE International Conference on E-Commerce Technology, IEEE.
21. Jim Smith and Paul Watson, 2005. Fault-Tolerance in Distributed Query Processing, Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS'05), IEEE.
22. Tanenbaum, A.S. and M.V. Steen, 2002. Distributed Systems Principles and Paradigms" Prentice-Hall International Inc.
23. Scott, D., Stoller, Fred and B. Schneider, 2005. Automated Analysis of Fault-Tolerance in Distributed Systems, Formal Methods in System Design, 26L: 183-196, 2005 Springer Science + Business Media, Inc. Manufactured in the Netherlands.
24. Vijay, K., Gag and Brian Waldecker, 1994. Detection of Weak Unstable Predicates in Distributed Programs, Ieee transactions on parallel and distributed systems, 5: 3.
25. Andreas Schaad, Volkmar Lotz and Karsten Sohr, 2006. A Model-checking Approach to Analyzing Organizational Controls in a Loan Origination Process, SACMAT'06, June 7-9, Lake Tahoe, California, USA, pp: 139-149.
26. Srikanta Tirthapura, 2001. Self-Stabilizing Distributed Queuing, International Symposium on Distributed Computing (DISC), pages 209-223, USA.
27. Lippmann, R.P., 1989. Pattern Classification Using Neural Networks, IEEE Communications Magazine.
28. Williamson, P.R. and M.S. Karasik, 1993. Neural Networks at Work. IEEE Spectrum (June), Hammerstrom, Dan, pp: 26-32.
29. Henzinger, T.A., P.H. Ho and H. Wong-Toi, 1995. A user's guide to HyTech, in First Workshop on Tools and Algorithms for the Construction and Analysis of Systems: TACAS94, lecture notes in computer science, 1019, Springer-Verlag, pp: 41-71.
30. Inversion of Feed forward Neural Networks: Algorithms and Applications, 1999. Craig a. Jensen, russell d. Reed, robert j. Marks, proceedings of the ieee, 87(9): 1536-1549.
31. Stefan Wermter, Jim Austin and David Willshaw, 2001. Emergent Neural Computational Architectures based on Neuroscience, Springer, Heidelberg, New York, March.

32. Catriona Mairi Kennedy, 2003. Distributed Reflective Architectures For Anomaly Detection And Autonomous RECOVERY", School of Computer Science Faculty of Science University of Birmingham, June, DOCTOR OF PHILOSOPHY.
33. Alur, R., T.A. Henzinger and P.H. Ho, 1996. Automatic symbolic verification of embedded systems, IEEE Trans. on Software Engineering, 22: 181-201.
34. Bayat, P., M. Challenger and M.E. Shiri, 2005. A New Reliable Distributed Mutual Exclusion Algorithm" Accepted for publication and presentation in IASTED conference on Software Engineering and Applications, USA.
35. Kennedy, C.M., 2003. Distributed Reflective Architectures for Anomaly Detection and Autonomous Recovery, A PHD thesis submitted to the Faculty of Science of the University of Birmingham.
36. Jantzen, K.J., F.L. Steinberg and J.A.S. Kelso, 2005. Functional MRI reveals the existence of modality and Coordination dependent timing networks, NeuroImage, 25: 1031-1042.
37. D'Arcangelo, G., G. Panuccio, V. Tancredi and M. Avoli, 2005. Repetitive low-frequency stimulation reduces epileptic form synchronization in limbic neuronal networks, Neurobiology of Disease, 19: 119-128.
38. Jianyin Zhang, Sen Su and Fangchun Yang, 2006. Detecting Race Conditions in Web Services, Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services IEEE AICT/ICIW.
39. Stefan Savage and Thomas Anderson, 1997. Eraser: A Dynamic Data Race Detector for Multithreaded Programs Multithreaded Programs, ACM Transactions on Computer Systems, 15(4): 391-411.
40. Joffroy beauquier, Sébastien cantarell, Ajoy K. datta and Franck petit, 2003. Group Mutual Exclusion in Tree Networks, Journal Of Information Science And Engineering, 19: 415-432.
41. Bradley Long, 2005. Testing Concurrent Java Components, School of Information Technology and Electrical Engineering The University of Queensland, DOCTOR OF PHILOSOPHY.
42. Nemeroff, J., L. Garcia, D. Hampel and S. DiPierro, 2001. Application of Sensor Network Communications, IEEE.
43. Allen, S.M., D. Evans, S. Hurley and R.M. Whitaker, 2002. Communications Network Design with Mobility Characteristics, IEEE.
44. Vincenzo Piun, I994. Design of Fault-Tolerant Distributed Control Systems, ieee transactions on instrumentation and measurement, 43(2): 257-264.
45. Henrik Lonn and Rolf Snedsbg, 1995. Synchronisation in Safety-Critical Distributed Control Systems, IEEE, pp: 891-899.
46. Michael Hauf, Janek Schwarz and Andreas Polze, 2001. Role-based Security for Configurable Distributed Control Systems, 0 IEEE, pp: 111-118.
47. Dan Simon, 2001. Distributed Fault Tolerance in Optimal Interpolative Nets, IEEE Transactions on Neural Networks, 12(6): 1348-1357.