International Journal of Research in Advanced Engineering and Technology
Online ISSN: 2455-0876
www.engineersjournals.com
Volume 1; Issue 2; November 2015; Page No. 53-64

Intelligent neural network with greedy alignment for Job-Shop scheduling

Fatin I Telchy

Control and Systems Department/Computer Branch University of Technology Baghdad Iraq.

Abstract
Job-Shop Scheduling (JSS) processes have highly complex structure in terms of many criteria. Because there is no limitation in the number of the process and there are many alternative scheduling. In JSS, each order that is processed on different machines has its own process and process order. It is very important to put these processes into a sequence according to a certain order. In addition, some constraints must be considered in order to obtain the appropriate tables.
In this paper, a 3-layers Feed Forward Backpropagation Neural Network (*FFBNN*) has been used for two different purposes, the first one task is to obtain the priority and the second one role is to determine the starting order of each operation within a job. Precedence order of operations indicates the dependency of subtasks within a job, Furthermore, the combined greedy procedure along with the back propagation algorithm will align operations of each job until best solution is obtained. In particular, greedy type algorithm will not always find the optimal solution. However, adding a predefined alignment dataset along with the greedy procedure result in optimal solutions.

Keywords: Scheduling Techniques, Job-Shop, Feed Forward Neural Network (*FFNN*), Greedy Alignment, Priority, Job Queue.

Introduction

Job Shop Scheduler

Traditional job-shop scheduling belongs to a large class of Nondeterministic Polynomial time complete (*NP*-complete) problems [1]. Because of the NP-complete characteristic of job-shop scheduling, it is difficult to find an optimal solution.
Some of the scheduling problems go to *NP*-Hard problem class. *JSSP* have highly complex structure in terms of many criteria. Because there is no limitation in the number of the process and there are many alternative scheduling. In JSS, each order that is processed on different machines has its own process and process order. It is very important to put these processes into a sequence according to a certain order. In addition, some constraints must be considered in order to obtain the appropriate tables [2].
Production scheduling is allocation of resources overtime to perform a collection of tasks. Of all kinds of production scheduling problems, the *JSSP* is one of the most complicated and typical. It aims to allocate m machines to perform n jobs in order to optimize certain criterion [3].
Anilkumar and Tanprasert (2006) [4] described the design and implementation of a NN-based job priority assigner system for a JSS environment. It was concluded that a back propagation neural network-based priority procedure would recognize jobs from a job queue by estimating each job's priority. This method provided 'rule less' solution environment for an application which is the greatest advantage of the NN in such cases.
Anilkumar and Tanprasert (2006) [3] presented a Feed-Forward Neural Network (*FFNN*), together with an alignment algorithm to solve a Generalized Job-Shop Problem (*GJSSP*), The trained NN had been embedded with predefined criteria which is relevant to determine starting time of various operations. The proposed scheduling approach is mainly used to analyze the performance of the NN in a GJSSP environment. Simulations of the proposed scheduler have shown that the NN with the alignment algorithm approach is efficient with respect to the quality of expected solutions and the solving speed but the NP-complete characteristic of JSS makes it difficult to reach an optimal solution level.
Anilkumar and Tanprasert (2007) [5] described a generalized JSS using a 3-layer *FFNN* and a greedy alignment procedure. The *NN* is used to detect precedence order of operations within each job which is humanly subjective in nature. The greedy alignment procedure aligns operations of various jobs on respective machines with feasible Finishing Time (*FT*).the problem of achieving the most feasible schedules in the case of a *GJSSP* which has n independent jobs and m machines and each job has *j* fixed operations has been achieved.
This Work is an extended version for previous work [6], where the proposed scheduler is used only to amylase the performance of *NN* in Generalized Job Shop problem environment, where it was concluded that the performance of *NN* is optimal with respected with the embedded information and the given dataset. However further analysis of using the Greedy Algorithm and comparison of the application of the Greedy with and without *NN* are required as will be illustrated in details in the paper.

Description of the Scheduler

Traditionally, the job-shop scheduling problem can be stated as follows [7]: given n jobs to be processed on m machines in a prescribed order under certain restrictive assumptions. The objective of JSS is to optimally arrange the processing order and the start times of operations under optimized certain criteria. In general, there are two types of constraints; the precedence between the operations of a job should be guaranteed, this is a precedence constraint. The second type of constraint is that no more than one job can be performed on a machine at the same time, this is a resource constraint. A *JSSP* is completely solved if the starting times of all operations are determined, and the precedence and resource constraints are not violated.

The concept of the generalized job-shop environment with independents jobs and their operations scheduled on a set of machines is shown in Figure 1.

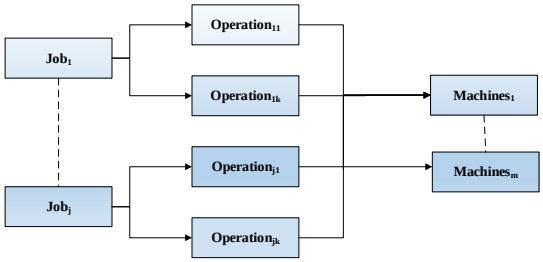

Fig 1: A generalized Job Shop Environment with n jobs and m Machines [3]

Job Shop Complexity

A job passes through a sequence of work centers as specified in its routing and it may wait for the required resources at those work centers. The total waiting time of the job in the entire process usually constitutes a major part of production lead time. This undesirable time is usually large, particularly for job shops with high-mix, low-volume production. It is not easy to measure the total job waiting time in such shops because:

a) Jobs with diverse routings are processed simultaneously.
b) The process time of an operation of a job may vary with both job and work center.
c) Product mix keeps changing frequently.
d) Resources have limited capacity.

This complexity makes it difficult to accurately predict job progress on shop floor, Work In Progress (WIP) level at each work center, bottleneck formations, resource utilization, shop throughput and job completion times. The bottlenecks may keep moving across work centers due to the changing product mix. In job shops, it is not easy to do preemptive capacity planning for preventing bottleneck creations and for improving the workflow, production lead times, on-time delivery and shop performance.
Three main reasons for JS complexity are:

a) The unpredictability of the nature and receiving time of customer orders.
b) The loading of a job only after receiving a client order and the required material.
c) The simultaneous production of diverse, low-quantity jobs using shared resources of finite capacity.

Many JS hit due dates for customer orders, whenever allowed by the customers, based on some average lead times (for example, three weeks), regardless of the existing situation. Similarly, they fix production start times based on due dates and average lead times. Small and mid-sized JS that work with low capital and receive diverse, low-volume orders with poor predictability may not be able to maintain inventory of raw materials and finished goods for many parts. Relatively, material requisition is made only after accepting such an order and production starts only after receiving the material. Since material inventories have a significant impact on production cost in JS, material for any job must be received just before the scheduled start time of the job. Actually, material requirements planning and the production schedule must be synchronized with each other [7].

Intelligent Job-Shop Scheduler Structure

The scheduler is designed to utilize all machines operate in parallel to maximize scheduling efficiency. That is passing all waiting operations one by one to available machines without violating their order. Scheduler concept used in this research is shown in Figure 2.
Certain notations have been used to formulate a *GJSSP*:
$J = (1,\ldots,j)$ is the job set.
O_{ik} = Operation k of job i.
D_i = Deadline or Due date.
P_i = Priority or Critical type of operation O_{ik}
S_i = Start time of O_{ik}
C_i = Processing time.
FT = Finishing Time

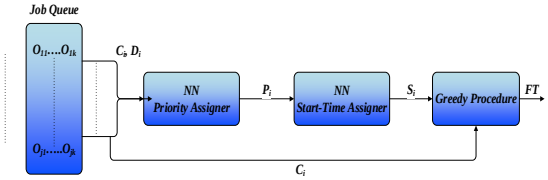

Fig 2: The Proposed Scheduler Structure

A job queue has many operations related to various jobs. But each job holds a fixed set of operations. Attributes such as D_i and C_i of each operation are submitted to the first *NN* to get priority (P_i) for each operations on a real time basis then again these attributes in addition to the priority of each operation are submitted to the second *NN* to get start time orders of operations on a real time basis. The greedy alignment procedure includes a predefined alignment dataset which helps the scheduler to generate various possible alignment combinations before establishing final result which must be the best schedule with minimum Finishing Time (*FT*) or complete time. Moreover, the alignment procedure is based on S_i and P_i.
Two Feed-Forward Neural Networks (*FFNN*) have been developed to solve the scheduling problem. The first *FFNN* is used to determine the priority Pi and the second *FFNN* is used to determine Si. The starting time for various operations will be determined using *FFNN*. Then, these operations will be scheduled by greedy algorithm. The greedy alignment procedure aligns operations of various jobs on respective machines with feasible / optimal *FT*. Each operation within a job is predefined priority by another *FFNN*, which is trained to recognize jobs from a job queue to estimate each job's priority. After testing with various possible network topologies, it is found that a three layer *NN* with 20 neurons is a suitable one as shown in Figure 3 and Figure 4.
Figure 3 and Figure 4 show the first *FFNN* structure which consist of 3 layers (input layer with 2 inputs "D_i and C_i", one hidden layer with 20 neurons, and output layer with one output "P_i") this *FFNN* acts as task priority assigner P_i.

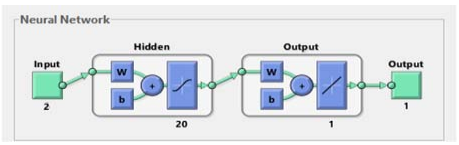

Fig 3: First *FFNN* Structure

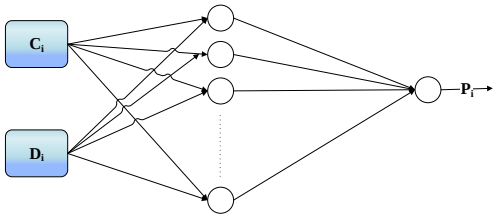

Fig 4: The First *FFNN* that Determines P_i

Figure 5 and Figure 6 show the second *FFNN* structure which consist of 3 layers (input layer with 3 inputs "D_i, C_i, and P_i", one hidden layer with 20 neurons, and output layer with one output "S_i") this *FFNN* acts as task starting time assigner S_i.

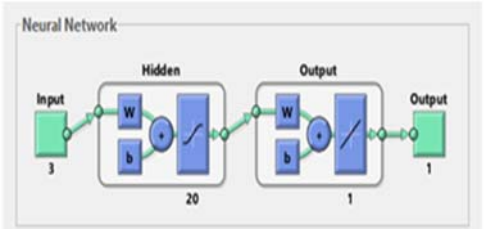

Fig 5: Second *FFNN* Structure

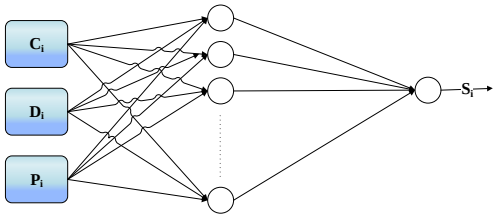

Fig 6: The Second *FFNN* that Determines S_i

Training Conditions

For the initial training of the *NN*, four numerical values have been introduced with their linguistic terms. The four inputs with initial training values used in the *NNs* are given in Table 1.

Table 1: Initial Training Data Conditions

C_i Execution/ Processing Time	**0.1 (Very Less)**	**0.3 (Less)**	**0.5 (not More)**	**0.7 (More)**	**0.9 (most)**
D_i Deadline of Operation	0.1 (Very Near)	0.3 (Near)	0.5 (not Far)	0.7 (Far)	0.9 (Very Far)
P_i Critical Type or Priority	0.1 (Very Simple)	0.3 (Simple)	0.5 (not Critical)	0.7 (Critical)	0.9 (Very Critical)
S_i Start Time	0.01 (Very Earlier Release			0.99 (Very Late Release)	

Variable of the *NN* must follow the given subjective criteria:

- A very critical/critical critical_type operation must hold very near/near start time.
- A very critical critical_type operation with near/very near deadline and with less/very less processing time must start first.
- Earliest setup of a very critical critical type operation is always earlier/very earlier.
- A simple/very simple critical type operation always keeps far/very far deadline and not earlier/very late earliest setup. Such operations will be released late.
- An operation with late/ very late earliest setup and with very simple critical type can release later. An operation with early/very early earliest setup, critical/very critical
- Critical type, and not more duration, then it must release near to its earliest setup.
- A very late earliest setup operation with not critical critical type with far deadline and with less/very less duration can be released very late.
- An operation with not critical critical type and with late/very late earliest setup and with less/very less duration must be released near to its earliest setup.
- A very critical critical type operation with very earliest setup and with very near deadline and with not more duration must be released first.

A. Assumptions

Based on the assumption, datasets have been created based on Table 1 criteria and normalized onto the interval [0, 1]. Normalizing original sample data can avoid saturations of neurons and speed the convergence of the neural network.

Performance of Scheduling Procedure

The optimality of the scheduler can be expressed by the given Theorem [5]: Every feasible schedule has FT not earlier than the time$\left(\frac{\sum_{i=1}^{j} C_i}{m}\right)$,

Where C_i is processing time and m is number of machines. That is a schedule with FT and with m machines can use a total of at most m. Note that FT is any given time units. Therefore the total time require for all jobs is $\sum_{i=1}^{j} C_i$ time units. Hence, FT>$\left(\frac{\sum_{i=1}^{j} C_i}{m}\right)$, is a feasible schedule. If any schedule has FT=$\left(\frac{\sum_{i=1}^{j} C_i}{m}\right)$, then that schedule is an optimal one.

Let X =$\left(\frac{\sum_{i=1}^{j} C_i}{m}\right)$, then Relative Error (*RE*) of a schedule is $\frac{FT-X}{X}$, for optimal schedules RE is always zero [8].

Scheduling Procedure

Scheduling procedure is developed in this paper to test and generate possible schedules as shown in the chart of Figure7. The purpose is to achieve optimal / nearly optimal results with various jobs size and machines. Details of the procedure used are given below:

1) Given dataset, Back propagation algorithm is used to train the *NN* to get the precedence order of all operations from the given operation attributes.
2) Sort out the release order of operations in order to get their precedence order.
3) The alignment procedure align various operations on given machines (job size > machine size). Predefined alignment dataset helps to align operations until to acquire best schedule without violating the operations order.

4) Transfer each operation result to buffer unless the machines are going to handle the identical job operations.
5) From the various alignments the procedure returns best schedule.

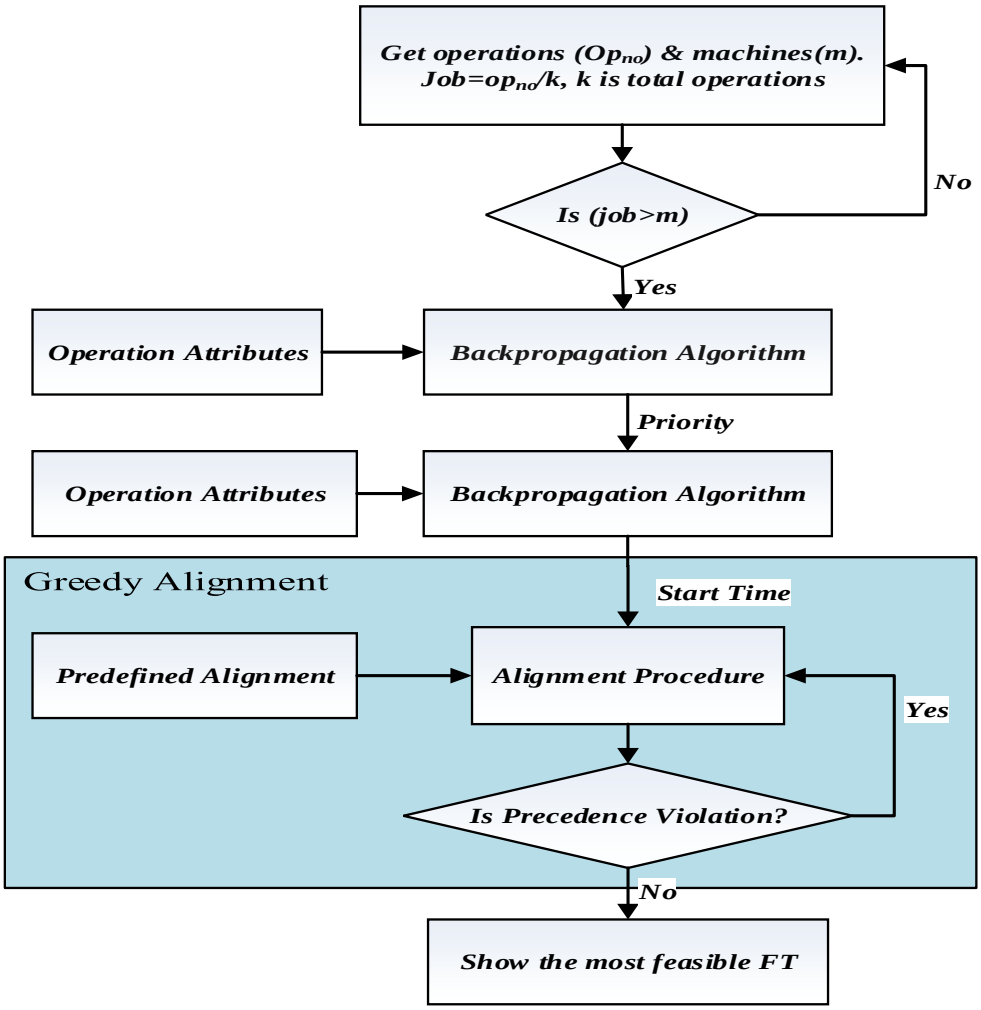

Fig 7: Proposed Job Shop Scheduling Procedure

Simulation and Results

As shown in Figure 2, two *FFNN* has been employed to do 2 different tasks, the following algorithm describes the applied steps of the intelligent *JSS*:

1) Train first *FFNN* to act as priority assigner, supervised training data has been created according to closest deadline then processing time, as shown in Figure 3.
2) Train second *FFNN* to act as start-time assigner, prepare the supervised training data according to conditions mentioned in Section 3. Figure 5 and 6 show the *FFNN*.
3) Apply Greedy algorithm to the prepared data for scheduling and optimization.

The proposed scheduler is written in *Matlab 2012*. Detailed description of the simulations will be discussed in the following cases.

Case 1: Scheduling 3 Jobs each with 3 Operations on 3 Machines

Three jobs each with three operation will be scheduled using three machines. The given information about each job can be summarized in Table 2, where the *D* represents the deadline, while the *C* represents the processing time.

Table 2: Case Study of 3 Machines, 3 Jobs each 3 Operations

Job/Operations	*D*	*C*
1	0.84	0.30
2	0.84	0.30
3	0.84	0.30
4	0.84	0.30
5	1.20	0.30
6	1.09	0.30
7	0.65	0.16
8	1.30	0.40
9	0.95	0.05

Scheduling process will be implemented in three stages as follows:

Stage 1

The data from Table 2 is used as an input to the *NN* priority assigner of Figure 2 in order to get the priority *P* for each operation, as shown in Table 3.

Table 3: Results of Stage 1

Job	*D*		*C*	*P*
1	0.84		0.30	0.3
2	0.84		0.30	0.3
3	0.84		0.30	0.3
4	0.84		0.30	0.3
5	1.20		0.30	0.3
6	1.09		0.30	0.3
7	0.65		0.16	0.9
8	1.30		0.40	0.1
9	0.95		0.05	0.5

Stage 2

The results from stage 1 is used an input to *NN* starting-time assigner of Figure 2 in order to get the start times for each operation as shown in Table4:

Table 4: Results of Stage 2

Job	**D**	**C**	**P**	**Estimated S**
1	0.84	0.30	0.3	0.54
2	0.84	0.30	0.3	0.19
3	0.84	0.30	0.3	0.19
4	0.84	0.30	0.3	0.19
5	1.20	0.30	0.3	0.61
6	1.09	0.30	0.3	0.57
7	0.65	0.16	0.9	0.10
8	1.30	0.40	0.1	0.51
9	0.95	0.05	0.5	0.9

Stage 3

By applying Greedy Algorithm, as described in the chart shown in Figure 7 on data in Table 4. The resulted scheduling process described by the tree terminology shown in Figure 8

Note: Comp. Time = (*S*) + (*C*)

Table 5: Results of Stage 3 with Optimal Solution

Job	*D*	*C*	*P*	New *S*	Mach.	Comp. Time
1	0.84	0.30	0.3	0.3	3	0.6
2	0.84	0.30	0.3	0	2	0.3
3	0.84	0.30	0.3	0	3	0.3
4	0.84	0.30	0.3	0.3	2	0.6
5	1.20	0.30	0.3	0.56	1	0.86
6	1.09	0.30	0.3	0.6	3	0.9
7	0.65	0.16	0.9	0	1	0.16
8	1.30	0.40	0.1	0.16	1	0.56
9	0.95	0.05	0.5	0.6	2	0.65

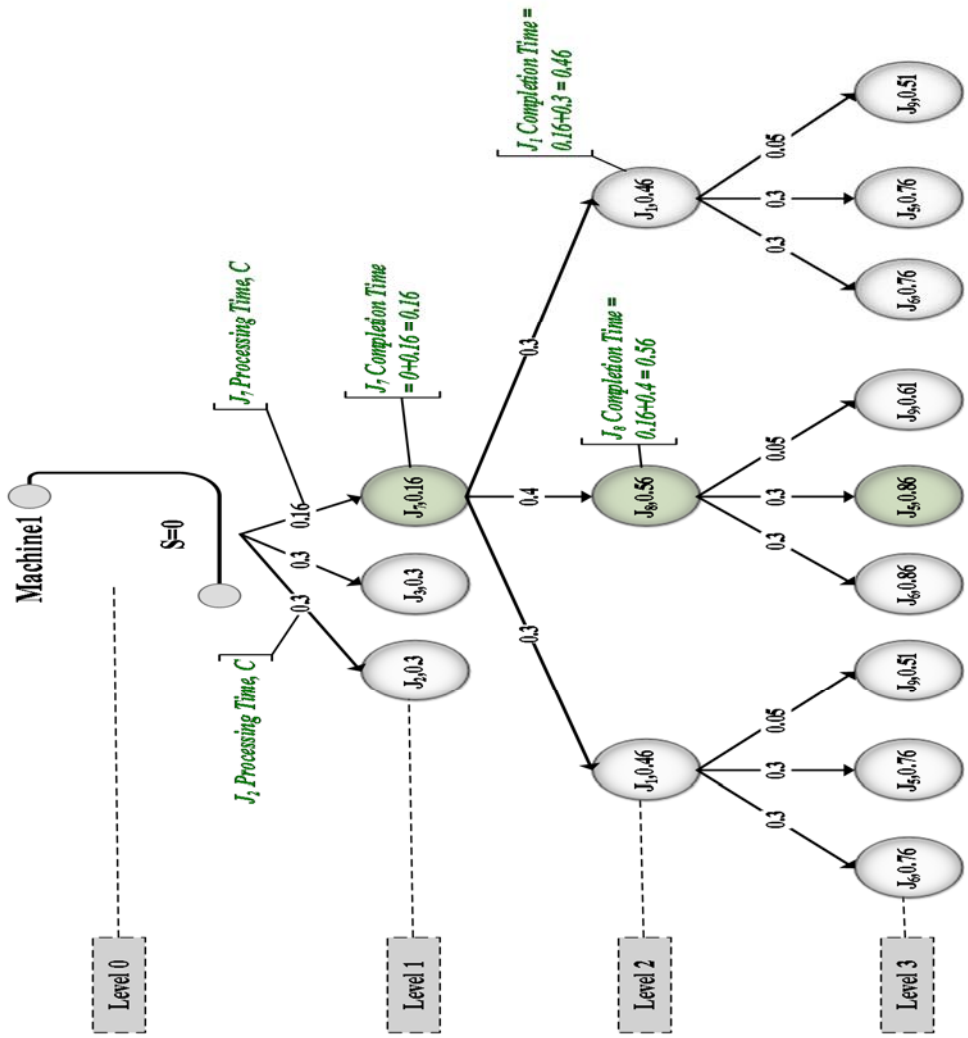

Fig 8 (a): Greedy Algorithm for 9 Operation 3 Machines

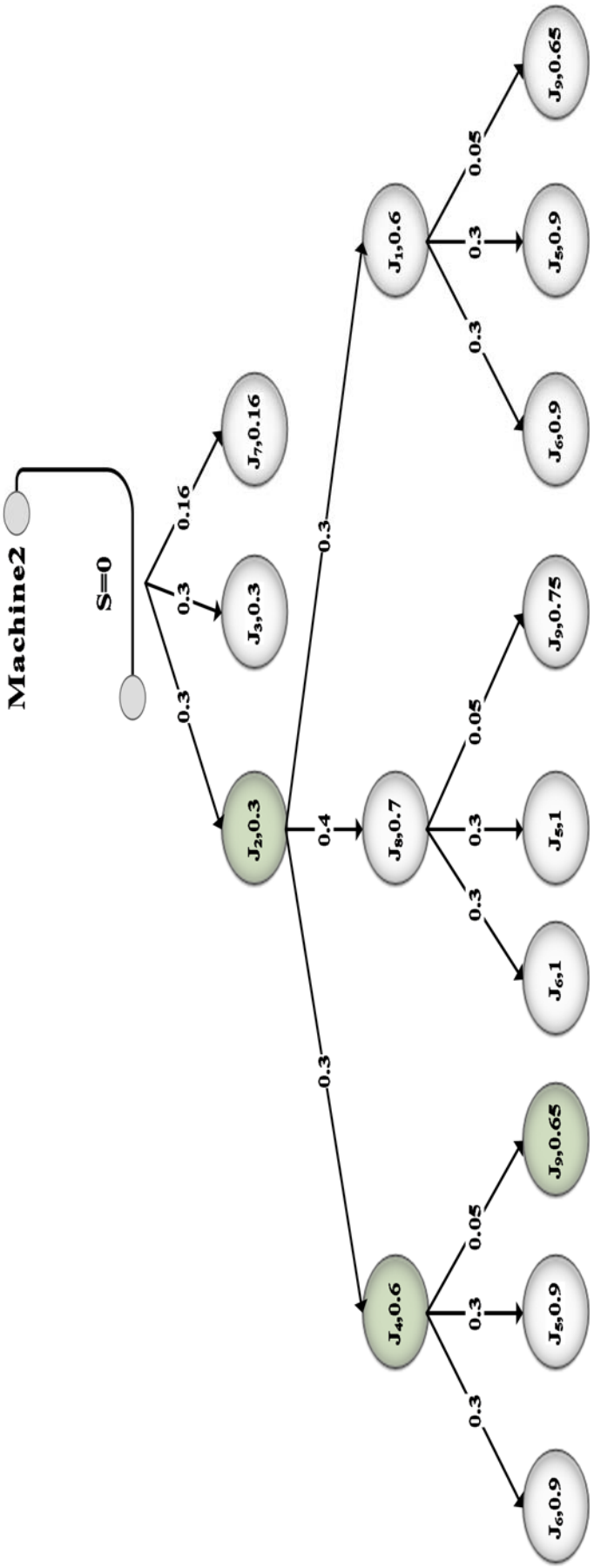

Fig 8: (b) Greedy Algorithm for 9 Operation, 3 Machines

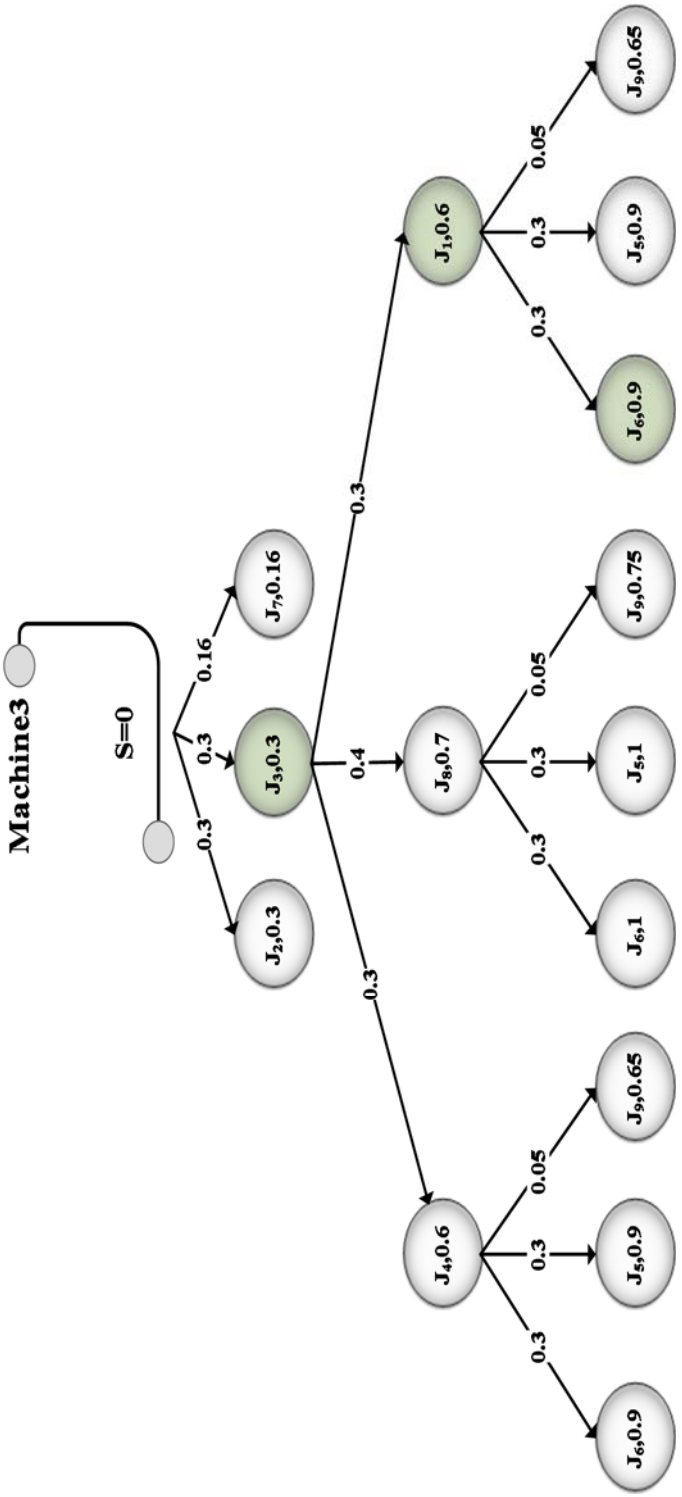

Fig 8: (c) Greedy Algorithm for 9 Operation, 3 Machines

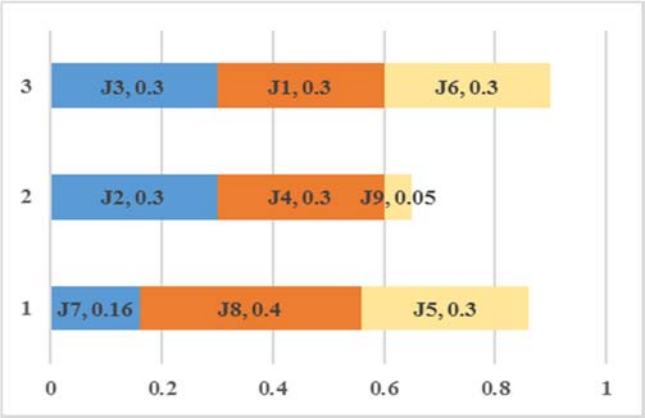

Fig 9: Optimal Scheduling of 3 Jobs each with 3 Operations with 3 Machines

According to Figure 9, it can be noticed that *FT*=0.9 greater than *X*= 0.81, nevertheless the scheduling can be considered optimal due to the presence of constraints.

Greedy Algorithm Scheduling, No Constraints & No NN Presence.

In this following, the applied Greedy algorithm will be described in details, Figure 10 shows the same problem stated in Section 5.1, but it solved by using Greedy Algorithm without constraints and *NN*.

- At level zero, as shown in Figure 10-(a), it required 9 operations to be uploaded to machine 1 to make proper selection of operation.
- After making the selection of jobs for each of 3 machines the remaining number of jobs = 9 – 3 = 6.
- In level one, 6 jobs will be uploaded for selection in similar way to that applied to level zero, the remain jobs 6 – 3 = 3, this will continue to the last level where the goal is reached

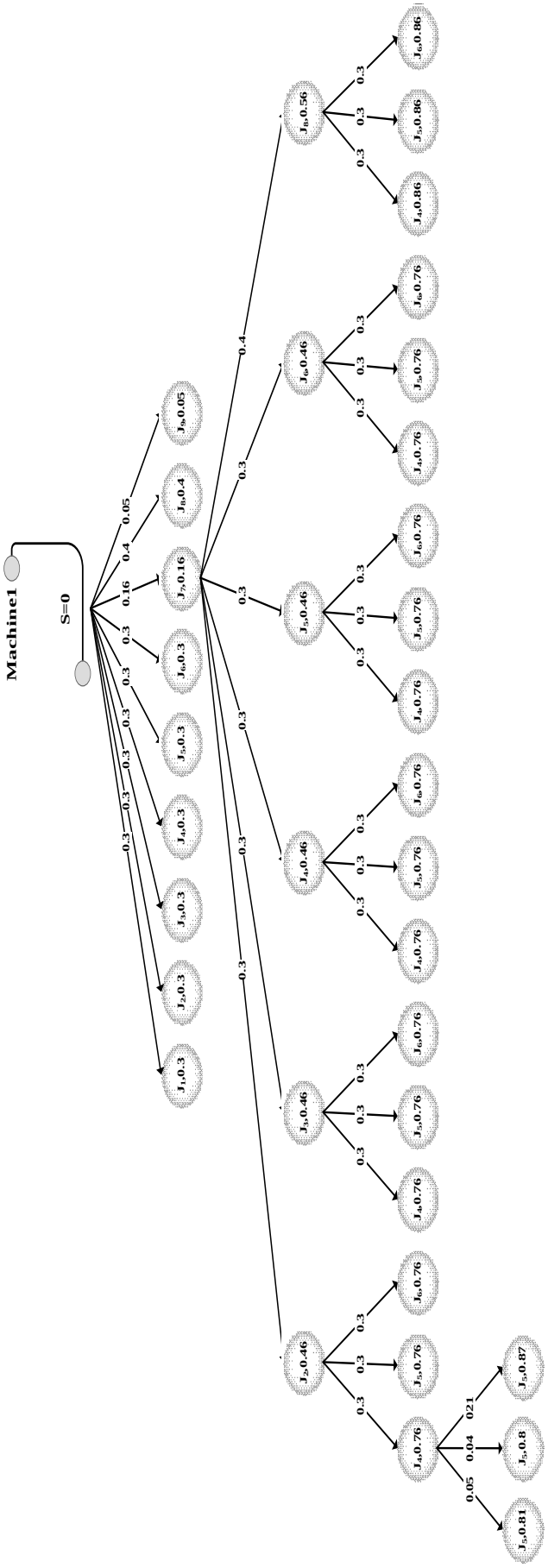

Fig 10 (a): Scheduling without Constraints for 3 Jobs each with 3 Operations with 3 Machines

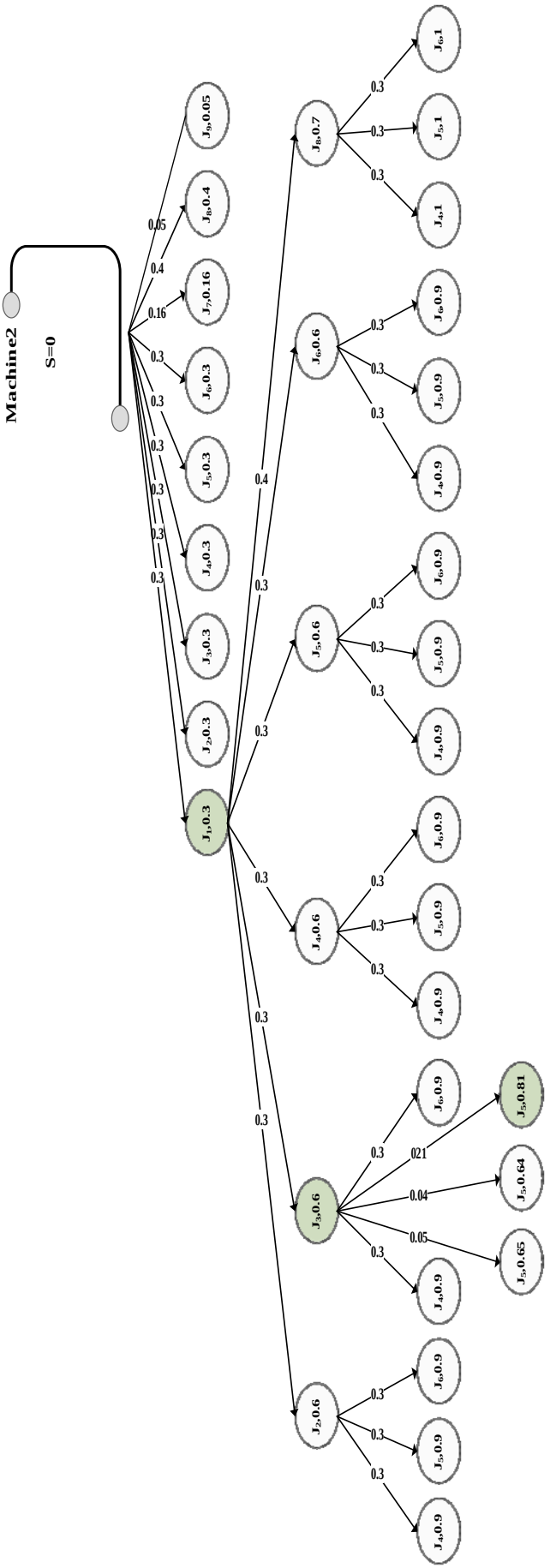

Fig 10 (b): Scheduling without Constraints for 3 Jobs each with 3 Operations with 3 Machines

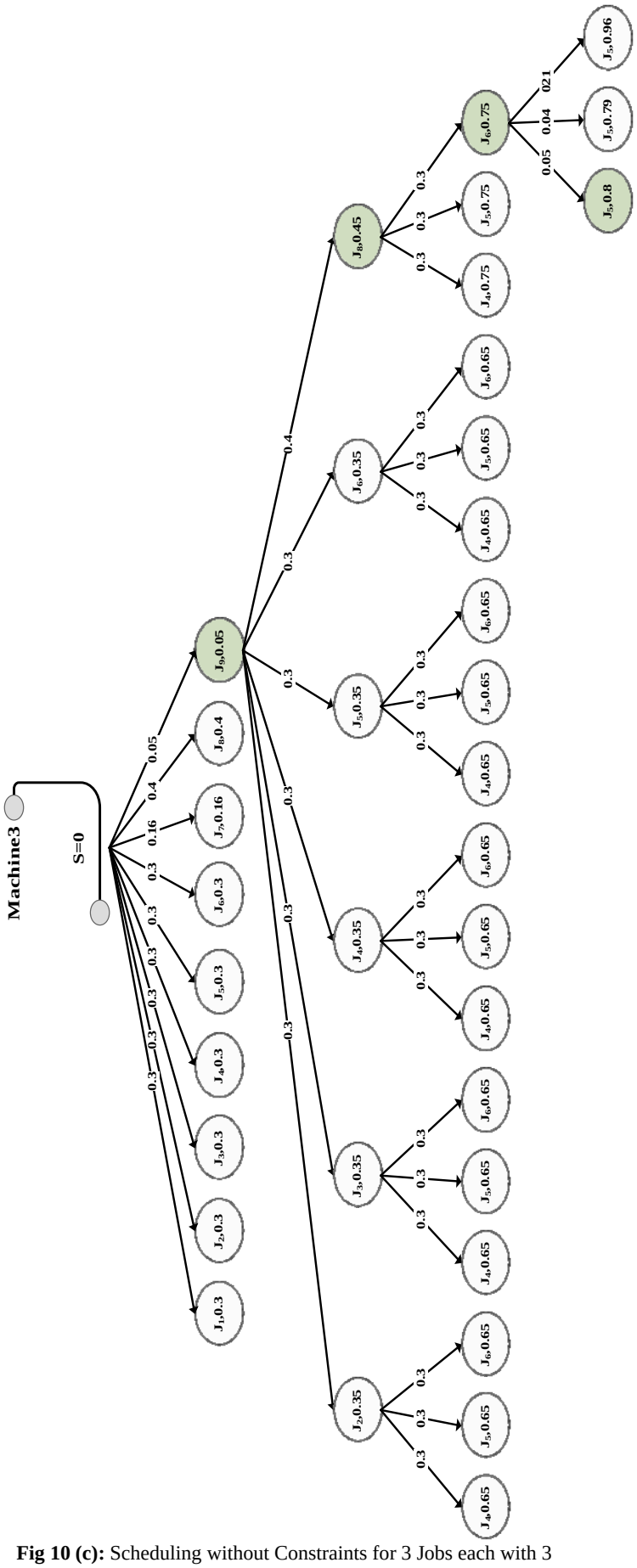

Fig 10 (c): Scheduling without Constraints for 3 Jobs each with 3 Operations with 3 Machines

Scheduling case 1 using Greedy algorithm only with no constraints "priority and non-preemption" (just to make the application of Greedy algorithm simple) is shown in Figure 10 which result in *FT*=0.8 that equal to performance index *X* and represents optimal scheduling according to Section 3 conditions. Although this solution seems to give optimal scheduling as in using the two proposed *NN*, but it does not take constraints in consideration which means the tasks will be divided between machines and this is impossible for our case study to divide the task on multi machines. So, to apply the same constraints in the greedy algorithm will result in a higher *FT* or performance index *X*. Therefore, to simplify the process for the Greedy algorithm to find the optimal solutions with constraints with less complexity and memory overhead, can be achieved by adding a predefined alignment dataset along with the Greedy algorithm.

Case 2: Scheduling 4 Jobs each with 2 Operations on 2 Machines

Another application for the proposed scheduler represents 8 operations (4 Jobs) to be scheduled using different number of machines; First application includes 2 machines, Second application includes 4 machines.

In a similar way to the procedure that applied in Case 1 in Section 5.1, the optimal schedule on 2 machines is obtained, as can be summarized in Table 6 and represented in Figure 12.

Figure 11 shows the scheduling by Greedy Algorithm under constraints obtained by the two proposed *NN* as mentioned in Case 1 stages from 1 to 3.

Table 6: Scheduling 4 Jobs each with 2 Operations with 2 Machines

Job	D	C	P	S	Comp. Time
1	0.6	0.5	0.8	0	0.5
2	0.6	0.3	0.9	0	0.3
3	0.8	0.4	0.5	0.5	0.9
4	0.8	0.3	0.6	0.5	0.8
5	0.7	0.4	0.7	0.8	1.2
6	0.9	0.3	0.3	0.8	1.1
7	0.9	0.3	0.2	1.2	1.5
8	0.8	0.6	0.4	1.1	1.7

According to the results shown in Figure 11 and Figure 12, and according to the theorem mentioned in Section 3.2, it can be noticed that *FT* equal to *X* value where it is equal to 1.6 where $X=\left(\frac{\sum_{i=1}^{j} c_i}{m}\right)$.

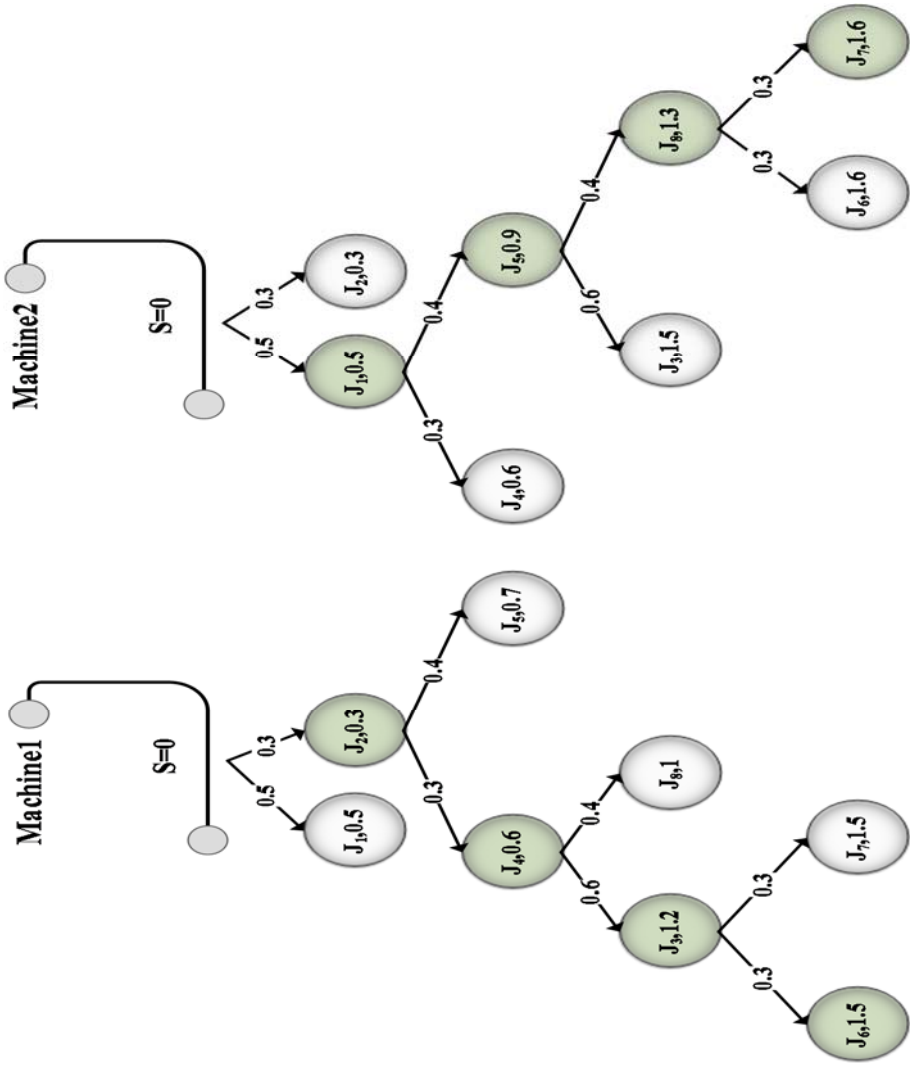

Fig 11: Optimal Scheduling of 4 Jobs each with 2 Operations on 2 Machines

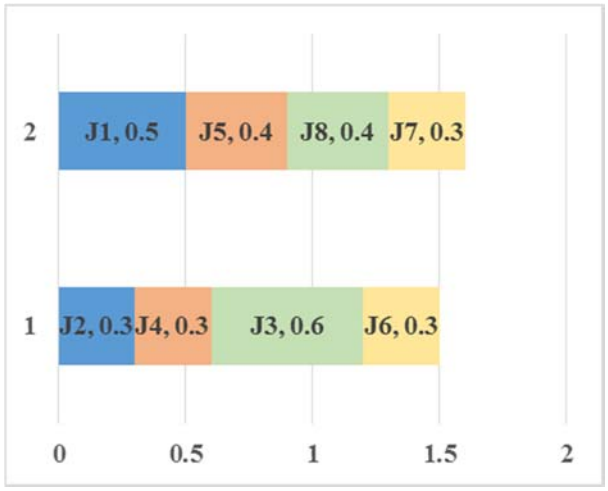

Fig 12: Optimal Scheduling of 4 Jobs each with 2 Operations with 2 Machines

Similar to the resolution that given in Section 5.1.4 for case 1. The Figure 13 shows the scheduling of case 2 without constraints of "priority and non-preemption", with FT=1.55 and equal to performance index X which represent the optimal. Although this solution seems to give the best optimal scheduling but since it is impossible to divide an operation to be processed on more than one machine in the same time, so practically, so, it is clear from this resolution and the resolution given in case 1 that using Greedy algorithm alone with no intelligence can provide feasible solution but it difficult to find optimal scheduling.

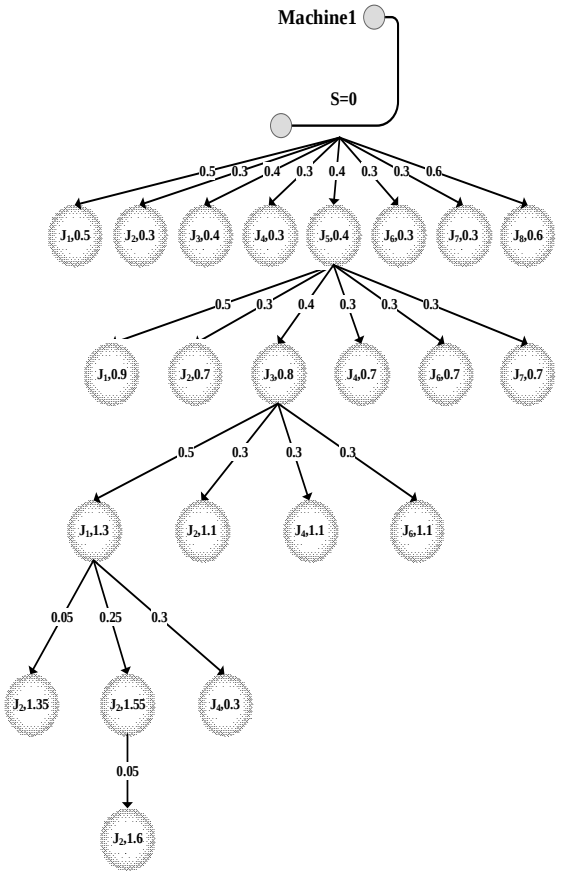

Fig 13 (a): Scheduling without Constraints for 4 Jobs each with 2 Operations with 2 Machines

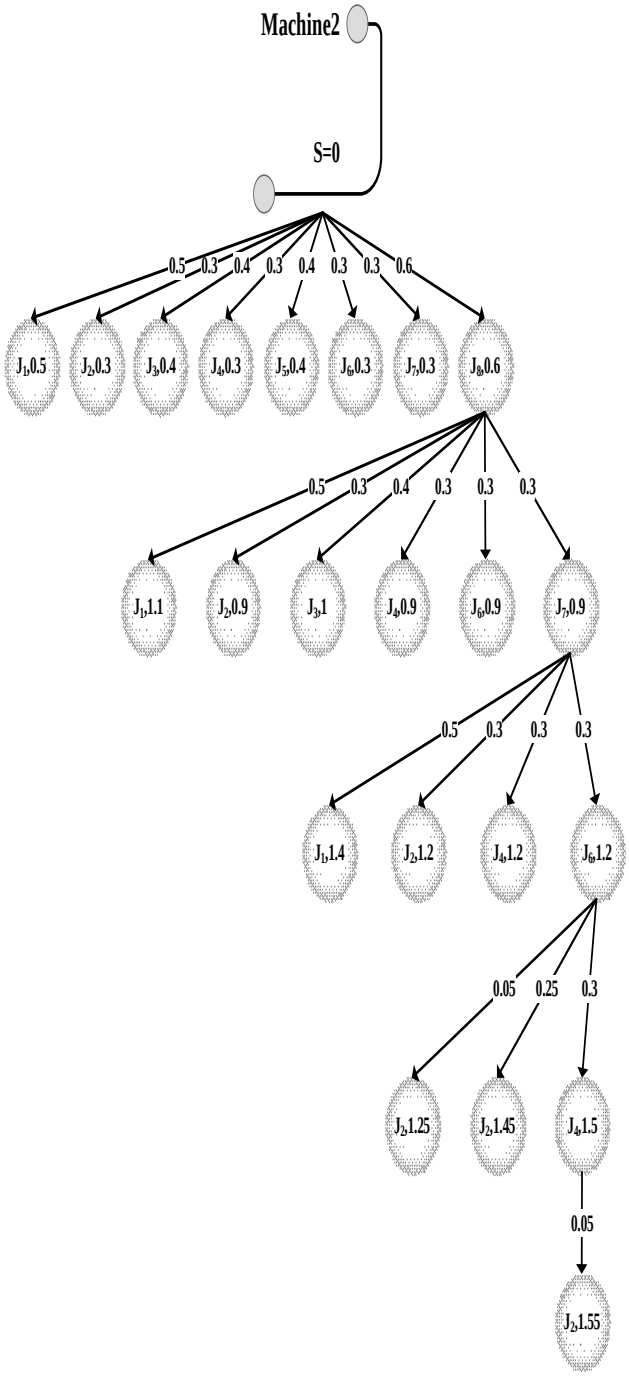

Fig 13 (b): Scheduling without Constraints for 4 Jobs each with 2 Operations with 2 Machines

The optimal schedule on 4 machines can be summarized in Table 7 and represented in Figure 15.

Table 7: Scheduling 4 Jobs each with 2 Operations with 4 Machines

Job	D	T	P	S	Comp. Time
1	0.6	0.5	0.8	0	0.5
2	0.6	0.3	0.9	0	0.3
3	0.8	0.4	0.5	0.3	0.7
4	0.8	0.3	0.6	0	0.3
5	0.7	0.4	0.7	0	0.4
6	0.9	0.3	0.3	0.5	0.8
7	0.9	0.3	0.2	0.3	0.6
8	0.8	0.6	0.4	0.8	1.4

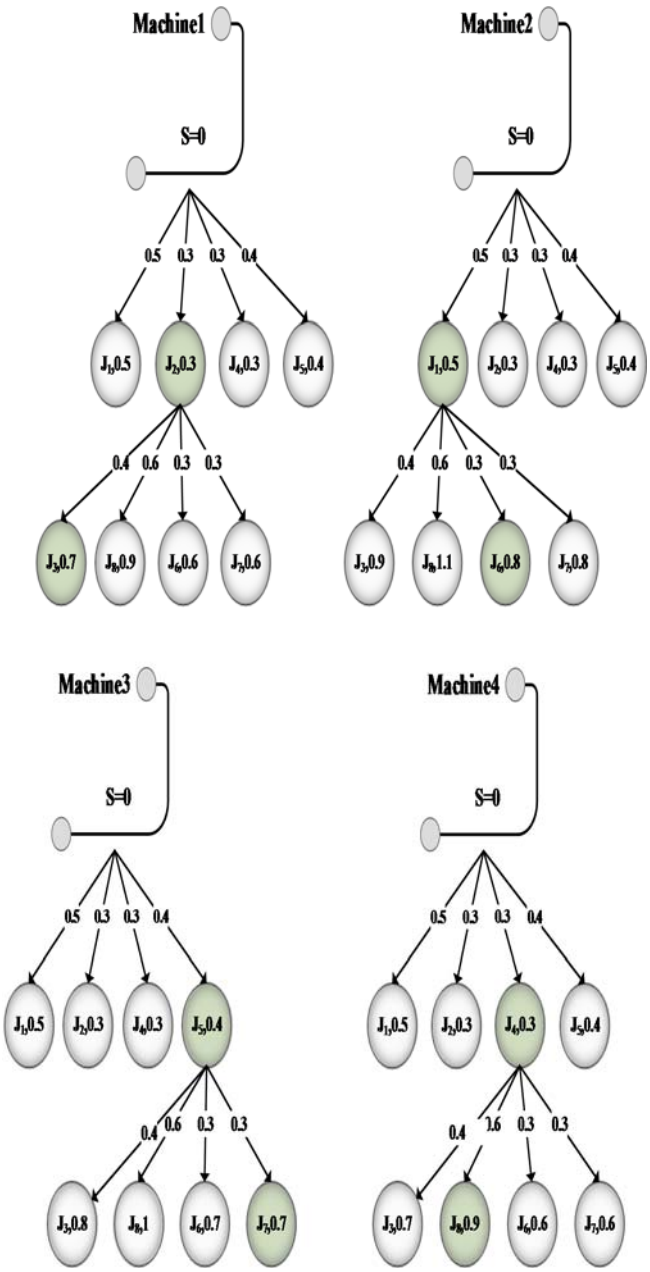

Fig 14: Optimal Scheduling of 8 Operations on 4Machines

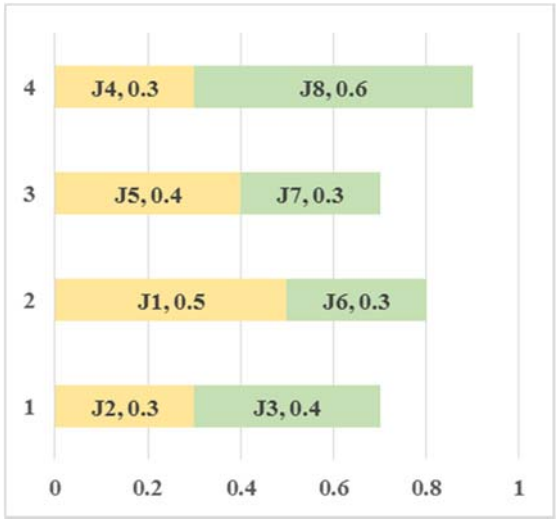

Fig 15: Optimal Scheduling of 8 Operations on 4 Machines

As shown in Figure 15 and similar to the results mentioned in Section 5.1 for the optimal results, it can be noticed that *FT* greater than *X* which is equal to 0.9, nevertheless the scheduling can be considered optimal due to the presence of constraints.

Conclusion

In this paper, two NN have been developed to accomplish the required scheduling; the first NN provides the priority for each job the second NN and estimates the start time of each operation, indicates the minimal amount of time to complete a *JSS*. The proposed scheduling problem of n jobs assigned to m machines with a multilayer *FFNN* is mainly used to enhance Greedy algorithm scheduling method results' and find the optimal scheduling. That will leads to enhance the performance of the *GJSSP* environment. This approach is characterized by:

Flexibility; it can be applied to any number of jobs and any number of machines.

Less memory usage; instead of uploading all number of job to memory the only number of jobs required is equal to the number of machines in each step level

Low overhead/Faster than applying Greedy algorithm without the proposed NN; as with the proposed NN the Greedy algorithm needs to go through less uploaded number of jobs than without it.

It is generic and the *FFNN* concept allows expansion in any of the two dimensions, i.e., it can readily be adopted for any number of jobs, and any number of machines. The *NP*-complete characteristic of *JSSP* makes it difficult to reach an optimal solution level. Nevertheless, after testing the scheduler it is found that the *NN* with proper embedded knowledge based on the designed *NN* job-shop with embedded knowledge base can help to detect operation sequence within a job. Combining the greedy alignment algorithm along with the *NN* will enables to depict the resulted schedule pattern without violating precedence and resource constraints. Simulation results proves that the performance of the *NN* is optimal with respect to the embedded information and the given data set.

From the overall observation, for a small size *JS* environment, the *NN* with proper embedded knowledge and with aid of the greedy alignment algorithm will surely lead to optimal solution.

References

1. Anany Levitin. Introduction to the Design and Analysis of Algorithms, 3rd Ed. New Jersey, USA Pearson, 2012.
2. Yılmaz ATAY, Halife KODAZ. Optimization of Job Shop Scheduling Problems Using Modified Clonal Selection, Selcuk University, Engineering Faculty, TURKEY, 2013.
3. Kothalil, Gopala Krishnan, Anilkumar, Thitipong Tanprasert. Neural Network Based Generalized Job-Shop Scheduler, Malaysia, 2006.
4. Kothalil, Gopala Krishnan, Anilkumar, Thitipong Tanprasert. Neural Network Based Priority Assignment for Job Scheduler, in http://www.academia.edu/2420131/Neural_Network_Based_Generalized_Job-Shop_Scheduler, Bankok, 2006, 181-186.
5. Kothalil, Gopala Krishnan, Anilkumar, Thitipong Tanprasert. Generalized Job-Shop Scheduler Using Feed Forward Neural Network and Greedy Alignment Procedure, in IASTED International Conference on Artificial Intelligence and Applications, part of the 25th

Multi-Conference on Applied Informatics, Innsbruck, Austria, 12-14, February 2007, 115-120.
6. Fatin Telchy, Safanah Rafaat. Intelligent Job Shop Scheduling using NN, in Zaytoonah University International Engineering Conference on Sustainability: Design and Innovation, Amman, Jordan, May 2014, 13-15.
7. Prasad Velaga. Optimal Solution for Prodction Scheduling, visited on Oct, 2012. http://www.optisol.biz/
8. Stinson, an Introduction to the Design and Analysis of Algorithm: Cambridge University Press, 1980.