The CSI Journal on
Computer Science and Engineering
Vol. 2, No. 2&4 (b), Summer & Winter 2004
Pages 19-29
**Regular Paper**

# A Neural Network Realization of File Transfer Scheduling

Mohammad Kazem Akbari  Seyed Mehdi Hosseini-nejad  Mohammad Kalantari

Computer Engineering Department, Amirkabir University of Technology
Tehran, Iran

## Abstract

Neural network models have been successfully applied to solve a variety of problems requiring associative recall, constraint satisfaction, and optimization. This paper presents a new scheduling approach based upon a deterministic modified Hopfield model to solve "File-Transfer" scheduling, an NP-Complete constraint satisfaction problem. The proposed model is mapped onto a 2-dimensional neural network architecture for the transfer scheduling of files between various nodes of a network, by which the overall transfer times is to be minimized. Neural Network-based Scheduling is achieved by formulating the scheduling problem in terms of energy function, and by using the "Motion Equation" corresponding to the variation of energy levels. The main contribution of this work is an efficient and fast parallel algorithm under time and resource constraints, appropriate for implementation on the parallel machines. However, neurons' motion equation is the core of this guided movement mechanism which searches the scheduling space in parallel, and guarantees that the state of system mostly converges to the optimum state. Yet another important contribution of this work is the new strategy of constraints and variables containments by which the performance and efficacy of the system was considerably improved.

**Keywords:** neural networks, computer networks, parallel computing

## 1. Introduction

A large class of logical problems arising from real world situations can be formulated as optimization problems, and thus qualitatively described as a search for the best solution. In each of these optimization problems, an attempt can be made to quantify the vague criterion "best" by the use of a specific mathematical function to be optimized. Often, what is truly desired is a *very good* solution, which will be uniquely best only for simple tasks. In many situations, a very good answer computed on a time scale short enough so that the solution can be used in the choice of appropriate action is more important than a nominally-better "best" solution [1].
This is especially true in the biological and roboticstasks of perception and pattern recognition, becausethese problems typically have an immense number of variables and the task of searching for the mathematical optimum of the criterion can often be of considerable combinatorial difficulty, and hence time consuming.
Most of general purpose digital computers would fail to provide the combination of power and speed to solve the perceptual and/or combinatorial problems efficiently. One of the central goals of research in neuroscience is to understand how the biophysical properties of neurons and neural organization combine to provide such impressive computing power and speed to tackle and solve the intractable and cumbersome problems in short period of time.
it is clear from studies in anatomy, neurophysiology, and

psychophysics that part of the answer to how nervous systems provide computational power and speed is through parallel processing.

The focus of this research is to render a fast and efficient parallel algorithm under time and resource constraints for the *"File Transfer Scheduling"* (FTS), a NP-Complete problem.

Several solutions and formulations have been proposed to obtain both "fast" and "optimal" scheduling for the problem, but mostly were found very slow, restricted and in some cases impractical, due to complexity of the problem. However, no attempt has been made to identify and exploit any special structure such as Artificial Neural Networks (ANN) to solve the problem.

One of the major contributions to the area of neural networks was made in the early 1980's by John Hopfield [2], who studied an autoassociative network that has some similarities with the perceptrons, but also some important differences. Hopfield's contribution was not simply the suggestion of a suitable model, but his extensive 'analysis and study, which has led to his name being associated with the network. He developed the use of an energy function, and related the networks to other physical systems. In clear and simple terms he described how computational capabilities can be built from networks of neuronlike components. He illustrated an associative memory that can be implemented with his network [3], and later demonstrated optimization problems that could be solved [1][4]. The appearance of the Hopfield network renewed interest in previous research results and percipitated a rebirth of enthusiasm for neural networks.

This paper is organized in 5 sections to address the detail of our neural solution for the problem. In this regard the next section describes the problem (File Transfer Scheduling) and the problem complexity. Section 3 discusses our neural solution and the problem formulation in detail. In the same section we also talk about our improvements in the mathematical method. Section 4 presents the simulation analysis of the new algorithm. Finally section 5 summarizes the accomplishments of this research.

## 2. File Transfer Scheduling

File-Transfer scheduling is concerned with one fundamental problem of distributed processing, that of transfering large files between various nodes a network. In particular, we are interested in how collections of such transfers can be scheduled so as to minimize the total time for the overall transfer process. In general, minimizing the transfer time is a crucial problem in areas like Wide Area computer Networks (WAN), Local Area Networks (LAN), and telecommunications. It also can be extended to other important areas such as "multiprocessor scheduling" in a MIMD machine or in some areas like task assignment in a company and Multi-tasking operating systems.

This scheduling example belongs to the large class of NP-Complete (Non-deterministic Polynomial time Complete) problems, just as most of combinatorial optimization problems like Traveling Salesman Problem (TSP) and resource constraint scheduling problems. Bennington and McGinnis [11] pointed out that little progress has been made in the area of resource-constrained scheduling problems due to difficulties of including both the precedence and resource constraints in a mathematical formulation that maintains sufficient structure to aid in the solution.

The problem could be modeled as a labeled, undirected graph $G = (V, E)$, which we shall call the *file transfer graph* [12]. Both the vertices and edges of this graph are labeled with integers. Vertices correspond to workstations or communication centers, each of which is assumed to have the ability to communicate directly with every other center. As it can be seen in Figure 1, the label $p(v)$ of a vertex $v$ is its *port constraint*, and denotes the maximum number of simultaneous file transfers that the given vertex can engage in. Edges correspond to the files to be transferred, with the label $L(e)$ of an edge $e$ representing the amount of time needed to transfer that file. It is assumed that once the transfer of a file begins, it continues without interruption until the transfer is complete, and also forwarding is not allowed; each file is transferred directly between the centers that are its endpoints.

This problem is relevant to a variety of situations. For instance, consider a network of home computers, where each computer has an automatic dialer and interconnection is accomplished via a standard telephone network. This would correspond to our model graph with $p(v) = 1$ for each vertex. On a larger scale, one could consider the computer centers of a large company, where each has many automatic dialers. As another example, when a heavily anticipated software release isposted on the Internet, a downloading frenzy begins. Thousands of clients attempt to download a small set of files from a small number of servers all at once. Many will be repeatedly turned away as the server is unable to handle the peak load. In such an environment maximizing client utility should be the primary concern of the server. It can be shown that maximizing client utility is essentially a problem of optimal file transfer scheduling.

In the former example, we concentrate on the problem of minimizing the *makespan* of the schedule (the time interval between the beginning of the first transfer and the completion of the last transfer). In the dial-up applications, this might be motivated by the need to make all transfers during periods of low usage (or low telephone rates), and hence to find schedules which are "short" enough to fit into such intervals. However, in most cases, what is truly desired is a *very good* (not optimum) and *fast* solution.

### 2.1 Complexity of the problem

This section has been written with extensive reference to the work of Coffman et al [12]. Given a file transfer graph $G(V, E)$, *a schedule* can be viewed formally as a function $s : E \to [0, \infty]$ that assigns to each edge $e$ a start time $s(e)$, such that, for each vertex $v$ and time $t \geq 0$,

$$\left|\{e : v \text{ is an end point of } e \text{ and } s(e) \leq t \leq s(e) + L(e)\}\right| \leq p(v)$$

The *length* or *makespan* of a schedule s is the largest finishing time, i.e., the maximum, over all edges *e*, of $s(e)+L(e)$. Figure 1 illustrates file transfer graph and the timing diagram to be used in representing schedules. Our goal is to find, given a file transfer graph *G*, a schedule *s* with the minimum possible makespan.

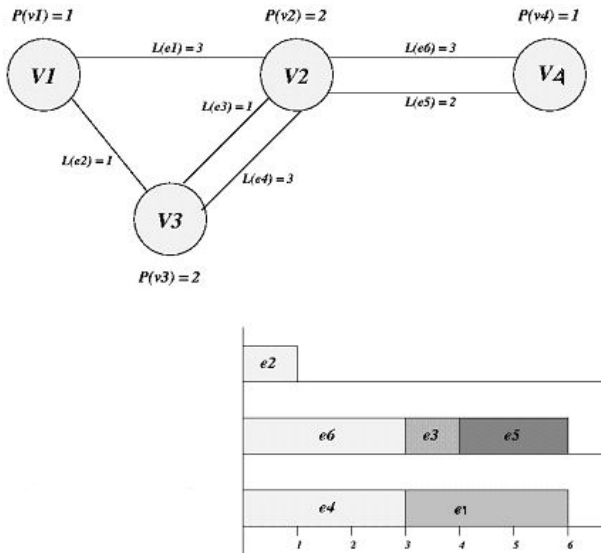



Figure 1. A File-Transfer Graph and a schedule

An elementary lower bound is obtained as follows. Let $E_u$ denote the set of files that are to be sent or received by vertex *u*. The degree of node *u* is given by $d_u = |E_u|$. Let $E_{u,v}$ denote $E_u \bigcap E_v$, the set of files to be communicated between *u* and *v*. We define $\sum_u = \sum_{e \in E_u} L(e)$ and $\sum_{u,v} = \sum_{e \in E_{u,v}} L(e)$. For consistency with this notation, we shall also use $p_u$ to denote the port constraint $p(u)$ for each vertex *u*.

The time to transmit all of the files sent or received by vertex *u* is at least $\left\lceil \sum_u / p_u \right\rceil$. Thus we have the following:

*• The optimal schedule length OPT(G) for any graph G must satisfy*

$$OPT(G) \geq \max_u \left\lceil \sum_u / p_u \right\rceil$$

Note that this lower bound is achieved by the schedule in Figure 1, where $T = \left\lceil \sum_{u_2} / p_{u_2} \right\rceil = 12/2 = 6$. But there are many cases that the optimal solution *(OPT(G))* can be substantially larger than $\max_{u \in V} \left\lceil \sum_u / p_u \right\rceil$. It is not always "easy to see" what *OPT(G)* is, however. The general decision problem "Given *G* and a bound *B*, is there a schedule s for G with makespan *B* or less?" is NP-Complete and hence unlikely to be solvable efficiently, i.e., by a traditional method. It is important to note that there are two distinct reasons why the general problem is NP-Complete, one having to do with the structural complexity of the file transfer graph and the other with the complexity involved in balancing the transmission of files of different lengths. In their paper, Coffman et al [12] illustrated this by proving that two very restricted versions of the problem are NP-Complete.

- FILE TRANSFER SCHEDULING is NP-Complete even when restricted to file transfer graphs in which the port capacity $p(v)$ of each vertex is 1, there is at most one edge between each pair of vertices, and all edges have the same length. In this case our problem can be mapped on the known NP-Complete problem *Chromatic Index* [13].
- FILE TRANSFER SCHEDULING is NP-Complete even when restricted to file transfer graphs with just two vertices, *u* and *v*, with all edges of equal length. In this case the problem can be transformed to known NP-Complete problem *Multiprocessor Scheduling* [14].

Coffman et al, in their paper have suggested several algorithms to solve the File Transfer Scheduling, in which they have tried to compute the minimum makespan in polynomial time. But the solutions are restricted by some conditions which are as follows:

- The File Transfer Graph (*G*) is bipartite and all the files have equal length.
- The graph *G* possess no simple cycle except in the trivial case of 2-vertex cycle induced by multiple edges and same value lengths.
- The *G* is a multiple-edged graph with even cycle and equal files lengths.
- The *G* is a multiple-edged graph with odd cycle, equal lengths of files, and all port capacities equal to one.

Nevertheless, the general case remains unsolved, and therefore, it might need a parallel/neural algorithm to solve the problem properly and perfectly.

# 3. Neural Solution and the Problem Formulation

## 3.1 The Hopfield's approach for the optimization problems

The general structure of the Hopfield model is illustrated in Figure 2. Neurons are modeled as amplifiers that have a sigmoid input/output function defined as :

$$V_i = g(U_i) = \frac{1}{2}(1 + \tanh l U_i) \quad (1)$$

where $U_i$ and $V_i$ are input and output voltages respectively, of the *i*th operational amplifier, and $l$ is the gain factor. If $l >> 1$, i.e., amplifier in high-gain mode, then the above equation becomes approximately a step function. Synapses are modeled by permitting the output of any neuron to be connected to the input

of any other neurons. The strength of the synapses is modeled by a resistive connection between the output of a neuron and the input to another. The amplifiers provide integrative analog summation of the currents resulting from the connections to other neurons as well as connection to external inputs.

To model both excitatory and inhibitory synaptic links, each amplifier provides both a normal output $V$ and an inverted output $\overline{V}$. The normal outputs range between 0 and 1 while the inverted amplifier produces corresponding values between 0 and -1. The synaptic link between the output of one amplifier and the input of another is defined by a conductance $T_{ij}$ which connects either the positive or the negative output of amplifier *j* to the input of amplifier *i*. In the Hopfield model, the connection between neurons *i* and *j* is made with a resistor having a value $R_{ij} = 1/T_{ij}$. To provide an excitatory connection (positive $T_{ij}$) the resistor is connected to the normal output of amplifier *j*, and for inhibitory one (negative $T_{ij}$), the resistor is connected to the inverted output of amplifier *j*. The connections among the neurons are defined by a matrix $T$ consisting of the conductances $T_{ij}$.

Hopfield has shown that a symmetric $T$ matrix ($T_{ij} = T_{ji}$) causes convergence to a stable state in which the outputs of all amplifiers are either 0 or 1. Additionally, when the diagonal entries of the T matrix are all 0 (i.e., $i = j$ and $T_{ii} = 0$, which means no self-feedback is allowed), and the amplifiers are operated in the high-gain mode, the stable states of a network of *N* neurons correspond to the local minima of the following quantity:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} T_{ij} V_j V_i + \sum_{i=1}^{N} V_i I_i \qquad (2)$$

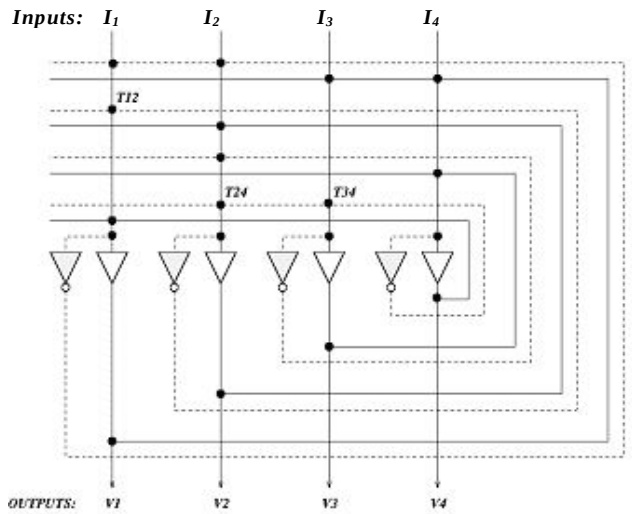



Figure 2. A Schematic for Hopfield Network with Four Neurons

where $V_i$ is the output of the *i*th neuron and $I_i$ is the externally supplied input to the *i*th neuron; it is used to set the level of excitability of the network (threshold) through constant biases. Hopfield refers to $E$ as the computational energy of the system. The input bias currents can be used to shift the input-output relation along the $U_i$ axis, and to turn on specific neurons ("clamping") by introducing a constant bias voltage (threshold). It can be easily shown that the equation of motion describing the time evolution of the circuit is:

$$\frac{du_i}{dt} = -\frac{u_i}{t_i} + \sum_{j=1}^{N} T_{ij} V_j + I_i \qquad (3)$$

where time constant $t_i$ of the *i*th operational amplifier is determined by $t_i = R_i C_i$, and $C_i$ is the total input capacitance. The total input resistance $R_i$ is the parallel combination of all input resistances given by:

$$\frac{1}{R_i} = \frac{1}{r_i} + \sum_{j=1}^{N} G_{ij} \qquad (4)$$

where $r_i$ is the input resistance at *i*th operational amplifier. In order to have the same time constant $t$ for all neurons, it is desired to have the total resistance $R$ matched. By choosing appropriate values for the synapse strengths (weights), input currents, and gains of amplifiers, the system can be programmed to find solutions to a wide variety of problems including optimization and costraint satisfaction problems [1].

### 3.1.1 Mathematical neural method, an alternative approach

This section has been written with extensive reference to the work of Dr. Takefuji [5]. Since Wilson and Pawley strongly criticized the neural network methods (specifically Hopfield model) for optimization problems [6], and in addition, after the publication of discouraging report of Paielli [7] regarding the drawbacks of Hopfield network (e.g., convergence to local minima, limited capacity of network, and inability for solving hard-learning problems), it has been widely believed that the neural network methods are not suitable for optimization problems. But Takefuji et al [5] in their continuous and unfailing efforts have demonstrated the capability of the artificial neural networks (i.e., Hopfield-like method) for solving optimization problems, over the best known algorithms and methods. In his method, Takefuji exploits the topology of Hopfield net in conjunction with both mathematical method of *McCulloch-Pitts*, and Maximum (winner-take-all) function to tackle and solve the problems. His works covers a wide variety of professional fields including game theory, computer science, graph theory, molecular biology, VLSI computer aided design, communication, and computer networks [8,9].

The McCulloch and Pitts' mathematical model was introduced in 1943 [10] in which the binary input/output function of their neuron model can be defined as follows:

$$V_i = f_{MP}(U_i) = \begin{cases} 1 & if \quad U_i > 0 \\ 0 & otherwise \end{cases} \qquad (5)$$

where $V_i$ and $U_i$ are the output and input of neuron *i* respectively, and $f_{MP}$ is called the McCulloch-Pitts neuron's input/output function. The characteristics of the McCulloch-Pitts function is nondifferentiable, nonlinear, non-decreasing, and has two binary states. The convergence of the McCulloch-Pitts function is relatively faster than of the Hopfield's sigmoid function because the Hopfield's function has infinite states in compare with two states of McCulloch-Pitts function. But, the McCulloch-Pitts model sometimes shows oscillatory behavior in neural dynamics, which is predictable. To alleviate the undesirable oscillatory behavior, *hysteresis* is embedded in the model, and the new model is called *Hysteresis McCulloch-Pitts Neural Model.* The corresponding input/output function of the new model is given by:

$$V_i = f_{MP}(U_i) = \begin{cases} 1 & if \;\; U_i > UTP \;\; (UpperTripPoint) \\ 0 & if \;\; U_i < LTP \;\; (LowerTripPoint) \\ unchanged & otherwise \end{cases} \qquad (6)$$

where UTP is always larger than LTP. This suppressing procedure (adding hysteresis) will also shortens the convergence time consequently. The input/output function of the McCulloch-Pitts binary model and also the Hysteresis McCulloch-Pitts model are illustrated in Figure 3.

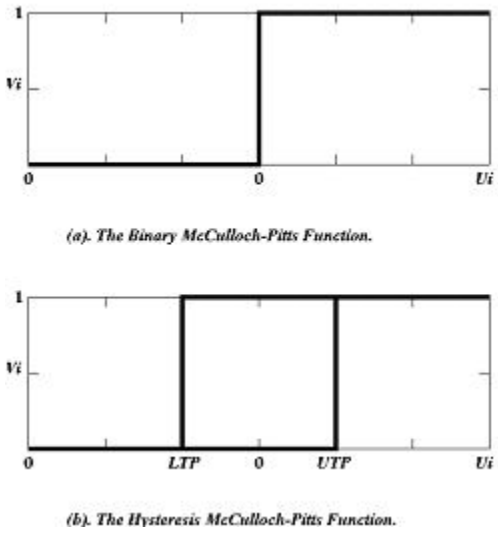



Figure 3. A Binary and a Hysteresis McCulloch-Pitts Input/Output Function

In the McCulloch-Pitts model with/without hysteresis it is not always guaranteed that the converged state is an acceptable solution. In other words, it is not guaranteed that the converged state satisfies the required constranits, thus, it may or may not be the best solution. To resolve the problem, a modified version of McCulloch-Pitts model was introduced, the *Maximum Neuron Model,* where the "winner-take-all" function is embedded in the original model. The latter model (maximum neural model) is composed of *M* clusters where each cluster consists of *n* neurons. In this model, one and only one neuron out of *n* neurons in every cluster with a maximum value is encouraged to be fired. The corresponding input/output function of the *i*th neuron in the *m*th cluster is given by:

$$V_{mi} = \begin{cases} 1 & if \;\; U_{mi} = \max\{U_{m1},...,U_{mn}\} \;\; and \;\; U_{mi} \geq U_{mj} \;\; for \; i > j \\ 0 & otherwise \end{cases} \qquad (7)$$

In the maximum neuron model it is always guaranteed to generate satisfactory solutions. Because the maximum neural model always keeps one and only one neuron to be fired in every cluster.

In order to solve an optimization problem using a neural network, it is necessary to find a proper representation of the problem. Neurons, for example, may represent objects, object attributes or perhaps relationships between objects. Optimization problems require the maximization or minimization of a function, usually while simultaneously satisfying a number of constraints. Such problems can frequently be represented as a two-dimensional array of neurons in which each neuron represents a hypothesis that may be either true or false. Therefore, the first consideration should be the representation of an entity in a neural network model. Each entity may be represented by a single neuron or by a pattern of neurons. Sometimes there is a natural one-to-one representation of an entity by a computing element.

However, solving an optimization problem with constraints satisfaction requires selecting an appropriate representation of the problem, and choosing appropriate interconnection weights and also input biases [15,16,17,18]. In this regard, the "File-Transfer Scheduling" is amenable to solution using two-dimensional Hopfield-like neural network model. The forementioned Mathematical (Hopfield-like) model is used to represent a matrix of optimized schedule-of-files in which the transfer-time of files is calculated by a modified Hysteresis McCulloch-Pitts input/output function (i.e., maximum neuron model). In the model, the hysteresis property is exploited in order to reduce the complicated oscillatory behaviors of neuron dynamics. In other words, the hysteresis with each neuron enhances the state of system to stay in the region of phase space where it is located. Therefore, the hysteresis property of each neuron suppresses the oscillatory behavior of the system so that the convergence time to the global minimum is drastically shortened.

In our neural model, the interconnections between the *i*th neuron and other neurons of the network are specified by the motion equation. Also the input value (state of input) of the *i*th neuron is given by motion equation. Practically, motin equation can be described as the partial derivatives of the computational energy function *E* with respect to the output of the *i*th neuron where *E* follows an n-variable function $E(V_1, V_2, ..., V_n)$. For instance, the motion equation of the *i*th neuron can be defined:

$$\frac{dU_i}{dt} = -\frac{\partial E(V_1, V_2, ..., V_n)}{\partial V_i} = -\frac{dE}{dV_i} \qquad (8)$$

In general, the goal of neural computation is to optimize the fabricated computational energy function. The energy function not only determines how many neurons should be used in the system but also it

specifies the strength of synaptic links between neurons. Indeed, energy function is constructed from information in the given problem, considering the required constraints and/or cost function. Practically, it is usually easier to calculate the motion equation (partial differential of the energy function) than the energy function itself. The superiority of the motion equation over energy function can be articulated as follows: its simplicity (step by step computations and ease of formulation), binary behavior, ease of application, and the flexibility in which all constraints can be incorporated. It also resolves the deficiencies of the Hopfield net which was discussed earlier. The energy function, however, can be defined:

$$E = \int dE = -\int \frac{dU_i}{dt} dV_i \qquad (9)$$

In order to numerically solve the partial differential equation or the differential equation to determine the value of motion equation, the first order *Euler method* is widely used where it is the simplest among the existing numerical methods. Therefore, based upon the first order Euler method the value of $U(t+1)$ determined as below:

$$U(t+1) = U(t) + \Delta U(t).\Delta t \qquad (10)$$

where $\Delta U(t)$ is given by Equation 8. With $\Delta t = 1$, the Equation 10 for two-dimensional neural array of the proposed model will becomes as:

$$U_{ij}(t+1) = U_{ij}(t) + \Delta U_{ij}(t) \quad for \;\; 1 \le i \le T \;\; and \;\; 1 \le j \le N \qquad (11)$$

where $N$ is the number of files to be transferred, and the $T$ is the optimum overall finishing time in which all files will be transferred. In order to solve the File-Transfer Scheduling problem, the initial values of $U_{ij}(0)$ are randomly given, and the Equation 11 is iteratively used until the state of system reaches the equilibrium state. However, the equilibrium state is not always equivalent to the optimum solution. Finally, the termination condition for our model is given by:

$$\Delta U_{ij}(t) = 0 \quad for \quad 1 \le i \le T \quad and \quad 1 \le j \le N \qquad (12)$$

The condition of the Equation 12 implies that the constraints are all satisfied. In other words, no violations can be found in the Equation 11, is either sequentially or parallelly updated. In the parallel computing method, the updating procedure can be done either in a synchronous or asynchronous scheme. In the synchronous method, clock or handshaking signals are applied for the purpose of synchronous intercommunication between neurons. In the asynchronous method, the state of every processing element (neuron) is updated asynchronously. These two methods can be rephrased as follows:

- **Synchronous.** At time $t+\Delta t$, generate the new inputs $U_{ij}(t+\Delta t)$ for *every* neuron, using the values $U_{ij}(t)$ that were computed at the last iteration.
- **Asynchronous**. At time $t+\Delta t$, generate *only one* new input $U_{ij}(t+\Delta t)$, leaving the other inputs to be computed at future iterations.

Practically speaking, in many situations in which the asynchronous method converges, the synchronous method will also converge, if the time step $\Delta t$ is assumed small enough. It should be noted that asynchronous method provides an approximation to the Euler backward approximation, which is known to be better for solution of "stiff" systems than the forward approximation.

## 3.2 Neurons' Motion Equation

Our approach to the "File-Transfer Scheduling" is based upon reformulation of the Hopfield neural network and the modified McCulloch-Pitts method. Here, we use a deterministic modified Hopfield network, shown in Figure 4, in which each neuron is simulated by a simple digital processor rather than the Hopfield's analog amplifiers (Figure 2). In this fashion each neuron processor has autonomous control and is able to perform all the neuro-computing phases (motion equation computations, learning and relaxation phases). Every neuron employs a modified hysteresis McCulloch-Pitts input/output function (the maximum method) to compute the outputs, which are defined in Equations 6, and 7.

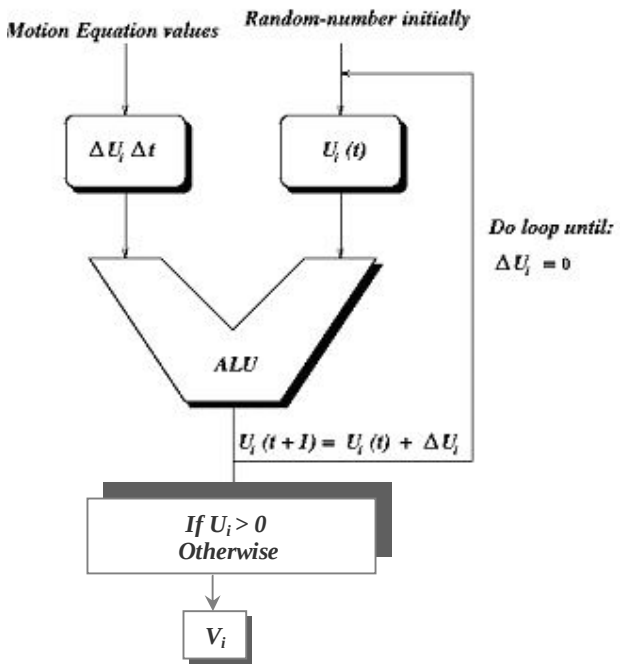



Figure 4. A Partial View of Neuron's Data Path

The input/output function is a gradient descent function. It means, based upon firing condition of neurons, as long as the motion equation satisfies the Equations 6, and 7, the predefined energy function *E* decreases monotonically. In fact, whatever the computational energy function *E* is, the motion equation forces it to decrease monotonically. It should be noted that there are some important points in our method, which can be summarized in the following:

1. Although it is possible to find the exact form of energy function *E*, we do not need it in the

process because of the motion equation $dU/dt = -dE/dV$. When we use $dU/dt$ to find a new state of the system, the energy function will be employed implicitly.
2. The termination condition and the hill-climbing factors in the motion equation together cause the convergence of system in the global minimum. The maximum neuron model also provides another enforcement factor in this regard.
3. By using the motion equation method, in fact we are employing a sparsely connected network as opposed to the Hopfield's fully connected network. Therefore, for the big problems there is no need to employ a huge number of neural connections, which would cause a spurious convergence.
4. The mathematical formulation of motion equation renders a powerful mechanism with which the hard-learning problems (e.g., *XOR, detection of connectedness and parity)* can be easily formulated and/or solved.

As mentioned earlier, the proposed Neural Network Scheduling, employs a two-dimensional neural network. Each dimension corresponds to one of the important factors involved in the scheduling process. The first dimension represents the transfer-time (schedules) which is called makespan. The second dimension depicts all the files which should be transferred in a minimum period of time. Every column of the matrix is assigned to one specific file, in which only one neuron is allowed to be fired, representing the transfer-time of the file. Therefore, the total required neurons to solve File-Transfer scheduling problem is $T*N$, where $N$ is the number of files and $T$ is the required time to transfer all the files.

### 3.2.1 Mapping constraints into motion equation

To formulate the motion equation of each neuron, we present all of the constraints, including number of files, number of nodes, number of ports, precedence, and other information in a compact form. To enforce the scheduling constraints, two kinds of forces are employed in the motion equation: the excitatory and the inhibitory forces. The inhibitory factor discourages a neuron to fire when it is going to violate the problem conditions. Conversely, the excitatory one encourages a neuron to fire when all the requirements and conditions for the firing are satisfied. Also, to avoid the local minima some hill-climbing terms were added to the motion equation, which they help the maximum neuron model to force the system to converge in the global minimum. In addition the inserted hysteresis in the motion equation supresses the oscillatory behavior of the system, from which the convergence time of the system will be considerably shortened. However, the basic parameters to be encoded in the motion equation are as follows:

- The number of files and the number of nodes.
- The source and destination nodes from/to which the files should be transferred.
- The length of each file.
- The number of ports of the nodes.
- The priority with which the files should be transferred.

Obviously, when there are $n$ files and $m$ ports available, then at most $m$ files can be sent/received independently at time $t$. This type of constraint arises when $n > m$. In this situation, it is necessary to choose $m$ out of $n$ unique files based upon their priorities. Hence, this constraint also must be considered in the motion equation. However, the motion equation of neurons which defines the File-Transfer Scheduling is given by:

$$\begin{aligned}\frac{dU_{ij}}{dt} = & -A*(\sum_{i=0}^{T} V_{ij} - 1) && \text{-1st inhibitory factor.}\\ & - B1*Freq*F1*F3 && \text{- 2st inhibitory factor.}\\ & - B2*Freq*F2*F3 && \text{- 3rd inhibitory factor.}\\ & - B3*F4*(Length_j * V_{ij}) && \text{- 4th inhibitory factor.}\\ & + C1*(H1*\sum_{i=0}^{T} V_{ij})*H2*H3*Length_j && \text{- 1st excitatory factor.}\\ & + C2*(H1*\sum_{i=k}^{T} V_{ij})*(H4*Rand)*H5 && \text{- 2nd excitatory factor.}\end{aligned} \quad (13)$$

where $j$ is the file on which we are working to schedule a Transfer-time, and $i$ is the time step in which we are trying to find out if it is appropriate to assign a Transfer-time for file $j$. The parameters $A$, $B1$, $B2$, $B3$, $C1$, and $C2$ in the motion equation are the interconnection strengths (weights) of neurons which will be defined once at the initial state of system and thereafter will be unchanged during the processing period of the system. Other parameters in the motion equation are defined as follows:

- **Freq:** It is a hysteresis factor which improves the performance of system based on "hysteresis McCulloch-Pitts function", and its value is given by;

$$Freq = \begin{cases} V_{ij} & if \quad \text{mod}(Iteration - No, 10) < 8 \\ 1 & otherwise \end{cases}$$

• **F1, F2:** These two functions determine the availability of ports in both source and destination nodes respectively, and their values are computed as follows:

$$F1 \text{ and/or } F2 = \begin{cases} X \quad (number\ of\ available\ ports) & if \quad X > 0 \\ 0 & otherwise \end{cases}$$

- **F3:** It is a precedence factor in which the priority of files could be formulated.
- **F4:** This function computes the total finishing time of the file $j$ and returns one of the following values after comparing the new computed value with the time-limit $T$.

$$F4 = \begin{cases} 1 & if \quad Time\_step(i) + Length(j) < T \\ 0 & otherwise \end{cases}$$

- **H1:** It is a normal hill-climbing factor, which takes action whenever file $j$ is not scheduled previously. Hence, it encourages the pertaining neuron to be fired as soon as possible.

$$H1 = \begin{cases} 1 & if \ \ file\ j\ has\ not\ been\ scheduled\ before \\ 0 & otherwise \end{cases}$$

- **H2:** It is another hill-climbing factor which helps to reenforce the precedence condition(s).
- **H3** = MIN(free ports of source node, free ports of destination node). This function gives some balance to the first excitatory factor.
- **H4** and **H5:** These two functions help to avoid local minima and also resolve the oscillation between some files with equal weights. They have another property with which we can reschedule file *j* at a more appropriate time step. The value of these two terms is given by:

$$H4 = \begin{cases} 0 & if\ "Rand-function" > 0.5 \\ 1 & otherwise \end{cases}$$

$$H5 = \begin{cases} 1 & if\ both\ nodes\ have\ free\ ports \\ 0 & otherwise \end{cases}$$

Moreover, the effects of overall inhibitory and excitatory factors in the motion equation can be summarized as:

- The first inhibitory factor functions if file *j* has been scheduled previously. Since the neural system must fire only one neuron per file in each column of the neural matrix, if one neuron has already been fired for file *j*, then this term inhibits the excessive firing of the neurons.
- The second inhibitory factor takes action if all the ports of source node are busy for the scheduling of file *j*.
- The third inhibitory factor helps the process if all the ports of destination node are busy for the scheduling of file *j*.
- The fourth inhibitory factor influences the motion equation if total finishing time of file *j* exceeds the time limit *T*.
- The first excitatory factor forces the process if file *j* has not been scheduled previously and also it has an urgent priority to be scheduled.
- The second excitatory factor acts if the file j has not been scheduled before and both source and destination nodes have free ports.

## 3.3 Improvements in the mathematical neural method

In addition to the main contribution of this work (providing a neural algorithm for the FTS problem), to improve the performance and efficacy of the system, a novel technique was employed to decrease the amount of required resources involved and also increase the speed of the system. Generally, it is common to apply a large amount of neurons and sometimes extra network dimensions to contain the requirements of the problem, and then satisfy all the constraints and/or conditions in computations steps of motion equation. In this work, we replaced the extra neurons by some useful look-up tables, which in contrast they not only maintained the characteristics of neural/parallel processing properly but also released a large amount of the resources (neurons). This means more speed, efficacy, flexibility, and power in the computation processes.

To achieve this, several look-up tables were devised and employed, from which a considerable better performance, and speed were observed. In the design, the very first two tables are called Length-matrix (L[files]) and Port-matrix (P[nodes]) which were employed to contain the length of all files and the number of every node's ports respectively. The third table is to save the source and destination nodes from/to which files should be transferred (SD[2][files]). The first row of matrix represents the source nodes and the second one represents the destination nodes. The forth one is called File-Node matrix (FN[nodes][files]) on which the inter-relation between files and nodes have been mapped. In other words, it illustrates the entire data dependency between the node (source/destination) and the files. The Input-matrix (U[times][files]) is another matrix which is needed to store the partial values for the next iteration of calculation and process updating. The last one is called Schedule-matrix or Output-matrix (V[times][files]) which depicts an optimum scheduling of start and finishing transfer-time of all files.

## 3.4 Neural algorithm for the FTS problem

The following procedure describes the proposed algorithm based on the first order Euler method. It generates an optimal (sometimes near optimal) scheduling for the File-Transfer problem.

- *Step 0* **(Scanning in the data):**
  - Read the number of files and nodes.
  - Read the source-destination information of each file.
  - Read the length of each file and also the number of ports for each node.
- *Step 1* **(Initialization of the network):**
  - Find the file possessing the maximum length.
  - Filling the dependency-matrix (File-Node matrix).
  - Computing the time-limit based on Lemma1: $OPT(G) \geq \max_u \left\lceil \sum_u / p_u \right\rceil$.
  - Set the weights to one ($A = B1 = B2 = B3 = C1 = C2 = 1$).
  - Set the time to zero ($t = 0$).
  - Set the output-matrix and schedul-matrix to zero ($V - matrix = 0,$ and $V1 - matrix = 0$).
  - Initialize the input-matrix (U-matrix) with random values between 0 and –10.
- *Step 2* **(Computing the output $V_{ij}(t)$):** Evaluate the value of output based on the following input/output function:

$$V_{ij}(t) = \begin{cases} 1 & if\ \ U_{ij}(t) > 0\ \ and\ \ U_{ij}(t) = \max_k U_{kj}(t)\ \ for\ \ 0 \leq k \leq i \\ 0 & otherwise \end{cases}$$

- *Step 3* **(Computing the $\Delta U_{ij}(t)\Delta t$):** Use the motion equation in Eq. 13 to compute $\Delta U_{ij}(t)\Delta t$.
- *Step4* **(Computing the input $U_{ij}(t+1)$):** Compute new inputs for neurons based on the first order Euler formula [a] : $U_{ij}(t+1) = U_{ij}(t) + \Delta U_{ij}(t)\Delta t$ where, $1 \leq i \leq T$ (time-limit) and $1 \leq j \leq N$ (number of files).
- *Step5* **(Termination condition):** If all constraints are satisfied, i.e. all $\Delta U_{ij}(t)\Delta t = 0$, then terminate

the algorithm, else increment *t* by one and go to step .

# 4. Simulation Analysis of the Neural Algorithm

The neural network scheduling algorithm described in the previous sections has been modeled in C and implemented on Pentium IV of having a 512 MB of RAM. As for the initialization part, a completely random (timed seed) data was used to evaluate the system with different settings of the data. Therefore, the behavior of the system could be generalized when we analyze the performance of system in several individual runs (e.g., 50). The crucial point in solving the File-Transfer scheduling is to acquire a proper strategy for finding the only optimum value of the overall finishing time. To achieve this, we have used some extra terms (i.e., H2 and H5) in the motion equation to encourage the system to fire the neurons as high (early) as possible in the scheduling space, with which we had dominantly actual optimal results. By this, the performance of the system was considerably improved, so that we could come up with a much better (shorter) scheduling within O(1) time (reference [5] presents a formal proof of the running time). Therefore, in most of cases the time-limit resulted from Lemma 1 ($\max_u \left\lceil \sum_u / p_u \right\rceil$) was the best choice in the determination of optimum-overall-finishing-time.

For the evaluation of the system's performance, lots of different instances of different nature and sizes: small (e.g., 6 x 4), medium (e.g., 10 x 11), large (e.g., 25 x 33), and extra large (100 x 39), with totally different sets of random data were hired and ran, for 50 times. The results were analyzed carefully and the outcomes showed an increasing nonlinear behavior (a poisson-like distribution) regarding the growth of network-size in compare with its corresponding average-iteration required to converge.

Our experimental results showed that examples with extra large files affect the system to run slower. Because the neural algorithm needs more time to calculate the transfer-time of the files and then reschedule the previously determined time-stamps for finding an optimum makespan, and in the same time solving the likely collision problem(s) in the scheduling space.

Table 1. The simulation results of ten different sizes of the FTS problem

| No. | ANN Size | # of Files | # of | Iterations | | | Conv. Time (Sec.) | | | Optimum |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Nodes | Min. | Avg. | Max. | Min. | Avg. | Max. | Conv. Ratio |
| 1 | 24 | 6 | 4 | 12 | 31.42 | 45 | .44 | .91 | 1 | 100 |
| 2 | 36 | 6 | 6 | 10 | 21.4 | 34 | .29 | .76 | 1 | 100 |
| 3 | 50 | 10 | 5 | 27 | 32.62 | 64 | 1 | 2.20 | 4 | 100 |
| 4 | 99 | 9 | 11 | 17 | 38.75 | 65 | 1 | 2.35 | 5 | 100 |
| 5 | 162 | 27 | 6 | 45 | 114.5 | 285 | 8 | 20.66 | 54 | 100 |
| 6 | 280 | 20 | 14 | 14 | 78.22 | 165 | 3 | 14.4 | 20 | 100 |
| 7 | 288 | 18 | 16 | 8 | 57.32 | 102 | 1 | 5.24 | 9 | 100 |
| 8 | 300 | 25 | 12 | 20 | 113.38 | 262 | 5 | 27.11 | 52 | 100 |
| 9 | 825 | 25 | 33 | 41 | 96.57 | 324 | 9 | 23.24 | 61 | 99 |
| 10 | 3900 | 100 | 39 | 63 | 101.4 | 137 | 64 | 121.2 | 142 | 98 |

This is also true when we deal with examples of small quantity of nodes but with a large volumes of files. The main reason here is the frequent collision problem during the process of time-stamp assignments and the optimization of overall finishing time (makespan). This gets worse when we encounter the examples which possess both property, numerous extra large files and a limited number of nodes. In this case the success rate to converge to an optimum solution will be diminished to about 98% but all of those non-optimal solutions are very good and close to optimal solutions. Table 1 depicts all the information in this respect.

However, simulation results have shown that always an optimal, sometimes near-optimal but acceptable, solution can be found quickly and efficiently, using the designated neural algorithm. It is noteworthy that the algorithm can be tuned somehow to produce rapid and optimal schedules in the networks that have specific and unalterable number of nodes and the corresponding ports. Notwithstanding, working in the areas of constraint satisfaction and combinatorial optimization problems where a real-time solution is required, the quick calculation of near-optimal solution is more attractive than a slowly computed globally optimal solution. For example, robot trajectory planning problems, aircraft altitude control problems, aircraft navigation control problems, and optimal control problems that must respond quickly to radically changing environmental conditions are of this type. It should be noted that, the above running outcomes do not represent full capacity of our neural design, because the implementation has been done on a sequential machine. Indeed, our C modeling on a sequential machine is just a simulation of the parallel machine. Of course the running time will be reduced significantly if we use a parallel architecture.

## 4.1 Modeling of FTS problem with Boltzmann and Gaussian machines

According to [20], a general framework that includes the Boltzmann machine, Hopfield net, and other neural networks is known as the Gaussian machine. A Gaussian machine is described by the following three parameters: $a$ (slop parameter of the sigmoid function), $T$ (temperature), and $\Delta t$ (time step). The operation of the net consists of the following steps:

1. Calculating the value of input to unit $U_i$ of network :

$$net_i = \sum_{j=1}^{N} w_{ij} v_j + q_i + e$$

where $e$ is random noise, which depends on the temperature *T*.

2. Changing the activity level of unit $U_i$ :

$$\frac{\Delta u_i}{\Delta t} = -\frac{u_i}{t} + net_i$$

3. Applying the output function

$$v_i = f(u_i) = 0.5\left[1 + \tanh(a u_i)\right]$$

where the binary step function corresponds to $a = \infty$.

The Hopfield machine corresponds to a Guassian machine with $T = 0$ (no noise). The Boltzmann machine is obtained by setting $\Delta t = t = 1$ , to obtain $\Delta u_i = -u_i + net_i$, or

$$u_i(new) = net_i = \sum_{j=1}^{N} w_{ij} v_j + q_i + e$$

If the noise follows a Gaussian distribution with mean value of zero and standard deviation $d = T\sqrt{8/p}$ , then the distribution of outputs has the same behavior as a Boltzmann machine with probabilistic acceptance of state transitions.
Integrating the Gaussian noise distribution (approximately), we get the following Boltzmann acceptance function:

$$\int_0^{\infty} \frac{1}{\sqrt{2pd^2}} \exp \frac{(x-u_i)^2}{2d^2} dx \approx A(i,T) = \frac{1}{1+\exp\left(-\frac{u_i}{T}\right)}$$

Solving the *"File Transfer Scheduling"* problem with Gaussian and Boltzmann machines, and comparing the results with the results obtained from previous section, we come up with Table 2 and 3 in which the results regarding Boltzmann and Gaussian machines are depicted respectively. As it can be seen, for the problem with size of large, neither Boltzmann nor Gaussian models can get converged. The analytical results showed the superiority of our approach to the Boltzmann and Gaussian machines.

## 5. Conclusions

This study presented a new scheduling approach based on a deterministic modified Hopfield model (mathematical approach) to solve "File-Transfer Scheduling", an NP-complete constraint satisfaction problem. Our model was mapped onto a 2-dimensional neural network architecture for the transfer scheduling of the files. Neural-network-based scheduling was achieved by formulating the scheduling problem in terms of energy function, and by using the "motion equation" corresponding to the variations of energy level. Constant positive and negative current biases were applied to specific neurons in the motion equation, as "excitations" and "inhibitions", respectively, to enforce the scheduling constraints. Our algorithm searches the scheduling space in parallel and converges to an optimal solution with the time complexity of near O(1). The simulation results showed that in almost all the cases our neural algorithm rapidly converges to the optimum state. The abovementioned results showed very promising when were compared with two similar models, Boltzmann and Gaussian machines, and analytical results proved the superiority of our model. As it can be seen, the results of Gaussian model were better than Boltzmann machine.

Table 2. The simulation results of Boltzmann model for ten different sizes of the FTS problem

| No. | ANN Size | # of Files | # of Nodes | Iterations | | | Conv. Time (Sec.) | | | Optimum Conv. Ratio |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Min. | Avg. | Max. | Min. | Avg. | Max. | |
| 1 | 24 | 6 | 4 | 225 | 341.2 | 465 | 22 | 28.45 | 34 | 100 |
| 2 | 36 | 6 | 6 | 254 | 275.9 | 343 | 34 | 39.3 | 43 | 100 |
| 3 | 50 | 10 | 5 | 367 | 397.8 | 484 | 165 | 185.46 | 211 | 95 |
| 4 | 99 | 9 | 11 | 143 | 173.6 | 202 | 12 | 43.56 | 62 | 73 |
| 5 | 162 | 27 | 6 | 535 | 593.65 | 685 | 223 | 262.21 | 332 | 65 |
| 6 | 280 | 20 | 14 | - | - | - | - | - | - | No Converge |
| 7 | 288 | 18 | 16 | - | - | - | - | - | - | No Converge |
| 8 | 300 | 25 | 12 | - | - | - | - | - | - | No Converge |
| 9 | 825 | 25 | 33 | - | - | - | - | - | - | No Converge |
| 10 | 3900 | 100 | 39 | - | - | - | - | - | - | No Converge |

Table 3. The simulation results of Gaussian model for ten different sizes of the FTS problem

| No. | ANN Size | # of Files | # of Nodes | Iterations | | | Conv. Time (Sec.) | | | Optimum Conv. Ratio |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Min. | Avg. | Max. | Min. | Avg. | Max. | |
| 1 | 24 | 6 | 4 | 114 | 212.3 | 324 | 12 | 16.47 | 23 | 100 |
| 2 | 36 | 6 | 6 | 124 | 174.2 | 223 | 17 | 29.56 | 47 | 100 |
| 3 | 50 | 10 | 5 | 246 | 275.6 | 323 | 28 | 65.2 | 125 | 97 |
| 4 | 99 | 9 | 11 | 86 | 108.4 | 143 | 8 | 17.8 | 27 | 87 |
| 5 | 162 | 27 | 6 | 416 | 479.1 | 542 | 184 | 196.1 | 212 | 75 |
| 6 | 280 | 20 | 14 | - | - | - | - | - | - | No Converge |
| 7 | 288 | 18 | 16 | - | - | - | - | - | - | No Converge |
| 8 | 300 | 25 | 12 | - | - | - | - | - | - | No Converge |
| 9 | 825 | 25 | 33 | - | - | - | - | - | - | No Converge |
| 10 | 3900 | 100 | 39 | - | - | - | - | - | - | No Converge |

## References


[1] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problem," *Journal of Biological Cybernetics,* Vol. 52, pp. 141-152, 1985.

[2] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. USA,* vol. 79, pp. 2554-2558, 1982.

[3] J. J. Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of Two-state Neurons," *Proc. Natl. Acad. Sci. USA,* vol. 81, pp. 3088-3092, 1984.

[4] D. Tank and J. J. Hopfield, "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems,* vol. CAS-33, no. 5, pp. 533-541, 1986.

[5] Y. Takefuji, *"Neural Network Parallel Computing,"* Kluwer Academic Publishers, 1992.

[6] G. V. Wilson and G. S. Pawley, "On Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank," *Biological Cybernetics,* vol. 58, pp. 63-70, 1988.

[7] R. A. Paielli, "Simulation Tests of the Optimization Method of Hopfield and Tank Using Neural Networks," NASA Technical Memorandom 101047, 1988.

[8] N. Yoshiike, Y. Takefuji, "Vehicle Routing Problem Using Clustering Algorithm by Maximum

Neural Networks," *In Proc. of the 2th Int. Conf. on* [9] *intelligent processing and manufacturing of materials*, IPMM'99, pp. 1109-1113, 1999.

[10] S. Bharitkov, K. Tsuchiya and Y. Takefuji, " Microcode Optimization With Neural Networks," *IEEE Transaction on Neural Networks*, vol. 10, no. 5, pp. 698-703, 1999.

[10] W. S. McCulloch and W. H. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115, 1943.

[11] G. E. Bennington and L. F. McGinnis, "A Critique of Project Planning with Constrained Resources," *Proceedings of Symposium on Theory of Scheduling and its Applications*, Raleigh, NC, 1972.

[12] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh, "Scheduling File Transfers," *SIAM Journal of Computing*, vol. 14, no. 3, pp. 744-765, 1985.

[13] I. Holyer, "The NP-Completeness of Edge-Coloring," *SIAM Journal of Computing*, vol. 10, pp. 718-720, 1981.

[14] M. R. Garey and D. S. Johnson, "Computers and Interactability: A Guide to the Theory of NP-Completeness," W. H. Freeman and Company, San Francisco, 1979.

[15] X. X. Li, Y. S. Yan, G. Qin and W. W. Liang, "Hopfield Neural Networks Optimization Method For VP routing in ATM Network," *In 5th world congress on intelligent control and automation, WCICA 2004*, pp. 1493-1470, 2004.

[16] D. Hwang and F. Fotouhi, "Modifications of Discrete Hopfiled Neural Optimization in Maximum Clique Problem," *In Proc. of the Int. Conf. on Neural Networks, IJCNN'02*, 2002.

[17] R. L. Wang, Z. Tang and Q. P. Cao, "A Hopfield Network Learning Method for Bipartite Subgraph Problem, "*IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1458-1465, 2004.

[18] E. Page and G. A. Tagliarini, "Solving Constraint Satisfaction Problems with Neural Networks," *Proc. of IEEE First Int. Conf. on Neural Networks*, 1987.

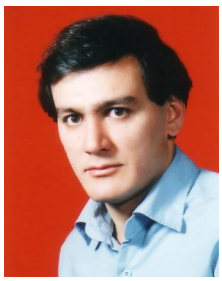
[19] Harold S. Stone, "High Performance Computer Architectures," Addison Wesley, 1990.

[20] L. V. Fausett, *Fundamentals of Neural Networks*, Prentice Hall, 1994.

[a] This is the main parallelizable formula in our algorithm. There are many parallel architectures (such as mesh, hypercube and connection machine) by which this formula can be computed in parallel [19].

**Mohammad Kazem Akbari** received B.sc. in Computer Engineering from National (Beheshti) University in 1984 and M.Sc. and Ph.D. in Computer Engineering from Case Western Reserve University in 1991 and 1995 respectively. His research interests are in neural networks, system design, parallel processing, grid computing, computer architecture, testing and fault tolerance.

**Seyed Mehdi Hosseininejad** received the B.sc. in computer engineering from Isfahan Univ. of Tech. in 1987 and M.sc. from Case Western Reserve university in 1991. His research interests are in high availability computing, database design, specification and analysis, with a focus on critical systems, neural networks based optimization and grid computing.



**Mohammad Kalantari** received the B.Sc. in Computer Engineering from Iran Univ. of Sci. & Tech. (IUST) in 1998 and M.Sc. degrees in Computer Engineering from Amirkabir Univ. of Tech. (Tehran Polytechnic) in 2001. Currently he is the Ph.D. candidate in Tehran Polytechnic university. His research interests are in networking, grid computing, scheduling, stochastic processes, distributed objects and software architecture.