IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN:2319-765X. Volume 10, Issue 2 Ver. VI (Mar-Apr. 2014), PP 25-37
www.iosrjournals.org

Constructing of an Artificial Neural Networks to Minimize Total Completion Time and Total Tardiness

Tariq Salih Abdul-Razaq[1] , Faez Hassan Ali[2]
[(1)]*Mathematics Department, College of Sciences / University of Al-Mustansiriya, Iraq.*
[(2)]*Mathematics Department, College of Sciences / University of Al-Mustansiriya, Iraq.*

Abstract: *This paper presents an approach to schedule n jobs with processing times and due dates on a single machine based on Artificial Neural Network. The purpose of this paper to find a schedule that minimize a function of the sum of completion time and sum of tardiness (i,e to minimize the multiple objective functions* (ΣC_i,ΣT_i)). *Neural network technique was be found to be effective and used to select the best efficient and optimal schedule which minimizes the sum of completion time and sum of tardiness.*

Keywords: *Back Propagation, Multiple Objective Functions, Neural network, Particle Swarm Optimization.*

I. Introduction

Various applications, such as communications, routing, industrial control, and production planning employ scheduling concepts. Most problems in these applications are confirmed to be NP-complete or combinatorial problems. The machine scheduling problem is to find the optimal processing order of these jobs on each machine to minimize the given objective function. [1].

Human beings are constantly thinking since ages about the reasons for human capabilities and incapabilities. Successful attempts have been made to design and develop systems that emulate human capabilities or help overcome human incapabilities. A number of mechanisms which seems to enable human brain to handle various problems. These mechanisms include association, generalization and self-organization. A Neural Network (NN) is not programmed to solve a problem-instead, it learns to solve a problem [2].

In 1994, Willems and Rooda [3] looked at first formulating a job shop scheduling problem as an integer programming problem, and using a NN to solve the resultant integer programming problem.

Foo and Takefuji, 1998, employed integer linear programming NNs to minimizing the total starting times of all jobs with a precedence constraint [4].

In 2001, Chen and Huang [1] investigate the employs of the competitive Hopfield NN to resolve a multiprocessor problem with no process migration, time constraints (execution time and deadline), and limited resources.

In 2002, Hamad et al [5], present an approach for scheduling under a common due date on static single machine problem based on artificial NN. The objective is to minimize the total earliness and the total tardiness cost.

In 2013 paper of Muralidhar and Alwarsamy [6] considers the problem of scheduling jobs on parallel machines with the combined objective to minimize the makespan, total tardiness and total earliness using NN technique.

In this paper, first we will compute the input units of NN then initialize the weights connected between the nodes of NN which is learned by BP and PSO for minimize the multiple objective functions (ΣC_i,ΣT_i)) and single objective function ($\Sigma C_i+\Sigma T_i$)) then compare the learning results obtained from the learning process of NN finally, evaluate which is better in such scheduling problem employing the precedence rule concept.

Most of real world decision problems involve multiple and conflicting objectives that need to be tackled while respecting the various constraints. In multi-objective problems, there may not exist one solution, which is best w.r.t all objectives. There exists a set of solutions, which are better than the other solutions in the search space when all the objectives are considered but are inferior to the solutions in one or more objectives and these solutions are called non-dominated solutions. In multi-objective optimization, the optimal solution is generally called **Pareto optimal solution** [7].

Scheduling problem with multiple objective can be formulated as follows: minimize $F(s)=(f_1(s),(f_2(s),\ldots,f_k(s))$ s.t. $s\in S$ where s is a solution, S is the set of feasible solutions, k is number of objectives in the problem, F(s) is the image of s in the k-objective space and each $f_i(s)$, $i=1,\ldots,k$ represents one minimization objective.

The more common notations which used in scheduling are:

n : number of jobs
p_i : Processing time of job i
d_i : Due date of job i

C_i : Completion time of job i
T_i : the Tardiness of job i
SPT : Shortest Processing Time
EDD : Earliest Due Date
MSP : Machine Scheduling Problem
MOF : Multi-Objective Function
BAB : Branch and Bound

II. Problem Formulation

The problem of scheduling N={1,2,…,n} the set of n jobs which are processed on a single machine to minimize the multi-criteria may be stated as follows. Each job i∈N has is to be processed on a single machine which can handle only one job at a time, job i has a processing time p_i and due date d_i, all jobs are available for processing at a time zero. If a schedule σ=(1,2,…,n) is given, then the earliest completion time $C_i = \sum_{j=1}^{i} p_j$ for each job i can be computed and consequently the tardiness of job i $T_i=\max\{C_i-d_i,0\}$ is easy to compute. Our objective is to find a schedule σ∈S (where S is the set of all feasible schedules) that minimizes the multicriteria $(\Sigma C_i,\Sigma T_i)$ for the $1//(\Sigma C_i,\Sigma T_i)$ problem. This problem belongs to simultaneous optimization and written as:

$$\left.\begin{array}{ll} \text{Min } \{ \Sigma C_i , \Sigma T_i \} & \\ \text{Subject to} & \\ C_i \geq p_i, & i=1,2,\ldots,n. \\ T_i \geq C_i-d_i, & i=1,2,\ldots,n. \\ T_i \geq 0, & i=1,2,\ldots,n. \end{array}\right\} \quad \ldots(P_1)$$

It's clear that there are two special cases for the problem (P_1). The 1st one is: $1//Lex(\Sigma C_i,\Sigma T_i)$ problem. The 2nd one is $1//\Sigma C_i+\Sigma T_i$ problem which can be written as:

$$\left.\begin{array}{ll} \text{Min } \{ \Sigma C_i + \Sigma T_i \} & \\ \text{Subject to} & \\ C_i \geq p_i, & i=1,2,\ldots,n. \\ T_i \geq C_i-d_i, & i=1,2,\ldots,n. \\ T_i \geq 0, & i=1,2,\ldots,n. \end{array}\right\} \quad \ldots(P_2)$$

The aim for the problem (P_2) is to find a processing order of the jobs on a single machine to minimize the sum of total completion times and the total tardiness, which is a single object and can be minimized by BAB method.

III. Dominance Rules (DR)

Reducing the current sequence is done by using several DR's. DR's usually specify whether a node can be eliminated before its lower bound (LB) is calculated. Clearly, dominance rules are particularly useful when a node can be eliminated which has a LB that is less than the optimum solution. The DR's are also used within the BAB procedure to cut nodes that are dominated by others. These improvements lead to very large decrease in the number of nodes to obtain the optimal solution.

Emmons [8] has introduced essential conditions to find an optimal solution of ΣT_i in a single machine problem after proving some theorems.

Theorem (1) (Emmon's) [8]: For the $1/ /\Sigma T_i$ problem, if $\mathbf{p_i \leq p_j}$ **and** $\mathbf{d_i \leq d_j}$ then there exists an optimal sequencing in which job i sequencing before job j.

Definition (1) [9]: A feasible schedule σ is **Pareto optimal** (PO), or non-dominated (efficient) w.r.t. the performance criteria f and g if there is no feasible schedule π such that both $\mathbf{f(\pi) \leq f(\sigma)}$ **and** $\mathbf{g(\pi) \leq g(\sigma)}$, where at least one of the inequalities is strict.

3.1 Digraph Representation

Definition (2) [10]: A **graph** (G) is a finite set of points, called **vertices** or **nodes (V)**, together with a finite set of **edges**, each of which joins a pair of vertices. An edge joining a vertex to itself is called a loop.

Definition (3) [10]: If G is a graph that has n vertices, then the **matrix A(G)**, whose i, j^{th} element is 1 if there is at least one edge between V_i and V_j and zero otherwise, is called the **adjacency matrix** of G.

Definition (4) [10]: A **directed graph** or a **digraph** is a finite set of points, called vertices or nodes (V), together with a finite set of **directed edges**, each of which joins a ordered pair of distinct vertices.

Remark (1):

- A digraph G contains no loops.

- There are no multiple edges.
- The directed edge V_iV_j is different from V_jV_i.
- A(G) of G of a digraph need not be symmetric.

Example (1):

Fig. (1) shows digraph (G) consists of 5 nodes:

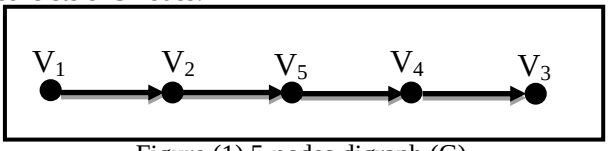


Figure (1) 5-nodes digraph (G).

The adjacency matrix A(G) of G is:

$$A(G)=\begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{array}\begin{array}{c} \begin{array}{ccccc} V_1 & V_2 & V_3 & V_4 & V_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Notice that the up-diagonal elements of A(G) are the binary complement of down- diagonal elements of A(G).

Definition (5) [10]: In digraph G with n vertices, for every pair of inidviduals V_i, V_j, either V_i dominates V_j or V_j dominates V_i, but not both. Then G is **complete digraph** and often called **dominated digraphs**.

Theorem (2) [10]: Let A(G) be the adjancacy matrix of digraph G and let the rth power of A(G) be B_r: $[A(G)]^r = B_r = [b_{ij}^{(r)}]$.

Then the i, jth element of B_r, $b_{ij}^{(r)}$, is the number of ways in which V_i, has access to V_j in r stages.

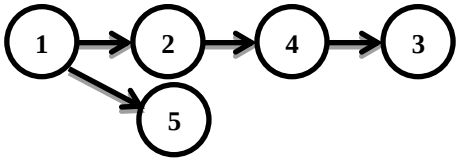
3.2 Adjancancy Matrix of Dominated Scheduling Sequences

Form Emmon's Theorem [8], we can find an optimal sequence for $1//\Sigma T_i$ problem, where the DR's are act here. Let us denote the number of dominated jobs by N(d) and the number of non-dominated jobs by N(nd).

Example (2)

In Table (1), for the following 5 jobs we can conclude the following DR:

Table (1) DR for example of 5 jobs for $\mathbf{N_{nd}=3}$.

J_i	p_i	d_i	$N_d=7$	Dominated jobs (Digraph G)	Nondom. Jobs
1	2	3	1→2 1→3 1→4 1→5 2→3 2→4 4→3		2 ↔ 5 3 ↔ 5 4 ↔ 5
2	3	6			
3	4	9			
4	3	7			
5	2	11			

The adjancancy matrix A(G) of grpah G is:

$$A(G)=\begin{array}{c} \\ J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \end{array}\begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & J_3 & J_4 & J_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & a_{25} \\ 0 & 0 & 0 & 0 & a_{35} \\ 0 & 0 & 1 & 0 & a_{45} \\ 0 & \bar{a}_{25} & \bar{a}_{35} & \bar{a}_{45} & 0 \end{bmatrix} \end{array}$$

Remark (2)

- The element i,jth is 1 if J_i dominated J_j, otherwise it's 0.
- The element i,jth is assigned to a_{ij} since we have no information about the precedessor between J_i and J_j (non-dominated jobs), therfore, the element j,ith $a_{ji}=\bar{a}_{ij}$.

From the above remark, A(G) can obtained as follows:

$$A(G)=[a_{ij}]=\begin{cases} 1, & \text{if } J_i \text{ proceed } J_j \\ 0, & \text{if } J_i \text{ not proceed } J_j \text{ or } i=j \\ a_{ij}, & \text{or } \bar{a}_{ij}=a_{ji} \text{ for non-dominated jobs} \end{cases} \quad ...(1)$$

3.3 Test the Subjection of A(G) to DR

Now we attempt to determine weither that A(G) is subject to DR or not? First we arise A(G) to power k=n, let $B=A^k(G)$ if B=O (where O is n×n zero matrix) then A(G) is subject to DR and it's a legal matrix, then we can find the corresponding sequence σ, otherwise, if $B=A^k(G)\neq O$, $\forall k\in N$, then the sequence σ is illegal, has local loop and not subject to DR.

For instance, recall example (2), if the vector $(a_{25},a_{35},a_{45})=(1,1,1)$, then $A_1(G)$ will be:

$$A_1(G)=\begin{array}{c} \\ J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \end{array}\begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & J_3 & J_4 & J_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The sequence σ_1 which can be obtained from $A_1(G)$ is $\sigma_1=(1,2,4,3,5)$ which is subject to DR shown in Table (1). We notice that $A_1^5(G)=O$ (zero matrix), but it's not for $A_1^i(G)$, i=1,2,3,4. In general, $A^n(G)=O$ for adjacency matrix subject to DR.

For another instance, if the vector $(a_{25},a_{35},a_{45})=(1,1,0)$, then $A_2(G)$ will be:

$$A_2(G)=\begin{array}{c} \\ J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \end{array}\begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & J_3 & J_4 & J_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

For $A_2(G)$, the corresponding sequence σ_2 which can be obtained from $A_2(G)$ is not illegle since it has the local loop $J_5\rightarrow J_4\rightarrow J_3\rightarrow J_5$, and this not allowed, generally, in combintorial problems. To do more test, we notice:

$$A_2^2(G)=A_2^5(G)=\begin{array}{c} \\ J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \end{array}\begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & J_3 & J_4 & J_5 \end{array} \\ \begin{bmatrix} 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}\neq O$$

This mean $A_2^i(G)=A_2^{i+3}(G)\neq O$, for i=2,3,4,..., and this loop will be continue. In general $A^k(G)\neq O$ the sequence σ_2 has local loop, $\forall k\in N$.

So we suggest an algorithm to find out that the sequence σ (or A(G)) is subject to the DR. This algorithm is called **Subjection Test (ST) Algorithm.**

Algorithm ST(A(G))=Boolean

Step(1): **READ** an n×n matrix A(G). { A(G) adjancancy matrix }
COMPUTE σ from A(G).
Step(2): **COMPUTE** $B=[A(G)]^n$.
Step(3): **IF** B = O **THEN** σ is subject to DR, TF=True.
ELSE σ is not subject to DR, TF=False.
END {ENDIF}
Step(4): **STOP**.

Remark (3):

From example (2), notice that we can examine $N_{nd}=2^3=8$ states instead of 5! states to find number of effecient sequences for problem (P_1). In this mannar we can find an algorithm to find number of effecient sequences for problem (P_1) based on dominiance rule, we called it **Some Effecient Sequences of (P_1) using Dominiance Rule (SESDR) algorithm**.

SESDR Algorithm

Step(1): **READ** n-Jobs Problem Data . { p_i *and* d_i }
Step(2): **COMPUTE** dominance rule D(k,1)=i, D(k,2)=j. {*Emmon's Theorem*}
COMPUTE non-dominance rule ND(k,1)=i, D(k,2)=j.
Step(3): **CONSTRUCT** adjancancy matrix $A=[a_{ij}]$. {*equation* (1) }
$K=2^{N(nd)}$, C=ϕ.
Step(4): **FOR** i=1:K

S=binary(i-1). {*length(S)=N(nd)*}
FOR j=1:N(nd)
$A_i(ND(k,1),ND(k,2))=S(j)$, $A_i(ND(k,2),ND(k,1))=S'(j)$.
END {ENDFOR j}
CALL ST(A_i) algorithm. { *subjection test for* A }
IF ST(A_i)=True **THEN COMPUTE** σ_i, C = C U { σ_i }.
ELSE ignore A.
END. {ENDFOR i }

Step(5): **FIND** Set of effeceint sequences E={ σ_{i1}, σ_{i2},..., σ_{ik}} from dom. set C.
Step(6): **STOP**.

Example (3):

Recall example (2), the set of effecient sequences for problem (P_1) is:
ES = {(1,5,2,4,3):(37,9), (1,2,5,4,3):(38,8), (1,2,4,5,3):(39,6)}.

Table (2) shows the results of applying SESDR algorithm on n=5 for problem (P_1).

Table (2) the results of applying SESDR algorithm on n=5 for problem (P_1).

i	Binary	Sequence σ	MOF	Subjection test decision
0	000	1 5 2 4 3	(37,9)*	True
1	001	Has local loop	---------	False
2	010	Has local loop	---------	False
3	011	Has local loop	---------	False
4	100	1 2 5 4 3	(38, 8)	True
5	101	1 2 4 5 3	(39,6)	True
6	110	Has local loop	---------	False
7	111	1 2 4 3 5	(41,7)	True

The shaded cells represent efficient points and the MOF with * sign is SPT sequence.

IV. Artificial Neural Networks (ANN)

Artificial neural networks (ANN), or simply called neural networks, refer to the various mathematical models of human brain functions such as perception, computation and memory. It is a fascinating scientific challenge of our time to understand how the human brain works. Modeling NNs facilitates us in investigating the information processing occurred in brain in a mathematical or computational manner [11].

Feed-Forward Neural Networks: Feed-forward NN (FNN), also referred to as multilayer perceptrons (MLPs), has drawn great interests over the last two decades for its distinction as a universal function approximator.

FNN features a supervised training with a highly popular algorithm known as the **Error Back-Propagation** (EBP) algorithm [11].

Let us consider elementary feedforward architecture of m neurons receiving n inputs. Its output and input vectors are, respectively

$$\mathbf{o} = [o_1,o_2,...,o_m]^t, \quad \mathbf{x} = [x_1,x_2,...,x_n]^t \qquad \ldots(2)$$

Weight w_{ij} connects the i^{th} neuron with the j^{th} input. The double subscript convention used for weights in this paper is such that the 1^{st} and 2^{nd} subscript denotes the index of the destination and source nodes, respectively. We thus can write the activation value for the i^{th} neuron as:

$$net_i = \sum_{j=1}^{n} w_{ij} x_j \text{ , } i = 1,2,...,m \qquad \ldots(3)$$

The following nonlinear transformation [Equation (3)] involving the activation function $f(net_i)$, for i = 1,2,..., m, completes the processing of x. The transformation, performed by each of the m neurons in the network, is expressed as:

$$o_i = f(\mathbf{w}_i^t \mathbf{x}), \text{ for } i = 1,2,..., m \qquad \ldots(4)$$

where weight vector w_i contains weights leading toward the i^{th} output node and is defined as follows:

$$\mathbf{w}_i = [w_{i1},w_{i2},...,w_{in}]^t, \; i = 1,2,..., m \qquad \ldots(5)$$

Introducing the nonlinear matrix operator Γ, the mapping of input space **x** to output space **o** implemented by the network can be expressed as follows:

$$\mathbf{o}=\Gamma[Wx] \qquad \ldots(6a)$$

where **W** is the weight matrix, also called the connection matrix:

$$W=[w_{ij}], \; i=1,\ldots,m, \; j=1,\ldots,n \qquad \ldots(6b)$$

$$\Gamma[\cdot]=diag(f(\cdot)) \qquad \ldots(6c)$$

Note that the nonlinear activation functions f(.) on the diagonal of the matrix operator Γ operate component wise on the activation values net of each neuron.

This type of network can be connected in cascade to create a multilayer network. In such a network, the output of a layer is the input to the following layer. Even though the FNN has no explicit feedback connection when **x**(t) is mapped into **o**(t), the output values are often compared with the "teacher's" information, which provides the desired output value [12].

V. NN Learning Algorithms

The user can uses several different training methods although the basic training algorithm is slow compared to other methods, it can provide, in some cases a better representation of the training set.

5.1 Back Propagation

The BP algorithm is the most effective and most widely used supervised learning method for the training of multilayered NN, which based on error correction learning rule. BP has two distinct properties; performs stochastic gradient descent in weight space and it's simple to compute locally. BP is an iterative gradient algorithm designed to minimize the mean-squared error between the desired output and the actual output for a particular input to the NN [13].

Basically, BP learning consists of two passes through the different layers of the network: **a forward pass** and **backward pass**.

During the forward pass the synaptic weights of the network are all fixed, during the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error – correction rule.

The learning rate (η) determines the portion of weight needed to be adjusted. However, the optimum value of η depends on the problem.

The momentum (α) determines the fraction of the previous weight adjustment that is added to current weight adjustment. It accelerates the network convergence process.

During the training process, the learning rate and the momentum are adjusted to bring the network out of its local minima, and to accelerate the convergence of the network [12].

BP Algorithm

1. Initialize network weights values.
2. Repeat the steps (3-7) until some criterion is reached: (for each training pair).
3. Sums weighted multiplied by input and apply activation function equal compute output of hidden layer.

   $$h_j = f\left(\sum_{i=1}^{n} x_i \; w_{ij}\right) \qquad j = 1,2,\ldots,m$$

   m: The actual output of the hidden neuron *j* for the input signal *i*.

   Sums weighted multiplied by the output of the hidden layer and apply the activation function of the computed output layer.

   $$y_k = f\left(\sum_{j=1}^{m} h_j \, w_{jk}\right) \qquad k = 1,2,\ldots,o$$

   o: The actual output of output neuron *k*
4. Compute back propagation error: $\delta_k = (d_k - y_k)\, f'(y_k)$
5. Calculate weight correction term: $\Delta w_{jk}(n) = \eta \delta_k h_j + \alpha \Delta w_{jk}(n-1)$
6. Sums delta input for each hidden unit and calculate error term:

   $$\delta_j = \sum_{k=1}^{o} \delta_k w_{ij} \, f'(h_j)$$
7. Calculate weight correction term: $\Delta w_{ij}(n) = \eta \delta_j x_i + \alpha \Delta w_{ij}(n-1)$
8. Update weights: w_{jk} (new) = w_{jk} (old) + Δw_{jk} , w_{ij} (new) = w_{ij} (old) + Δw_{ij}
9. End.

5.2 Particle Swarm Optimization (PSO)

PSO was originally developed by a social-psychologist J. Kennedy and an electrical engineer R. Eberhart in 1995 and emerged from earlier experiments with algorithms that modeled the "flocking behavior" seen in many species of birds. Where birds are attracted to a roosting area in simulations they would begin by flying around with no particular destination and in spontaneously formed flocks until one of the birds flew over the roosting area. PSO has been an increasingly hot topic in the area of computational intelligence. It is yet

another optimization algorithm that falls under the soft computing umbrella that covers genetic and evolutionary computing algorithms as well [14].

PSO Algorithm

The PSO algorithm depends in its implementation in the following two relations:

$$v_{id}=w*v_{id}+c_1*r_1*(p_{id}-x_{id})+c_2*r_2*(p_{gd}-_{id}) \quad ...(7a)$$

$$x_{id} = x_{id} + v_{id} \quad \ldots(7b)$$

where c_1 and c_2 are positive constants, r_1 and r_2 are random function in the range $[0,1], x_i=(x_{i1},x_{i2},\ldots,x_{id})$ represents the i^{th} particle; $p_i=(p_{i1},p_{i2},\ldots,p_{id})$ represents the best previous position (the position giving the best fitness value) of the i^{th} particle; the symbol g represents the index of the best particle among all the particles in the population, $v=(v_{i1},v_{i2},\ldots,v_{id})$ represents the rate of the position change (velocity) for particle i [15].

The original procedure for implementing PSO is as follows:

1. Initialize a population of particles with random positions and velocities on d-dimensions in the problem space.
2. PSO operation includes:
   a. For each particle, evaluate the desired optimization fitness function in d variables.
   b. Compare particle's fitness evaluation with its pbest. If current value is better than pbest, then set pbest equal to the current value, and p_i equals to the current location x_i.
   c. Identify the particle in the neighborhood with the best success so far, and assign it index to the variable g.
   d. Change the velocity and position of the particle according to equations (7a,b).
3. Loop to step (2) until a criterion is met.

The PSO algorithm is vastly different than any of the traditional methods of training. PSO does not just train one network, but rather training networks. PSO builds a set number of ANN and initializes all network weights to random values and starts training each one. PSO compares each network's fitness. The network with the highest fitness is considered the global best.

Each neuron contains a position and velocity. The position corresponds to the weight of a neuron. The velocity is used to update the weight. If a neuron is further away then it will adjust its weight more than a neuron that is closer to the global best [14].

5.3 Fitness Criterion of NN

One of these stopping criterions is the fitness value. Since the BP and PSO algorithms are chosen to be a supervised learning algorithms, then there are observed values of (O_i) and desired output values of (G_i). These two values have to be compared, if they are closed to each other, then the fitness is good, else the algorithm must continue its calculations until this condition is satisfied or the specified number of iterations is finished.

The corrections to the weights are selected to minimize the residual error between O_i and G_i output. The Mean Squared Error (MSE) is one of the tests for the comparison process:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(O_i - G_i\right)^2 \quad \ldots (8)$$

Where n is the number of the compared categories.

The number of interconnections for NN for one hidden layer is m(n+o), where n, m and o is the number of nodes in the input, hidden and output layer respectively.

VI. Constructing of Multi-Layer Neural Network for Single Machine

In scheduling environment, jobs that complete early must be held in finished goods inventory until their due date, while jobs that complete after their due dates may cause a customer to shut down operations. Therefore, an ideal schedule is one in which all jobs finish on their assigned due dates. This can be translated to a scheduling objective in several ways.

The neural network that is proposed for the single machine to minmize the multiple objective functions (ΣC_i,ΣT_i)) for scheduling problem is organized into three layers of processing units. There is an input layer of 10 units, a hidden layer, and an output layer that has a single unit. The number of units in the input and output layers is dictated by the specific representation adopted for the schedule problem [16]. In the proposed representation, the input layer contains the information describing the problem in the form of a vector of continuous values. The 10 input units are designed to contain the following information showed in Table (3) for each of the n jobs that have to be scheduled [5].

Table (3) 10 units information of input layer units of NN.

Input units									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\frac{p_i}{M_p}$	$\frac{d_i}{M_d}$	$\frac{Sl_i}{M_{sl}}$	$\frac{\alpha_i}{10}$	$\frac{\beta_i}{10}$	$\frac{\bar{p}}{M_p}$	1	$\frac{\overline{Sl}}{M_{sl}}$	$\sqrt{\frac{\sum(p_i-\bar{p})^2}{n\times\bar{p}^2}}$	$\sqrt{\frac{\sum(Sl_i-\overline{SL})^2}{n\times\overline{Sl}^2}}$

where Sli slack for job i s.t Sli = $d_i - p_i$,
M_p longest processing time among the n jobs = max $\{p_i\}$,
M_d longest due date among the n jobs = max $\{d_i\}$,
Msl largest slack time for the n jobs = max {Sli}, $\alpha_i=\beta_i=1$, $i\in N$, N={1,2,...,n}.

$\bar{p}=\sum_{i=1}^{n}\frac{p_i}{n}$ and $\overline{Sl}=\sum_{i=1}^{n}\frac{Sl_i}{n}$.

Each job is represented by a 10-input vector, which holds information particular to that job and in relation to the other jobs in the problem. The output unit assumes values that are in the range of 0.10-0.90, the magnitude being an indication of where the job represented at the input layer should desirably lie in the schedule. Low values suggest lead positions in the schedule; higher values indicate less priority and hence position towards the end of the schedule. The target associated with each input training pattern is a value that indicates the position occupied in the optimal schedule. The target value G_i for the job holding the i^{th} position in the optimal schedule is determined as in equation (10) [5];

$$G_i=\left\{0.1+0.8\times\left(\frac{i-1}{n-1}\right)\right\},\qquad i=1,2,\ldots,n \qquad ...(10)$$

Equation (10) ensures that the n target values are distributed evenly between 0.1–0.9. The number of units in the hidden layer is selected by trial and error during the training phase. The final network for single machine with multi objective has 8 units in its hidden layer and 1 unit of output layer, therefore known as 10-8-1 network.

Example (4)

Recall example (2) for single machine. The 5-jobs are converted first into their vectors representation by using the set of information of Table (1) input units (1-10). The result of this pre-processing stage is presented in Table (3) where the vectors J_1-J_5 represented job numbers 1-5, respectively. To solve the schedule problem, each vector is presented individually at the input layer of the NN. A feed forward procedure of calculations generates a value that appears at the output unit for each of the 5 input vectors. The output computed by the NN for each of the input vectors is given in column before the right most column of Table (4).

Table (4) the 10 input units of example (1) for n=5.

Job	Input units										O_i	MSE
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**		
1	0.5	0.03	0.11	0.1	0.1	0.7	1	0.5	0.27	0.6	0.29	
2	0.75	0.06	0.33	0.1	0.1	0.7	1	0.5	0.27	0.6	0.46	
3	1.0	0.09	0.56	0.1	0.1	0.7	1	0.5	0.27	0.6	0.63	0.0005
4	0.75	0.07	0.44	0.1	0.1	0.7	1	0.5	0.27	0.6	0.54	
5	0.5	0.11	0.10	0.1	0.1	0.7	1	0.5	0.27	0.6	0.62	

Scheduling the jobs in the order of the increasing output values results in the job schedule J_1-J_2-J_4-J_5-J_3 for (P_1) (or (P_2)) with objective value (39,6)=45, this sequence is an optimal schedule, so the NN scheduling gives one of the effecient solutions.

VII. Neural Network Training Results

To train the NN, each vector with their output is presented individually at the input layer and output layer of the NN. Before we disscuss the NN training we will describe the proposed **NN-Scheduling algorithm** using BP and PSO learning algorithms with or without using dominince rule.

NN-Scheduling Algorithm

Step(1): **READ** (n) { *n=number of jobs*}
READ (p_i , d_i) { *processing time and due date*}
Step(2): **DEFINE** input units { *as input equations in table* (3) }
Calculate G_i { *actual value as in equation* (10) }
Step(3): **INITIALIZATION**

Wjkl=random (-1,1) { *weight in layer j, node k connected node* l }
Vjkl=random (V_{min},V_{max}) {*velocity values for PSO only* }
Step(4): **CHOOSE** training algorithm TA=B or P {*B=BP , P=PSO*}
Step(5): IMSE = Maxint; Error= 0.0001; Threshold=0.01{*specified by experiance*}
Iter=1; NI=1000; BSC=Mxint; BST=Maxint;
WHILE (Iter ≤ NI) OR (IMSE ≤ Error) **DO**
Iter = Iter + 1;
IF TA='B' **THEN CALL** BP Algorithm
ELSE CALL PSO Algorithm.
Compute MSE { *using equation* (8) }
σ=**SORT** (O_i); (SC(σ),ST(σ))=**MOF**(σ);
IF (SC(σ)<BSC) **OR** (ST(σ)<BST) **THEN**
σ'=σ; BSC=SC(σ); BST=SC(σ); IMSE=MSE;
END; {*ENDIF*}
END; { *ENDWHILE* }
Step(6): **Best sequence = σ' , with minmium BSC and BST;**
Step(7): **END**;

While the NN-scheduling algorithm when using dominiance rule will be as follows:

NN-Dom-Scheduling Algorithm

Step(1): **READ** (n)
⋮
Step(5): **FIND** DR of sequence { *generate sequence* π }
Step(6): IMSE = Maxint; Error= 0.0001; Threshold=0.01{*specified by experiance*}
Iter=1; NI=1000; BSC=Mxint; BST=Maxint;
WHILE (Iter ≤ NI) OR (IMSE ≤ Error) **DO**
Iter = Iter + 1;
IF TA='B' **THEN CALL** BP Algorithm
ELSE CALL PSO Algorithm.
Compute MSE { *equation* (8) }
π_1=**SORT** (O_i);
Compute (SC(π_1),ST(π_1))=**MOF**(π_1); {*SC*=Σ*Ci*, *ST*=Σ*Ti*)
CALL SESDR algorithm (π_2)
Compute (SC(π_2),ST(π_2))=**MOF**(π_2);
IF (SC(π_1)≤SC(π_2)) **AND** (ST(π_1)≤ST(π_2)) **THEN** σ=π_1 {*Def* (1)}
ELSE σ=π_2;
Compute (SC(σ),ST(σ))=**MOF**(σ);
IF (SC(σ)<BSC) **OR** (ST(σ)<BST) **THEN**
σ'=σ; BSC=SC(σ); BST=SC(σ); IMSE=MSE;
END; {*ENDIF*}
END; { *ENDWHILE* }
Step(7): **Best sequence = σ' , with minmium BSC and BST**;
Step(8): **END**;

Training is considered completed after an average more than 10000 cycles using a 10-8-1 configuration. A cycle is concluded after the network has been exposed once, in the course of the BP or PSO algorithms, to each one of the available training patterns. The trained NN is used to find job schedule for our problem.

Example (5)

Recall example (2) for problem (P_1), in this example the NN-scheduling learned by BP without DR for problem (P_1) , the output results are illustrated Table (5).

Table (5) learning the NN-scheduling using BP without DR for (P_1) for n=5.

Iter.	MOF	Actual					Output					Seqence	MSE
1	(47,16)	0.5	0.3	0.7	0.9	0.1	0.52	0.39	0.36	0.40	0.49	3 2 4 5 1	0.106
4	(44,16)	0.7	0.5	0.9	0.1	0.3	0.47	0.48	0.42	0.34	0.43	4 3 5 1 2	0.071
5	(41,17)	0.9	0.5	0.1	0.3	0.7	0.58	0.61	0.57	0.67	0.53	5 3 1 2 4	0.102
7	(39,11)	0.7	0.1	0.9	0.3	0.5	0.48	0.60	0.62	0.48	0.49	4 1 5 2 3	0.082
378	(39,6)*	0.1	0.3	0.7	0.9	0.5	0.44	0.44	0.56	0.47	0.48	1 2 4 5 3	0.068
6516	(39,6)*	0.1	0.3	0.7	0.9	0.5	0.10	0.30	0.71	0.88	0.50	1 2 4 5 3	0.0001

The total computation time is 7 seconds. The * sign indicates the effeient point.

Now we propose the following computations of training NN for different n of (P_1):

1. Compare the results of NN-learning methods for BP and PSO without DR (WODR) for problem (P_1).
2. Using BP NN-learning (BP-NN) method with DR (WDR) for problem (P_1).
3. Compare the results of BB- and PSO-NN WODR for problem (P_2).
4. Using BP-NN method with DR (WDR) for problem (P_2).

AAE : **A**verage of **A**bsolute **E**rror.

$$AAE = \frac{|EffV - EffP|}{EffV} = \begin{cases} 0, & \text{if } EffV = EffP \text{ or } EffP \text{ another effecient value} \\ k \neq 0, & \text{otherwise} \end{cases}$$

Where in the following tables:

EffV: The current efficient value, **EffP**: Efficient point, **CE**: Complete enumeration, **BAB**: Branch and Bound is used for only a time t≤1000 seconds, **CT**: Completion time in seconds, **NI**: number of iterations, and **CR**: comparative results.

Tables (6,7,8) show the CR of BP- and PSO-NN WODR for (P_1) for different n.

Table (6) CR of BP and PSO-NN WODR for (P_1) for n=5,8,10.

n	Eff. (CE)	NN-BP				NN-PSO			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
5	(37,9) (38,8) (39,6)	(39,6) (37,9)	3	88 479	(0,0)	(39,6)	3	6	(0,0)
8	(117,61) (120,60)	(131,74)	11	20	(0.08,0.18)	(117,61)	4	1	(0,0)
10	(170,101) (171,99) (178,98)	(180,105)	18	481	(0.01,0.04)	(170,101) (171,99)	9	1 4	(0,0)

Fig. (2) depicts CR of NI and MSE for BP and PSO-NN WODR for (P_1) for n=10.

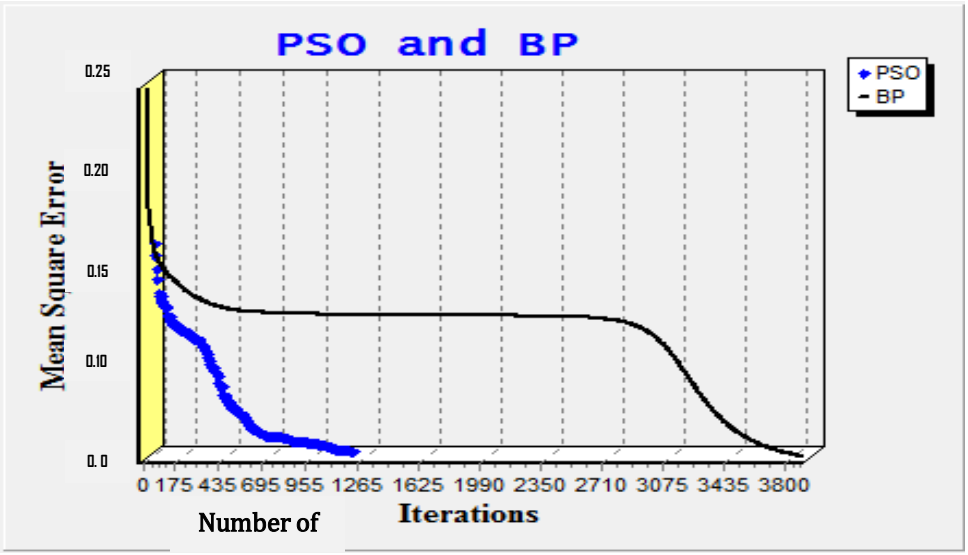


Figure (2) CR of NI and MSE for BP and PSO NN-learning WODR for(P_1), n=10.

Table (7) CR of BP- and PSO-NN WODR for (P_1) for n=12,15,18.

n	some Eff. (BAB)	NN-BP				NN-PSO			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
12	(198,85) (199,82) (200,79) ⋮	(213,102) (220,94)	22	34 201	(0.07,0.10)	(198,85)	10	1	(0,0)
15	(281,133) (282,132) (283,130) ⋮	(305,146)	22	101	(0.08,0.09)	(281,133) (284,131)	10	4 6	(0,0)
18	(439,235) (440,232) (441,231) ⋮	(502,304)	29	866	(0.13,0.23)	(439,235) (440,232) (444,231)	21	2 7 14	(0,0)

Table (8) CR of BP- and PSO-NN WODR for (P_1) for n=20,(10),50.

n	Eff. (BAB)	NN-BP				NN-PSO			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
20	(576,351) (578,347) (579,345) ⋮	(670,431)	38	263	(0.14,0.19)	(576,351) (578,348) (582,347)	38	1 6 7	(0,0)
30	(1449,1121) (1450,1119)	(1795,1478)	40	47	(0.19,0.24)	(1449,1122)	64	15	(0,0)
40	(2974,2511) 2477,2510)	(3643,3185)	43	43	(0.21,0.21)	(2974,2511)	76	1	(0,0)
50	(5026,4472)	(5634,5102)	46	137	(0.11,0.12)	(5026,4488)	91	2	$(0,10^{-3})$

Table (9) and Table (10) show the CR of BP-NN methods WODR and WDR for problem (P_1) for different n ($n \leq 20$).

Table (9) CR of BP-NN WODR and WDR for (P_1) for n=5,8,10.

n	Eff. (CE)	NN-BP without DR				NN-BP with DR			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
5	(37,9) (38,8) (39,6)	(39,6) (37,9)	3	88 479	(0,0)	(37,9) (38,8) (39,6)	1	1 19 28	(0,0)
8	(117,61) (120,60)	(131,74)	11	20	(0.08,0.18)	(117,61) (120,60)	5	31 40	(0,0)
10	(170,101) (171,99) (176,98)	(180,105)	18	481	(0.01,0.04)	(170,101) (171,99) (176,98)	15	268 415 605	(0,0)

Table (10) CR of BP-NN WODR and WDR for (P_1) for n=12,15,18.

n	Eff. (BAB)	NN-BP without DR				NN-BP with DR			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
12	(198,85) (199,82) (200,79) (202,77) (205,75)	(213,102) (220,94)	22	34 201	(0.07,0.10)	(199,82) (200,79) (202,77) (205,76) (208,75)	27	7 40 122 579 854	(0,0)
15	(282,132) (285,128) (288,127) ⋮	(305,146)	22	101	(0.08,0.09)	(282,132) (284,131) (285,128) (288,127) (294,126)	18	10 58 69 1767 2319	(0,0)
18	(444,227) (447,226) (450,225) ⋮	(502,304)	29	866	(0.13,0.23)	(447,226) (452,227) (454,225)	579	195 260 341	(0,0)

Table (11) shows CR of BP-NN WDR and PSO-NN for problem (P_1) for chosen n=(11,14,17,20).

Table (11) CR of BP-NN WDR and PSO-NNfor (P_1) for chosen n=11,14,17,20.

n	Eff(BAB)	NN-BP with DR				NN-PSO			
		Eff. of P_1	CT	NI	AAE	Eff. of P_1	CT	NI	AAE
11	(159,59) (160,56) (161,53) (166,50)	(159,59) (160,56) (161,53) (166,50)	25	3 6 19 46 64	(0,0)	(159,59) (162,54) (177,53)	10	2 4 6	(0,0)
14	(266,126) (267,124) (268,121) (270,119) (273,118) (276,117)	(267,124) (269,122) (270,119) (273,118) (276,117) (279,116)	21	26 27 534 632 641 2267	(0,0)	(266,126) (268,125) (269,122)	10	6 10 11	(0,0)
17	(392,197)	(392,203)	193	198	(0,0)	(387,205)	21	2	(0,0)

	(395,196) (398,195) ⋮	(393,198) (399,195)		613 626		(388,202) (389,201) (390,199)		4 8 15	
20	(576,351) (578,347) (579,345) ⋮	(608,357)	--	11	(0.05,0.02)	(576,351) (578,348) (582,347)	38	1 6 7	(0,0)

Table (12) shows the summery of CR of BP-NN WODR,WDR and PSO-NN for effecient points (Eff.) of problem (P_1) and for optimal values (OV) of (P_2) which is derived from results of problem (P_1) for n=10,(10),50.

Table (12) the summery of CR (BV=sum(Eff.)) of BP-NN WODR,WDR and PSO-NN for (P_1) and (P_2) for n=10,(10),50.

n	Eff. (BAB)	NN-BP without DR		NN-BP with DR		NN-PSO	
		BV=sum(Eff.)	AAE	BV=sum(Eff.)	AAE	BV=sum(Eff.)	AAE
10	(170,101)=271 (171,99)=270 (176,98)=274	(180,105)=285	0.056	(170,101)=271 (171,99)=270 (176,98)=274	0 0.004 0.015	(171,99)=270 (170,101)=271	0 0.004
20	(578,347)=925 (579,345)=924 (581,343)=924 ⋮	(670,431)=1101	0.189	(608,357)=965	0.042	(576,351)=927 (578,348)=926 (582,347)=929	0 0.003 0.001
30	(1449,1121)=2570 (1450,1119)=2569	(1795,1478)=3273	0.231	----------	----	(1449,1122)=2571	0
40	(2974,2511)=5485 (2977,2510)=5487	(3643,3185)=6828	0.245	----------	----	(2974,2511)=5485	0
50	(5026,4472)=9498	(5634,5102)=10736	0.130	----------	----	(5026,4488)=9514	0.002
MAE			0.153		0.010		0.001

The shaded cells are the effeceint points for problem (P_1) and the optimal values for problem (P_2) in the same time copmared with effecient points coloumn.

Table (13) shows the summery of CR of BP-NN WODR,WDR and PSO NN-learning method for BV of (P_2) for n=10,(10),50.

Table (13) CR of BP-NN WODR,WDR and PSO-NN for (P_2) for n=10,(10),50.

n	OV (CE,BAB)	NN-BP without DR		NN-BP with DR		NN-PSO	
	Optimal Value	Best Value	AAE	Best Value	AAE	Best Value	AAE
10	(171+99)=270	(177+107)=284	0.05	(171+99)=270	0	(170+101)=271	0
20	(579+345)=924	(788+569)=1375	0.33	(594,359)=953	0.03	(580+345)=925	0
30	(1453+1116)=2569	(1690+1353)=3043	0.19	----------	----	(1449+1121)=2570	0
40	(2975+2510)=5485	(3347+2896)=6243	0.14	----------	----	(2974+2516)=5490	0
50	(5030+4468)=9498	(6081+5550)=11631	0.23	----------	----	(5026+4472)=9498	0
MAE			0.17		0.01		0

Where MAE is the mean absolute error. The shaded cells are the best values for problem (P_2) copmared with optimal values (OV) coloumn.

In figure (3), the CR of of iterations and MSE for BP- and PSO NN methods without DR for (P_2) for n=10 are shown.

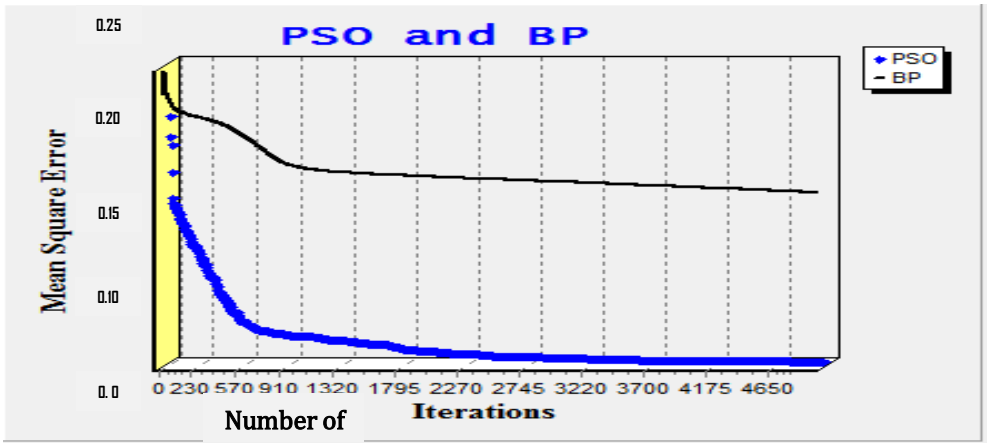


Figure (3) CR of NI and MSE for BP- and PSO NN WODR for (P_2) for n=10.

VIII. Conclusions

1. A single machine with multicriteria has been studied. The objective was to find an effeceint and an best scheduling of the sum of completion time and sum of tardiness (i,e to minimize the multiple objective functions (ΣC_i,ΣT_i)). A neural network model was developed to solve this problem. It was found that the multi layer neural network gave effecent and optimal solutions for $n \leq 50$ system.
2. From Table (12), notice that the performance (Eff. values, BV, AAE and MAE) of NN-BP with DR is best than NN-BP without DR and the same for PSO for $n<20$ for (P_1) and problem (P_2), while PSO is best than NN-BP without DR for $20 \leq n \leq 50$.
3. From Table (13), notice that the performance (Eff. values, BV, AAE and MAE) of NN-BP with DR is best than NN-BP without DR and PSO for $n<20$ for problem (P_2), while PSO is best than NN-BP without DR for $20 \leq n \leq 50$.
4. According to the conclusions mentioned in (2) and (3) above, the resaon behined the good performance of NN-PSO learning method compared with others, that is the swarm behaivour of this method since it using population of particles each considred as collection of weights for NN and we try to find best collection.
5. As suggetions for futute work, NN can be used for multi machines with multi objective functions.

References

[1]. Ruey-Maw Chen and Yueh-Min Huang, Competitive neural network to solve scheduling problems, *ELSEVIER, Neurocomputing 37*, 2001, 177-196.

[2]. B. Clow, A comparison of neural network training methods for character recognition, Department of Computer Science Carleton University, 95.495, 2003.

[3]. T. T. Willems and J. E. Rooda, Neural networks for job-shop scheduling, Control Engineering Practice *vol.2, no.1* 1994, p.31-39.

[4]. Y. P. S. Foo and Y. Takefuji, Integer linear programming neural networks for job-shop scheduling, *IEEE, International Conference on Neural Networks, Vol. 2*, 1998, pp. 341-348.

[5]. A. Hamad, B. Sanugi and S. Salleh, Single machine common due date scheduling problems using neural network, *Journal Teknologi, 36(C): Universiti Teknologi Malaysia*, 2002, 75–82.

[6]. A. Muralidhar and T. Alwarsamy, Multi-objective optimization of parallel machine scheduling using neural networks, *International Journal of Latest Trends in Engineering and Technology (IJLTET), Vol. 2, Issue 2*, March 2013.

[7]. B. S. P. Reddy and C. S. P. Rao, A hybrid multi-objective GA for simultaneous scheduling of machines and AGVS in FMS, *International journal Advance Manufacturing. Technology 31*, 2006, 602–613.

[8]. Bo Chen and C. N. Potts, *Complexity, algorithms and approximability*, Handbook of Combinatorial Optimization, 1998.

[9]. J. A. Hoogeveen, Invited review multicriteria scheduling, *European Journal of Operation Research 167*, 2005, 592-623.

[10]. B. Kolman, *Introductory linear algebra with applications*, Macmillan Publishing company, 1988.

[11]. Huajin Tang, Kay Chen Tan and Zhang Yi, *Neural networks: computational models and applications*, Springer-Verlag Berlin Heidelberg, 2007.

[12]. J. M. Zurada, *Introduction to artificial neural systems***,** West Publishing Company, 1992.

[13]. R. A. Kadhum, *Image noisy reduction using neural network*, M.Sc. thesis, Baghdad University, 2004.

[14]. Pomeroy P., An introduction to particle swarm optimization, Article, Mar, www.adaptiveview.com, 2003, p.1-7.

[15]. Y. Shi, *Particle swarm optimization*, Electronic Data Systems, Inc. Kokomo, IN 46902, USA Feature Article, IEEE Neural Networks Society, February 2004.

[16]. A. El-Bourni, S. Balakrishnan and N. Popplewell, Sequencing jobs on a single machine: a neural network approach, *Euro. Journal of Op. Res. 126:* 2000, 474 – 490.