Non-Greedy Heuristics and Augmented Neural Networks for the Open-Shop Scheduling Problem

Selcuk Colak, Anurag Agarwal

Department of Decision and Information Sciences, Warrington College of Business Administration, University of Florida, P.O. Box 117169, Gainesville, Florida 32611-7169

Received 22 June 2004; accepted 16 May 2005
DOI 10.1002/nav.20102
Published online 25 July 2005 in Wiley InterScience (www.interscience.wiley.com).

Abstract: In this paper we propose some non-greedy heuristics and develop an Augmented-Neural-Network (AugNN) formulation for solving the classical open-shop scheduling problem (OSSP). AugNN is a neural network based meta-heuristic approach that allows integration of domain-specific knowledge. The OSSP is framed as a neural network with multiple layers of jobs and machines. Input, output and activation functions are designed to enforce the problem constraints and embed known heuristics to generate a good feasible solution fast. Suitable learning strategies are applied to obtain better neighborhood solutions iteratively. The new heuristics and the AugNN formulation are tested on several benchmark problem instances in the literature and on some new problem instances generated in this study. The results are very competitive with other meta-heuristic approaches, both in terms of solution quality and computational times. © 2005 Wiley Periodicals, Inc. Naval Research Logistics 52: 631–644, 2005.

Keywords: scheduling; open shop; neural networks; heuristics

1. INTRODUCTION

The classical open shop scheduling problem (OSSP) is a well-known scheduling problem. It involves a collection of n jobs $J_1, J_2, \ldots, J_n$ and m machines $M_1, M_2, \ldots, M_m$. Each job J_i requires m operations $O_{i1}, O_{i2}, \ldots, O_{im}$ where O_{ij} has a processing time of $p_{ij} > 0$ and has to be processed on machine M_j. The operations in each job may be processed in any order. Each machine can process at most one operation at a time and each job can be processed by at most one machine at a time. Further, operations cannot be preempted; i.e., they must be processed without interruption. The objective is to obtain a feasible schedule with the minimum makespan. The OSSP, like many other scheduling problems, has received wide attention in the literature due to its applications in manufacturing. Algorithmic improvements in finding better solutions faster for these scheduling problems have cost-saving implications for many types of manufacturing activities.

We apply a new approach called augmented neural networks (AugNN) for solving the OSSP. Agarwal, Jacob, and Pirkul [1, 2] first applied the AugNN approach successfully to the task-scheduling problem. The approach is a hybrid of heuristics and neural-network approaches. It allows one to combine the advantages of heuristics approaches, i.e., finding good solutions fast, and adaptive learning approaches, i.e., finding better neighborhood solutions iteratively. The AugNN approach allows fast convergence, compared to other meta-heuristics such as genetic algorithms, simulated annealing, tabu search, and ant colonies.

We test the AugNN approach on some well-known existing heuristics for the OSSP and also explore several new heuristics to obtain very good results (optimal or near-optimal) in relatively few iterations. Our results are based on some benchmark problems of sizes 15×15 and 20×20 by Taillard [24] and 10×10 by Gueret and Prins [12], and also some new larger problems of sizes 25×25, 30×30, 50×50, and even 100×100 generated in this study.

This study makes several contributions. First, the AugNN formulation for the OSSP is developed for the first time. Second, a new greedy heuristic called DS/LTRPAM is developed and non-greedy versions of this and other greedy heuristics are proposed. The non-greedy versions of DS/LTRPAM gave the best results. Third, we apply new learning strategies not proposed earlier in the AugNN literature. Fourth, new benchmark problems of sizes 25×25, 30×30, 50×50, and 100×100 are generated. Except Alcaide, Sicilia, and Vigo [3], the other authors have restricted their

Correspondence to: A. Agarwal (aagarwal@ufl.edu) evaluations, for comparison purposes, to standard instances with up to 20 jobs–20 machines.

The rest of the paper is organized as follows. Section 2 presents a literature review for the OSSP. Section 3 describes the various heuristic dispatching rules used in this study. The AugNN formulation for the OSSP is explained in detail in Section 4. In Section 5, computational results are presented and discussed. Finally, Section 6 provides a summary of the paper and discusses future research ideas.

2. LITERATURE REVIEW

Gonzales and Sahni [10] presented a polynomial time algorithm for solving the OSSP for the case $m = 2$. They also showed that the non-preemptive OSSP is NP-hard for $m \geq 3$. Pinedo [21] proposed the longest alternate processing time first dispatching rule that solves the two-machine problem optimally. A branch-and-bound method for the general OSSP is developed by Brucker et al. [7]. Their method solved some benchmark problems in the literature to optimality for the first time. Gueret, Jussien, and Prins [11] improved the Brucker's algorithm by using an intelligent backtracking technique. Dorndorf, Pesch, and Phan-Huy [8] implemented another branch-and-bound algorithm based on constrained-propagation methods to reduce the search space. Branch-and-bound algorithms, in general, fail to provide good solutions for larger problem instances in reasonable time.

To overcome the computational complexity of finding optimal solutions, a variety of heuristics have been proposed for finding good suboptimal solutions fast. The literature is replete with such heuristics for all types of scheduling problems including OSSP. Gueret and Prins [13] presented two heuristics—(1) list-scheduling algorithm with two priorities and (2) based on the construction of matchings in a bipartite graph. Brasel, Tautenhahn, and Werner [6] implemented constructive insertion algorithms. Liaw [17] developed an iterative improvement approach based on Bender's decomposition. Meta-heuristic approaches such as tabu search, simulated annealing, genetic algorithms, and ant colonies have also been used for the OSSP. For example, Taillard [24] used tabu search to solve many open-shop problems to optimality. Taillard has also developed many randomly generated scheduling problems for the OSSP, which are used as benchmark problems in the literature. Alcaide, Sicilia, and Vigo [3] proposed tabu search algorithm as well, which uses simple list scheduling algorithms to build the starting solutions. They have tested the algorithm on instances up to 50 machines and 100 jobs. Another tabu search algorithm for the OSSP is applied by Liaw [18]. He also introduced a neighborhood-search algorithm using simulated annealing [19]. Several genetic algorithms have been proposed for the OSSP by Khuri and Miryala [16], Liaw [20], and Prins [22]. Blum [4, 5] developed ant-colony-optimization approaches for the OSSP.

Neural networks, besides being used for data mining tasks such as classification, clustering and pattern recognition, have been used for solving optimization problems as well. Hopfield and Tank [15] first applied neural networks to the traveling-salesman problem, a classic combinatorial optimization problem. The Hopfield and Tank approach was applied to the job-shop scheduling problems by Foo and Takefuji [9] and Sabuncuoglu and Gurgun [23]. While this approach worked for smaller problem instances (up to 5×5), it failed to provide good solutions in reasonable time, for larger problem instances such as (10×10). Agarwal, Jacob, and Pirkul [1, 2] proposed a different kind of approach for using neural networks, called the augmented-neural-network approach, for solving scheduling problems. Essentially, it is a hybrid of the heuristic and iterative learning approaches. In this approach, a scheduling problem is framed as a neural network, with layers of processing elements and connection weights, in a manner that allows one to integrate the problem specific knowledge with adaptive learning. It also allows a given heuristic to be embedded in the solution procedure. This approach converges very fast and gives near optimal solutions for even larger problems in reasonably small number of iterations.

Since we are embedding a heuristic in the AugNN approach, it is important to work with a good heuristic. We use some existing greedy heuristics and propose non-greedy variations of these heuristics. In addition we develop a new greedy heuristic and its non-greedy variations. We first describe these heuristics briefly in the next section before describing the AugNN formulation in the following section.

3. HEURISTICS

In this section we explain some heuristics to find feasible solutions for the OSSP in each iteration of the AugNN approach.

1. *Job TRPT Machine PT Rule:* If only one job is ready to start and machines are available for the unfinished operations of this job then the operation with the highest processing time (PT) is assigned to its machine. If no machines are available for this job, the job waits. If more than one job is ready to start, then the job with the highest total remaining processing time (TRPT) is considered first. TRPT is also known in the literature as MWR (Most Work Remaining).
2. *Job TRPT Machine PT Rule Non-Greedy:* This is our non-greedy variation of the above heuristic. It works the same as above except that, before making the assignment decision for the job with the

highest TRPT and the operation with the highest PT, we check whether the remaining slack on this job is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the job in question, we wait till the next operation is complete. Slack for a job is defined as the difference between the total processing time of the longest job and the job in question. So, the slack of the longest job is zero. Remaining slack for a job is the difference between the original slack and the slack consumed by the job.

3. *Job TRPT Machine RPT Rule:* If only one job is ready to start and machines are available for the unfinished operations of this job then the machine with the highest remaining processing time (RPT) is assigned to this job. If no machines are available for this job, then the job waits. If more than one job is ready to start, then the job with the highest total remaining processing time (TRPT) is considered first.
4. *Job TRPT Machine RPT Rule Non-Greedy:* This is our non-greedy variation of the above heuristic. It works the same as above except that before making the assignment decision for the job with the highest TRPT and the machine with highest RPT, we check whether the remaining slack on this job is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the job in question, we wait till the next operation is complete.
5. *DS/LTRP Rule:* This rule is used by Liaw [17–19]: Whenever a machine is idle, select among its available operations, the operation belonging to the job that has the longest total remaining processing time on other machines (LTRPOM) for this job (i.e., excluding the operation on the machine in question) to process next. If there is no available operation, the machine remains idle until the next operation has been completed on some other machine. If more than one machine is idle at the same time, select the machine with the longest remaining processing time on that machine.
6. *DS/LTRP Rule Non-Greedy with Job Slack:* This is our non-greedy variation of the above heuristic. It works the same as above except that before making the assignment decision for the job with LTRPOM and the machine with highest RPT, we check whether the remaining slack on this job is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the job in question, we wait till the next operation is complete.
7. *DS/LTRP Rule Non-Greedy with Machine Slack:* This is another non-greedy variation of the above heuristic. It works the same as the *DS/LTRP* heuristic except that before making the assignment decision for the job with LTRPOM and the machine with highest RPT, we check whether the remaining slack on this machine is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the machine in question, we wait till the next operation is complete. Slack for a machine is defined as the difference between the total processing time on longest processing time machine and machine in question. Remaining slack for a machine is the difference between the original slack for that machine and the slack consumed by it.
8. *DS/LTRPAM Rule:* Whenever a machine is idle; select, among its available operations, the operation belonging to the job that has the longest total remaining processing time (including the operation on the machine in question) to process next. If there is no available operation, the machine remains idle until the next operation has been completed on some other machine. If more than one machine is idle at the same time, select the machine with the longest remaining processing time on that machine. This is a new heuristic we are proposing in this study.
9. *DS/LTRPAM Rule Non-Greedy with Job Slack:* This heuristic is the same as above except that before making the assignment decision for the job with TRP and the machine with highest RPT, we check whether the remaining slack on this job is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the job in question, we wait till the next operation is complete.
10. *DS/LTRPAM Rule Non-Greedy with Machine Slack:* This heuristic is the same as *DS/LTRPAM* rule above except that before making the assignment decision for the job with TRP and the machine with highest RPT, we check whether the remaining slack on this machine is greater than the time the next operation will be complete. If the next operation is due to complete before the remaining slack of the machine in question, we wait till the next operation is complete.

We now explain the formulation of the AugNN approach for the OSSP in the following section.

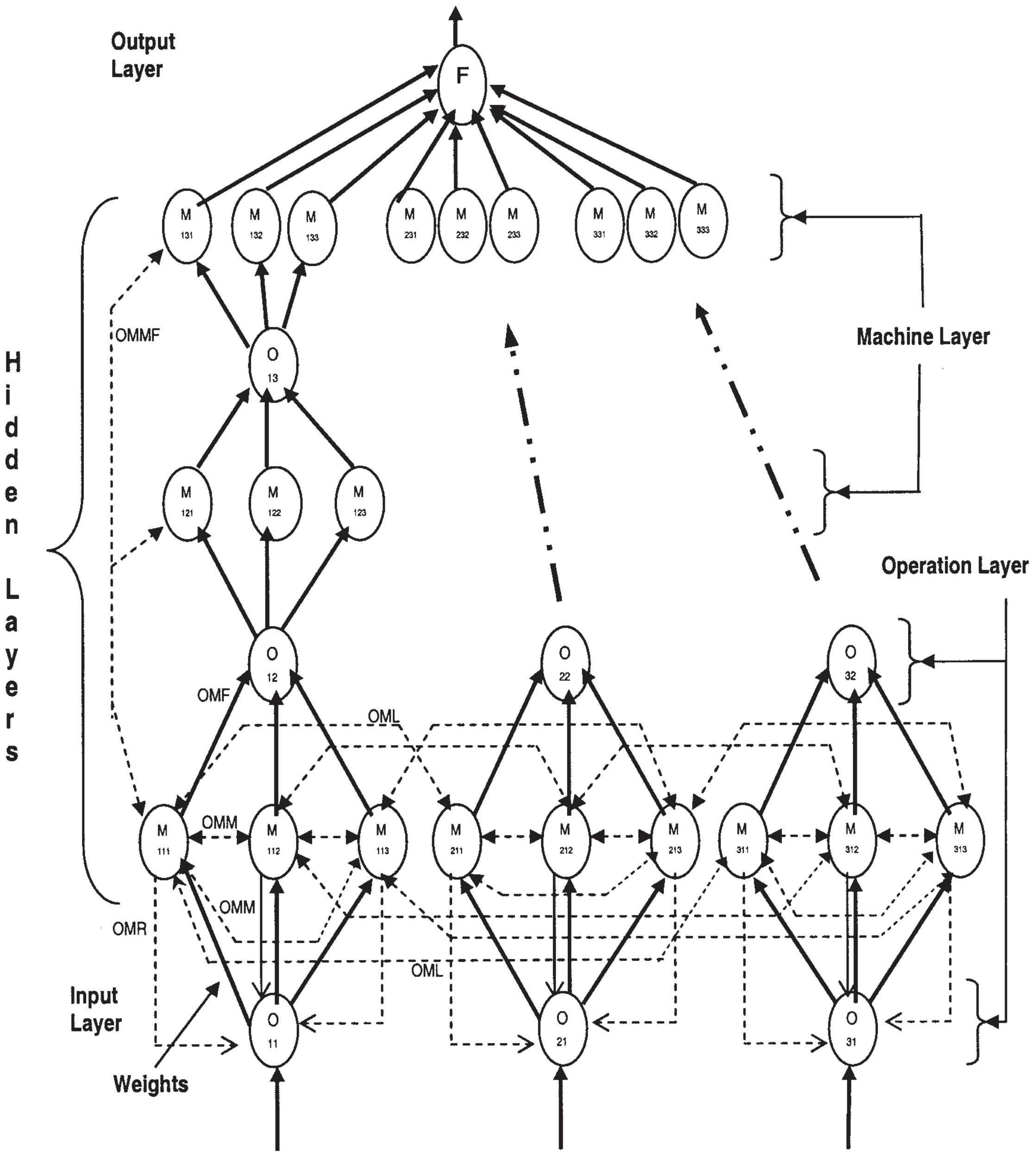

Figure 1. Neural network architecture for the AugNN approach for a 3 × 3 problem.

4. AUGMENTED NEURAL NETWORK FORMULATION

The general idea of the AugNN formulation is that we first convert a given OSSP into a neural network, with input layer, hidden layers, and output layer of neurons or processing elements (PEs). We define connections between these layers, characterized by weights. Input, activation, and output functions are designed for each node so as to enforce the constraints of the problem and embed a heuristic to generate a feasible solution in one pass (iteration) of the neural network. An iteration consists of calculating all the functions of the network from the input up to the output layer. We then apply a learning strategy to modify the weights on the connections such that learning takes place and better solutions are obtained in subsequent iterations.

We will now describe, with the help of an example, how to convert a given problem into a neural network. We will take a simple 3 × 3 problem for this purpose. Figure 1 shows the neural network architecture for this problem. In a 3 × 3 problem, there are 3 jobs, each with 3 operations, for a total of 9 operations (O_{11}, O_{12}, O_{13}, O_{21}, O_{22}, O_{23}, O_{31}, O_{32}, and O_{33}). We create three operation layers, corresponding to the three operations of each job. Each operation layer has three nodes corresponding to each job. Note that for a more general $n \times m$ case, there would be m operation layers, each with n nodes. Following each operation layer is a machine layer with 9 nodes each. Each of the three operation nodes is connected to three machine nodes for a total of 9 machine nodes. For a more general $n \times m$ case, there would be nm machine nodes in each machine layer. So, for example, operation O_{21} is connected to three machine nodes labeled M_{211}, M_{212}, and M_{213}. Operation layer 1 can be considered as the input layer of a neural network. The remaining operation layers and machine layers can be considered as hidden layers and a final node acts as the output layer. Connections between operation nodes and machine nodes are characterized by weights. These weights are all the same for the first iteration, but are modified for each subsequent iteration. There are also connections between machine nodes and subsequent operation nodes, which are not characterized by any weights. These connections serve to pass signals from one layer to the next to trigger some functions.

There are three types of connections between machine nodes. In the first type, each machine node is connected to nodes of other machines for the same operation. For example, M_{111} is connected to M_{112} and M_{113}. These connections are used to enforce the constraint that the same operation cannot be processed by more than one machine at the same time. In the second type, each machine node is connected to nodes of the same machine on other operations of other jobs. For example, M_{111} is connected to M_{211}, M_{311}, M_{221}, M_{231}, M_{321}, and M_{331}. These connections are used to enforce the constraint that the same machine cannot process more than one operation at the same time. In the third type, each machine node is connected to nodes of the same machine for other operations of the same job. So for example, M_{111} is connected to M_{121} and M_{131}. These connections are used to ensure that the same machine is not assigned to the same job more than once.

Machine nodes are also connected to the operation nodes in the reverse direction. Whenever a job is assigned to a machine, the machine node sends a signal to the operation node, indicating to the operation that it has been assigned. This signal changes the state of the operation node and triggers other functions.

We now describe the input, activation, and output functions for each node and the learning strategy for the weights. We will need the following notation to describe our functions:

N	Number of jobs
M	Number of machines
c	Current iteration
J	Set of jobs $= \{1, \ldots, n\}$
J_i	Job i, $i \in J$
M	Set of machines $= \{1, \ldots, m\}$
M_k	Machine k, $k \in M$
O	Set of operations $= \{1, \ldots, m\}$
O_{ij}	ijth operation node, $i \in J$, $j \in O$
M_{ijk}	Node for machine k, connected from O_{ij}, $i \in J$, $j \in O$, $k \in M$
TRP_j	Total Remaining Processing time for job j, $j \in J$
ω_{ik}	Weight on the link from O_{i*} to M_{i*k} machine nodes, $i \in J$, $k \in M$
ω_m	Weight on the links between machine nodes
α	Learning coefficient
ε_c	Error in iteration c
O_F	Final dummy operation node
ST_{ijk}	Start time of O_{ij} on M_k, $i \in J$, $j \in O$, $k \in M$
PT_{ik}	Processing time of J_i on M_k, $i \in J$, $k \in M$
Win_{ij}	Winning status of job J_i on machine M_j, $i \in J$, $j \in M$
$(\omega_{NG})_{ij}$	Weight for waiting for operation O_{ij}. NG stands for Non-Greedy
T	Elapsed time
JPT_i	Job Processing Time for job i = sum of PT of all operations for job i
$CJPT$	Critical Job Processing Time = Max of all JPT_i, $i \in J$
$JSLK_i$	Slack for job $i = CJPT - JPT_i$, $i \in J$
MPT_k	Machine Processing Time for machine k = Sum of PT of all operations on machine k
$CMPT$	Critical Machine Processing Time = Max of all MPT_k, $k \in M$
$MSLK_k$	Slack for machine $k = CMPT - MPT_k$, $k \in M$

Following are all functions of elapsed time t:

$IO_{ij}(t)$	Input function value of operation node O_{ij}, $i \in J$, $j \in O$
$IO_F(t)$	Input function value of operation node O_F
$IM_{ijk}(t)$	Input function value of machine node k from operation O_{ij}, $i \in J$, $j \in O$, $k \in M$
$OO_{ij}(t)$	Output function value of operation node O_{ij}, $i \in J$, $j \in O$
$OO_F(t)$	Output function value of operation node O_F
$\mathrm{OMF}_{ijkpq}(t)$	Output of machine node M_{ijk} to O_{pq} in the forward direction, i, $p \in J$, j, $q \in O$, $k \in M$, $q \neq m$

$OMF_{ijkF}(t)$	Output of machine node M_{ijk} to O_F in the forward direction, $i \in J$, $j = m$, $k \in M$
$OMR_{ijk}(t)$	Output of machine node M_{ijk} to O_{ij} in reverse direction, $i \in J$, $j \in O$, $k \in M$
$OML_{ijkpq}(t)$	Output of machine node M_{ijk} to M_{pqk} in lateral direction, $i, p \in J$, $j, q \in O$, $k \in M$, $i \neq p$
$OMM_{ijkp}(t)$	Output of machine node M_{ijk} to M_{ijp} in lateral direction, $k, p \in M$, $k \neq p$, $i \in J$, $j \in O$
$OMMF_{ijkp}(t)$	Output of machine node M_{ijk} to M_{ipk} in forward direction, $i \in J$; $p, j \in O$, $k \in M$, $p > j$
$\theta O_{ij}(t)$	Activation function of operation node O_{ij}, $i \in J$, $j \in O$
$\theta M_{ijk}(t)$	Activation function of machine node M_{ijk}, $i \in J$, $j \in O$, $k \in M$
$assign_{ijk}(t)$	Operation O_{ij} assigned to machine M_k
$S(t)$	Set of operations that can start at time t. $S(t) = \{O_{ij} \mid OO_{ij}(t) = 1\}$
$JSLKU_i(t)$	Slack of job i used up at time t, $i \in J$
$MSLKU_k(t)$	Slack of machine k used up at time t, $k \in M$
$RSJ_i(t)$	Remaining slack of job $i = JSLK_i - JSLKU_i(t)$, $i \in J$
$RSM_k(t)$	Remaining slack on machine $k = MSLK_k - MSLKU_k(t)$, $k \in M$

The neural network algorithm can be described with the help of the input functions, the activation functions, and the output functions for the operation nodes and the machines nodes and the learning strategy.

4.1. AugNN Functions

4.1.1. Operation Nodes

Input functions, activation states, and output functions are now explained for the operation nodes.

Input functions:

$$IO_{ij}(0) = 1 \qquad \forall\ i \in J, j = 1, \tag{1}$$

$$IO_{ij}(0) = 0 \qquad \forall\ i \in J, j \in O, j \neq 1, \tag{2}$$

$$IO_F(0) = 0. \tag{3}$$

These functions at time $t = 0$ provide initial signals to the operation layers. The first operation nodes (i.e., for $j = 1$) of all the jobs get a starting signal of 1 at time 0 [Eq. (1)]. The remaining operation layers get a signal of 0 [Eq. (2)] and the final output layer also gets a signal of 0 [Eq. (3)].

For time $t > 0$, we have the following functions. For all other operations, i.e., $\forall\ j > 1 \wedge t > 0$

$$IO_{ij}(t) = IO_{ij}(t-1) + \sum_k OMF_{ijkpq}(t) \qquad \forall\ i, p \in J,$$
$$i = p, j = q + 1, j, q \in O, k \in M, q \neq m, \tag{4}$$

$$IO_F(t) = IO_F(t-1) + \sum_i \sum_k OMF_{ijkF}(t),$$
$$j = m,\ \forall\ k \in M, i \in J, j \in O. \tag{5}$$

IO_{ij} [Eq. (4)] helps to enforce the constraint that a new operation of a job cannot start unless the current operation is complete. At $t = 0$, IO_{ij} is 0. When an operation node gets a signal from the machine node (*OMF,* described later), IO_{ij} becomes 1, which indicates that it is ready to start. IO_F [Eq. (5)] is the input of the final node. It gets an input from all the machines nodes of all the jobs. When IO_F becomes n, we know that all jobs are done.

Activation function: Operation nodes' initial activation state (i.e., at $t = 0$) is 1.

$$\forall\ i \in J, j \in O,$$
$$\theta O_{ij}(t) = \begin{cases} 1 & \text{if} \quad IO_{ij}(t) = 0 \\ 2 & \text{if} \quad (\theta O_{ij}(t-1) = 1 \vee 2) \wedge IO_{ij}(t) = 1 \\ 3 & \text{if} \quad (\theta O_{ij}(t-1) = 2 \vee 3) \wedge \sum_k OMR_{ijk}(t) = -1 \\ 4 & \text{if} \quad \theta O_{ij}(t-1) = 4 \\ & \quad \vee \left(\theta O_{ij}(t-1) = 3 \wedge \sum_k OMR_{ijk}(t) = 0\right) \end{cases}$$

State 1 above implies that operation O_{ij} is not ready to be assigned because input to this operation is still 0. State 2 implies that the operation is ready to be assigned to a machine because its input is 1. State 3 implies that the operation is in process because it is receiving a negative signal from a machine k that it is currently being processed. State 4 implies that the operation is complete and the negative signal from machine k is no longer there.

Output functions: For all heuristics not using the non-greedy scheme (heuristics 1, 3, 5, 8), we have the following output function for the operation nodes:

$$OO_{ij}(t) = \begin{cases} 1 & \text{if } \theta O_{ij}(t) = 2 \quad \forall\ i \in J, j \in O, \\ 0 & \text{otherwise.} \end{cases}$$

If an operation is ready to start [i.e., $\theta O_{ij}(t) = 2$], then the operation node sends a unit signal to each machine node that it can be assigned.

For all heuristics that use non-greedy job slack (heuristics 2, 4, 6, and 9 of Section 3), we have the following functions:

$$OO_{ij}(t) = \begin{cases} 1 & \text{if } \theta O_{ij}(t) = 2 \wedge (\omega_{NG})_{ij} * RSJ_i(t) \\ & \quad < \min_k \{ST_{i^*jk} + PT_{i^*k} - t\}, \qquad i^* \neq i, \\ & \forall\ i \in J, j \in O, k \in M, \\ 0 & \text{otherwise.} \end{cases}$$

If an operation is ready to start [i.e., $\theta O_{ij}(t) = 2$] and the weighted remaining slack on the job is less than the least (start time + processing time − elapsed time) of all other jobs, then the operation node sends a unit signal to each machine node that it can be assigned.

For all heuristics that use non-greedy machine slack (heuristics 7 and 10 of Section 3), we have the following functions:

$$OO_{ij}(t) = \begin{cases} 1 & \text{if } \theta O_{ij}(t) = 2 \wedge (\omega_{NG})_{ij} * RSM_k(t) \\ & \quad < \min_k \{ST_{i^*jk} + PT_{i^*k} - t\}, \qquad i^* \neq i \\ & \forall\ i \in J, j \in O, k \in M, \\ 0 & \text{otherwise.} \end{cases}$$

If an operation is ready to start [i.e., $\theta_{ij}(t) = 2$] and the weighted remaining slack on the machine is less than the least (start time + processing time − elapsed time) of all other jobs, then the operation node sends a unit signal to each machine node that it can be assigned.

F-Node

$$OO_F(t) = \begin{cases} t & \text{if } IO_F(t) = n, \\ 0 & \text{otherwise.} \end{cases}$$

The final node outputs the makespan (t), the moment it receives n signals (one from each job) indicating that all jobs are complete.

4.1.2. Machine Nodes

Input, activation, and output functions of machine nodes are now explained.

Input function:

$$IM_{ijk}(t) = OO_{ij}(t) * \omega_{ij} + \sum_{S(t)} OML_{ijkpq}(t) * \omega_m + \sum_{k^*} OMM_{ijk^*k}(t) * \omega_m + \sum_{p} OMMF_{ijkp}(t) * \omega_m \quad \forall\ i \in J, \quad j, p \in O, \quad k^* \neq k \quad (6)$$

There are four components of $IM_{ijk}(t)$. The first component ($OO_{ij}(t) * \omega_{ik}$) is the weighted output from operation node O_{ij}. Whenever it is positive, it means that machine k is being requested by operation O_{ij} for assignment. Remember OO_{ij} becomes 1 when it is ready to be assigned. The second and third components are either zero or large negative. The second component becomes large negative whenever machine k is already busy with another operation. The third component becomes large negative whenever operation O_{ij} is assigned to another machine. ω_m is a fixed weighted link between machines and is large negative to suppress the output of an operation if the machine is busy or assigned or the operation is assigned to another machine. The fourth component enforces the constraint that the same machine cannot operate on more than one operation of a job.

Activation function:

$$assign_{ijk}(t) = \begin{cases} 1 & \text{if } \quad \text{Max}(IM_{ijk}(t) * FirstHeuristicParameter) \\ & \wedge \text{Max}(SecondHeuristicParameter) \wedge IM_{ijk}(t) > 0, \\ & \forall\ i \in J, j \in O, k \in M, \\ 0 & \text{otherwise.} \end{cases}$$

We have mentioned earlier that the AugNN functions, in addition to enforcing the constraints of the problem, also help embed a chosen heuristic into the problem. We have also seen how using the output of the operation node, the non-greedy aspect of a heuristic was implemented. Through the assign function, the rest of the heuristic is implemented. The assignment takes place if the product of input of the machine node and the heuristic dependent parameter (such as PT or TRPT or RPT of Machine) is positive and highest. The requirement for it being positive is to honor the inhibitory signals. The requirement for highest is what enforces the chosen heuristic.

First or Second HeuristicParameter depends on the chosen heuristic. One of them is job dependent, and the other is machine dependent. They need to be evaluated in a given order. For a particular heuristic, there may not be a second parameter.

Job Dependent Heuristic Parameters

$$= TRPT_j \qquad \text{Total remaining proc time.}$$

Machine Dependent Heuristic Parameters

$$= \begin{cases} RPT_k & \text{Remaining proc time on machine } k. \\ PT_{ik} & \text{Proc time of job } i \text{ on machine } k. \end{cases}$$

If $assign_{ijk}(t) = 1$, then $ST_{ijk} = t$.

Whenever an assignment takes place, we record the start time of the operation O_{ij} on machine k.

If $|S(t)| > 1$, then if $assign_{ijk}(t) = 1$, then $Win_{ik} = 1$.

The Win_{ik} term will be used later during the learning strategy. We want to modify the weights of links based on whether a particular operation node won the competition in case there was more than one node competing for assignment.

Machine nodes' Initial Activation State (i.e., at $t = 0$) is 1.

$$\forall\ i \in M, j \in T,$$

$$\theta M_{ijk}(t) = \begin{cases} 1 & \text{: machine available} \\ 2 \text{ if } (\theta M_{ijk}(t-1) = 1 \vee \theta M_{ijk}(t) = 1) \wedge assign_{ijk}(t) = 1 & \text{: machine busy (just assigned)} \\ 3 \text{ if } (\theta M_{ijk}(t-1) = 2 \vee 3) \wedge t < ST_{ijk} + PT_{ik} & \text{: machine busy (processing)} \\ 4 \text{ if } \theta M_{ijk}(t-1) = 3 \wedge t = ST_{ijk} + PT_{ik} & \text{: machine processed} \\ 5 \text{ if } \theta M_{ijk}(t-1) = 1 \wedge \sum_{p \in S(t)} OML_{ijkpq}(t) * \omega_m < 0 & \text{: assigned to another job} \\ 6 \text{ if } \theta M_{ijk}(t-1) = 4 & \text{: machine } k \text{ is finished processing } O_{ij} \\ 1 \text{ if } (\theta M_{ijk}(t-1) = 1 \vee 5) \wedge \sum_{p \in S(t)} OML_{ijkpq}(t) * \omega_m = 0 & \text{: released by other job or not assigned to any other job} \\ 1 \text{ if } \theta M_{ijk}(t-1) = 1 \wedge \sum_{k*} OMM_{ijk*k}(t) * \omega_m < 0 & \text{: available but operation assigned} \end{cases}$$

At $t = 0$, all machines are ready to start (State 1). When an assignment occurs on a machine, that machine enters state 2. State 2 turns into state 3 the following time unit, and state 3 continues till the machine is processing a job. As soon as a machine is done processing, it enters state 4. When a particular machine node is assigned to a job, all other nodes that represent the same machine are given state 5. For example, if machine node M_{111} is assigned to operation O_{11} then machine nodes M_{211}, M_{311} enter state 5. In state 5, they cannot be assigned to an operation. When a machine is finished processing an operation, it reaches state 6. A machine node enters the state of 1 from a state of 5 if it stops receiving a negative signal from other machine nodes.

Output functions:

$$OMF_{ijkpq}(t) = \begin{cases} 1 & \text{if } \theta M_{ijk}(t) = 4, \\ 0 & \text{if } \theta M_{ijk}(t) = 1, 2, 3, 5, 6, \end{cases}$$

$$\forall\ i, p \in J, j, q \in O, k \in M, p = i + 1.$$

Whenever a machine node is done processing an operation, i.e., it reaches state 4, it sends a signal to the operation ahead of it that it might start.

$$OMR_{ijk}(t) = \begin{cases} -1 & \text{if } \theta M_{ijk}(t) = 2, 3, \\ 0 & \text{if } \theta M_{ijk}(t) = 1, 4, 5, 6, \end{cases}$$

$$\forall\ i \in J, j \in O, k \in M.$$

Whenever a machine node is busy processing an operation (i.e., in states 2 or 3), it sends a negative signal to the operation node that it is processing. This helps switch the state of the operation node from 2 to a 3.

$$OML_{ijkpq}(t) = \begin{cases} 1 & \text{if } \theta M_{ijk}(t) = 2, 3, \\ 0 & \text{if } \theta M_{ijk}(t) = 1, 4, 5, 6, \end{cases}$$

$$\forall\ i, p \in J, j, q \in O, k \in M, p \neq i.$$

Whenever a machine node is busy processing an operation (i.e., in states 2 or 3), it also sends a signal to the machine nodes corresponding to the same machine for other jobs in the same machine layer. This makes this machine ineligible to take up other jobs, because these machine nodes receive a high negative IM. This scheme enforces the constraint that the same machine cannot operate on two jobs at the same time.

$$OMM_{ijk*k}(t) = \begin{cases} 1 & \text{if } \theta M_{ijk}(t) = 2, 3 \\ 0 & \text{if } \theta M_{ijk}(t) = 1, 4, 5, 6, \end{cases}$$

$$\forall\ i \in J, j \in O, k \in M, k* \neq k.$$

Whenever a machine node is busy processing an operation (i.e., in states 2 or 3), it also sends a signal to other machine nodes in the same layer for the same job. This makes these other machine nodes ineligible to take up the same job, because the other machines receive a high negative IM. This scheme enforces the constraint that the same operation cannot be processed by more than one machine at the same time.

$$OMMF_{ijkp}(t) = \begin{cases} 1 & \text{if } \theta M_{ijk}(t) = 2, 3 \\ 0 & \text{if } \theta M_{ijk}(t) = 1, 4, 5, 6, \end{cases}$$

$$\forall\ i \in J, p, j \in O, k \in M, p > j.$$

Whenever a machine node is busy processing an operation (i.e., in states 2 or 3), it also sends a signal to machine nodes corresponding to the same machine in other machine layers in the forward direction for the same job. This ensures that the same machine is not assigned to the same job more than once. The output of F represents the makespan, and the $assign_{ijk}(t)$ gives the schedule. If a machine is either assigned or released during a certain time unit, all functions need to be recalculated without incrementing the time clock.

Learning strategy: A learning strategy is required to modify the weights. The idea behind weight modification is that if the error is too high, then the probability of different machines being winners is higher during subsequent iteration. Since the machine with the highest value of *IM,* is the winner, an increase of weights will make the machine more likely two in and conversely a decrease of weight will make it less likely. The magnitude of change should be a function of the magnitude of the error and of some job parameter, such as processing time. Keeping these points in mind, the following learning strategy is used for the weights on the links.

Winning tasks: If $Win_{ik} = 1$, then $(\omega_{ik})_{c+1} = (\omega_{ik})_c - \alpha * PT_{ik} * \varepsilon_c \quad \forall\, i \in J, k \in M.$

Non-winning tasks: If $Win_{ik} = 0$, then $(\omega_{ik})_{c+1} = (\omega_{ik})_c + \alpha * PT_{ik} * \varepsilon_c \quad \forall\, i \in J, k \in M.$

We modify the weights for non-greedy heuristics w_{NG} as follows:

$$((\omega_{NG})_{ik})_{c+1} = ((\omega_{NG})_{ik})_c + \alpha * PT_{ik} * \varepsilon_c.$$

In addition to these weight changes in each iteration, we propose two additional features that govern learning, namely, reinforcement and backtracking. These features are explained here briefly.

Reinforcement: Neural networks use the concept of positive reinforcement of weights if the network performs well. We implement this idea of reinforcement by implementing the following rule. If in a particular iteration the makespan improves, the weight changes of that iteration with respect to the previous iteration are magnified by a factor called the reinforcement factor (RF). We test several reinforcement factors from 1 through 5 to see which factor works better overall. We found that an RF of 4 worked well for most problems.

$$(\omega_{ik})_{c+1} = (\omega_{ik})_{c+1} + RF * ((\omega_{ik})_{c+1} - (\omega_{ik})_c).$$

Backtracking: Sometimes it is possible to not obtain any improvement over several iterations. When this happens, it is best to abandon that search path and start over from the previous best solution weights. We can parametrize how many iterations of no improvement to tolerate. This backtracking technique was part of our learning strategy. In order to do this, we store the set of weights corresponding to the best solution obtained so far and revert back to it whenever solution does not improve for some iterations.

End of iteration routines:

1. Calculate the gap (the difference between obtained makespan and the lower bound). Lower bound is the maximum amount of time that a job or machine requires.
   Lower Bound
   $$= \max\left\{\max_k\left\{\sum_{i=1}^{n} PT_{ik}\right\}, \max_i\left\{\sum_{k=1}^{m} PT_{ik}\right\}\right\}.$$
   The lower bound can be calculated once at the beginning.
2. Store the best solution so far.
3. If the lower bound is reached, or if the number of iterations is greater than a specified number, stop the program.
4. If continuing with the next iteration, modify weights using the learning strategy. Apply backtracking and reinforcement, whenever necessary.

5. COMPUTATIONAL EXPERIMENTS AND RESULTS

Three sets of problems are used to evaluate the performance of our algorithm. The first set consists of benchmark problems by Taillard [24]. We focused on the larger problems (15 × 15 and 20 × 20). These square problems are harder to solve than the problems in which the number of jobs and the number of machines are different (Taillard [24]). The second set of problems we use is from Gueret and Prins [12]. We focus on ten problem instances of size 10 × 10. The third set of problems was generated by us. These problems are of larger size—25 × 25, 30 × 30, 50 × 50, and 100 × 100. We generate 10 problems of each size. The processing times were integers, generated using uniform distribution in the interval [1, 99].

The AugNN approach was coded in Visual Basic 6.0 running on Windows-XP® operating system and implemented on a Celeron-900 personal computer. In our imple-

Table 1. Results for Taillard's instances.

Problem instance	Lower bound	Single pass best	AugNN best	Solution gap AugNN best (%)	Improvement from single pass (%)	AugNN DS/LTRPAM NG job slack	Solution gap with AugNN DS/LTRPAM NG JS (%)
tai15 × 15 a	937	998	***937***	0.00	6.11	***937***	0.00
tai15 × 15 b	918	957	***918***	0.00	4.07	*920*	0.22
tai15 × 15 c	871	929	***871***	0.00	6.24	***871***	0.00
tai15 × 15 d	934	1033	***934***	0.00	9.58	***934***	0.00
tai15 × 15 e	946	1020	***946***	0.00	7.25	***946***	0.11
tai15 × 15 f	933	1101	***933***	0.00	15.25	***933***	0.00
tai15 × 15 g	891	939	***891***	0.00	5.11	***891***	0.00
tai15 × 15 h	893	971	***893***	0.00	8.03	***893***	0.00
tai15 × 15 i	899	989	*901*	0.22	8.89	*907*	0.88
tai15 × 15 j	902	1072	***902***	0.00	15.85	*905*	0.33
tai20 × 20 a	1155	1249	***1155***	0.00	7.52	***1155***	0.00
tai20 × 20 b	1241	1252	*1242*	0.08	0.80	*1244*	0.24
tai20 × 20 c	1257	1285	***1257***	0.00	2.17	***1257***	0.00
tai20 × 20 d	1248	1295	***1248***	0.00	3.62	***1248***	0.00
tai20 × 20 e	1256	1296	***1256***	0.00	3.08	***1256***	0.00
tai20 × 20 f	1204	1294	***1204***	0.00	6.95	***1204***	0.00
tai20 × 20 g	1294	1511	***1294***	0.00	14.36	*1298*	0.31
tai20 × 20 h	1169	1257	*1173*	0.34	6.68	*1173*	0.34
tai20 × 20 i	1289	1321	***1289***	0.00	2.42	***1289***	0.00
tai20 × 20 j	1241	1340	***1241***	0.00	7.39	***1241***	0.00

mentation, the learning coefficient α is set to 0.001 and the weights are initialized at 1. An initial solution is generated using the heuristics given in Section 3. The weights are modified after each iteration, using the learning strategies explained in Section 4. The stopping criterion is to stop if the solution is equal to the lower bound or if a predetermined number of maximum iteration is reached. we set the maximum number of iterations to 1500 for Taillard's instances of size 15 × 15, and 20 × 20 and 5000 for the Gueret and Prins instances. For the new instances we run 500 iterations for 25 × 25, 30 × 30, and 50 × 50 and 200 for 100 × 100 problems.

Table 1 shows the results for Taillard's benchmark problems. We ran AugNN in conjunction with all 10 heuristics for all problems. So, for each problem instance, we have 10 results. In the interest of space, we report the best AugNN result. In addition, we report the results of "DS/LTRPAM Non-Greedy with Job Slack" heuristic, which performed the best amongst all ten heuristics. Note that DS/LTRPAM heuristic was proposed in this study.

For each problem instance we report the lower bound (LB), the initial single-pass solution of the heuristic which gave the best result with AugNN, the best AugNN solution (AugNN Best), the solution gap from the lower bound in percentage (Solution Gap (Best)) = {((AugNN Best − LB)/LB) $*$ 100}, and the percent improvement from the

Table 2. Results for Gueret and Prins instances.

Problem instance	Lower bound	Single pass best	AugNN best	Solution gap AugNN best (%)	Improvement from single pass (%)	AugNN DS/LTRPAM NG job slack	Solution gap with AugNN DS/LTRPAM NG JS (%)
gp10-01	1059	1305	*1113*	5.10%	14.71%	1113	5.10%
gp10-02	1065	1328	*1117*	4.88%	15.89%	1117	4.88%
gp10-03	1046	1265	*1104*	5.54%	12.73%	1104	5.54%
gp10-04	1045	1210	*1098*	5.07%	9.26%	1098	5.07%
gp10-05	1044	1249	*1095*	4.89%	12.33%	1095	4.89%
gp10-06	1055	1301	*1074*	1.80%	17.45%	1074	1.80%
gp10-07	1075	1218	*1084*	0.84%	11.00%	1098	2.14%
gp10-08	1047	1264	*1104*	5.44%	12.66%	1105	5.54%
gp10-09	1065	1396	*1130*	6.10%	19.05%	1130	6.10%
gp10-10	1057	1353	*1099*	3.97%	18.77%	1099	3.97%

Table 3a. Results for randomly generated problem instances of sizes 25 × 25 and 30 × 30.

Problem instance	Lower bound	Single pass best	AugNN best	Solution gap AugNN best (%)	Improvement from single pass (%)	AugNN DS/LTRPAM NG mach. slack	Solution gap with AugNN DS/LTRPAM NG MS (%)
25 × 25 a	1426	1644	*1427*	0.07	13.20	*1427*	0.07
25 × 25 b	1599	1720	***1599***	0.00	7.03	***1599***	0.00
25 × 25 c	1559	1662	***1559***	0.00	6.19	***1559***	0.00
25 × 25 d	1571	1690	***1571***	0.00	7.04	***1571***	0.00
25 × 25 e	1568	1660	***1568***	0.00	5.54	***1568***	0.00
25 × 25 f	1485	1632	***1485***	0.00	9.01	***1485***	0.00
25 × 25 g	1604	1717	***1604***	0.00	6.58	***1604***	0.00
25 × 25 h	1442	1563	***1442***	0.00	7.74	***1442***	0.00
25 × 25 i	1585	1739	***1585***	0.00	8.85	***1585***	0.00
25 × 25 j	1531	1640	***1531***	0.00	6.64	***1531***	0.00
30 × 30 a	1745	1891	***1745***	0.00	7.72	***1745***	0.00
30 × 30 b	1875	1994	***1875***	0.00	5.97	***1875***	0.00
30 × 30 c	1843	2024	***1843***	0.00	8.94	***1843***	0.00
30 × 30 d	1851	1977	***1851***	0.00	6.37	***1851***	0.00
30 × 30 e	1885	1998	***1885***	0.00	5.65	***1885***	0.00
30 × 30 f	1751	1853	***1751***	0.00	5.51	***1751***	0.00
30 × 30 g	1908	2075	***1980***	0.00	8.05	*1909*	0.05
30 × 30 h	1741	1893	*1742*	0.06	7.98	*1746*	0.29
30 × 30 i	1919	1980	***1919***	0.00	3.08	***1919***	0.00
30 × 30 j	1824	1917	***1824***	0.00	4.85	***1824***	0.00

single-pass solution. We also report the DS/LTRPAM Non-Greedy with Job Slack results and gaps. The optimal solutions appear in bold. As is seen in Table 1, 17 out of 20 problems were solved to optimality. Of the remaining three problems, the solution gap from the lower bound varies between 0.08% and 0.34%, an insignificant amount. The average solution gap for all 20 problems is only 0.032%. The computing time is relatively small. Average computing times and average num-

Table 3b. Results for randomly generated problem instances of sizes 50 × 50 and 100 × 100.

Problem instance	Lower bound	Single pass best	AugNN beste	Solution gap AugNN best (%)	Improvement from single pass (%)	AugNN DS/LTRPAM NG mach. slack	Solution gap with AugNN DS/LTRPAM NG MS (%)
50 × 50 a	3029	3157	***3029***	0.00	4.05	***3029***	0.00
50 × 50 b	3039	3181	***3039***	0.00	4.46	***3039***	0.00
50 × 50 c	3128	3280	*3129*	0.03	4.60	*3129*	0.03
50 × 50 d	3038	3209	***3038***	0.00	5.33	***3038***	0.00
50 × 50 e	2964	3105	*2966*	0.07	4.47	*2966*	0.07
50 × 50 f	2933	3100	*2936*	0.10	5.29	*2937*	0.14
50 × 50 g	3011	3160	***3011***	0.00	4.71	***3011***	0.00
50 × 50 h	3140	3286	***3140***	0.00	4.44	***3140***	0.00
50 × 50 i	3032	3087	***3032***	0.00	1.78	*3033*	0.03
50 × 50 j	2964	3097	***2964***	0.00	4.29	*2968*	0.14
100 × 100 a	5759	5875	*5762*	0.05	1.92	*5765*	0.10
100 × 100 b	5787	5953	***5787***	0.00	2.79	***5787***	0.00
100 × 100 c	5643	5958	*5647*	0.07	5.22	*5647*	0.07
100 × 100 d	5697	5866	***5697***	0.00	2.88	*5701*	0.07
100 × 100 e	6023	6248	***6023***	0.00	3.60	***6023***	0.00
100 × 100 f	5784	5951	*5789*	0.09	2.72	*5791*	0.12
100 × 100 g	5667	5875	*5669*	0.04	3.51	*5673*	0.10
100 × 100 h	5765	5945	***5765***	0.00	3.03	*5768*	0.05
100 × 100 i	5630	5712	*5635*	0.09	1.35	*5636*	0.11
100 × 100 j	5701	5870	*5703*	0.04	2.84	*5709*	0.14

Table 4. Computational times (in seconds) and iterations.

Problem instance set	Average computing time for best solution	Average number of iterations for best solution
tai15 × 15	17.3	118
tai20 × 20	28.6	143
GP10 × 10	36.5	214
25 × 25	56.2	121
30 × 30	76.8	167
50 × 50	861	227
100 × 100	2847	116

ber of iterations are given in Table 4. The AugNN DS/LTR-PAM Non-Greedy Job Slack heuristic performs the best for Taillard's instances, although almost all of the heuristics find optimal or near optimal solutions. The improvement due to the iterative approach of AugNN over the first pass solution ranged from 4.07% to 15.85% for the 15 × 15 problems and 0.80% to 14.36% for the 20 × 20 problems and averaged 8.64% for 15 × 15 and 5.50% for 20 × 20.

Table 2 gives the AugNN results for the ten Gueret and Prins hard instances of size 10 × 10. We report the best results from all 10 heuristics and also for the DS/LTRPAM Non-Greedy Job Slack heuristic. The improvement due to AugNN over single-pass heuristics ranged from 9.26% to 19.05% with an average improvement of 14.39%. The gaps from the lower bound ranged from 0.84% to 6.10%.

The results for the third set of problems are given in Tables 3a and 3b. The lower bound solution is found in 9 out of 10 instances for each of 25 × 25 and 30 × 30 problems. For the remaining two instances the solution gap is only 0.07% and 0.06% and the average solution gap is 0.0065%. The improvement due to the iterative approach of AugNN over the first pass solution ranged from 5.54% to 13.20% for the 25 × 25 problems and 3.08% to 8.94% for the 30 × 30 problems and average was 7.78% for 25 × 25, 6.41% for 30 × 30. For these problems DS/LTRPAM Non-Greedy Machine Slack heuristic gave the best results.

For the 50 × 50 problems, lower-bound solutions were obtained for 7 out of 10 instances, 5 of which were by our new heuristic DS/LTRPAM Non-Greedy Machine Slack. The remaining 3 out of 10 problems were solved within 0.03–0.1% of the lower bound. The improvement due to AugNN's iterative approach over the single-pass heuristic was 1.78% to 5.33% with an average of 4.34%. For the 100 × 100 problems, 4 out of 10 problems were solved to lower bound. The remaining 6 problems were solved within 0.04–0.09% of the lower bound. The improvement due to AugNN over single-pass heuristic ranged from 1.35% to 5.22% with an average of 2.98%. The average solution gap for all 50 × 50 and 100 × 100 problems was 0.029%. For these problems also, DS/LTRPAM Non-Greedy Machine Slack heuristic gave the best results.

Table 4 summarizes the computational times for the various sets of problems. It also provides the average number of iterations taken for the best solution.

Table 5 shows comparative results of other meta-heuristics that have been attempted on Taillard's benchmark problems of size 15 × 15 and 20 × 20. The solutions that have

Table 5. Comparison of results of AugNN with other techniques (Taillard's instances).

Problem instance	Lower bound	AugNN	Dorndorf B&B	Taillard TS	Liaw GA	Liaw SA	Liaw TS	Prins GA	Blum ANT
tai15 × 15 a	937	***937***	937	937	937	937	937	937	937
tai15 × 15 b	918	***918***	NR[a]	918	918	920	920	918	918
tai15 × 15 c	871	***871***	871	871	871	871	871	871	871
tai15 × 15 d	934	***934***	934	934	934	934	934	934	934
tai15 × 15 e	946	***946***	946	950	946	946	949	946	946
tai15 × 15 f	933	***933***	933	933	933	933	933	933	933
tai15 × 15 g	891	***891***	891	891	891	891	891	891	891
tai15 × 15 h	893	***893***	893	893	893	893	893	893	893
tai15 × 15 i	899	*901*	899	908	899	905	910	899	902
tai15 × 15 j	902	***902***	902	902	902	902	906	902	902
tai20 × 20 a	1155	***1155***	1155	1155	1155	1155	1155	1155	1155
tai20 × 20 b	1241	*1242*	NR	1244	1242	1253	1246	1241	1247
tai20 × 20 c	1257	***1257***	1257	1257	1257	1257	1257	1257	1257
tai20 × 20 d	1248	***1248***	NR	1248	1248	1248	1248	1248	1248
tai20 × 20 e	1256	***1256***	1256	1256	1256	1256	1256	1256	1256
tai20 × 20 f	1204	***1204***	1204	1209	1204	1204	1204	1204	1204
tai20 × 20 g	1294	***1294***	1294	1294	1294	1294	1298	1294	1296
tai20 × 20 h	1169	*1173*	NR	1173	1177	1189	1184	1171	1177
tai20 × 20 i	1289	***1289***	1289	1289	1289	1289	1289	1289	1289
tai20 × 20 j	1241	***1241***	1241	1241	1241	1241	1241	1241	1241

[a]NR: Not reported.

Table 6. Comparison of results of AugNN with other techniques (Gueret and Prins instances).

Problem instance	Lower bound	AugNN	Brucker et al. B&B [7]	Prins GA [22]	Blum ANT [4]	Longest processing time [22]	BBox [14]	Blum beam ANT [5]
gp10-01	1059	*1113*	1151	1113	1108	1151	1145	1099
gp10-02	1065	*1117*	1178	1120	1102	1477	1144	1101
gp10-03	1046	*1104*	1162	1101	1097	1512	1160	1082
gp10-04	1045	*1098*	1165	1090	1089	1244	1142	1093
gp10-05	1044	*1095*	1125	1094	1091	1356	1125	1083
gp10-06	1055	*1074*	1173	1074	1072	1448	1144	1088
gp10-07	1075	*1084*	1172	1083	1081	1487	1138	1084
gp10-08	1047	*1104*	1181	1098	1099	1313	1131	1099
gp10-09	1065	*1130*	1184	1121	1124	1403	1148	1121
gp10-10	1057	*1099*	1172	1095	1092	1237	1169	1097

not been solved to the lower bound are shaded. Our results (shown in the third column) appear to be quite competitive with the results of other techniques, although they do are not the best.

Table 6 shows the comparative results for Gueret and Prins instances with other techniques found in the literature. Again, AugNN results appear to be very competitive with other techniques, although it is not the best. Blum's [5] Beam-Ant approach appears to be the best. However, it may be noted that Blum gave the best results from 20 runs and each run has a time limit of 1000 seconds for GP instances. Our average CPU time for the GP instances was 36.5 seconds.

Table 7 shows the CPU times for AugNN and other meta-heuristics for the Taillard's problems. Due to the differences in processor speeds, a direct comparison, of course, is difficult.

6. SUMMARY AND CONCLUSIONS

In this study, we have proposed and developed a formulation for applying the augmented-neural-networks (AugNN) approach for solving the classical open-shop scheduling problem (OSSP) and have tested the formulation on several benchmark problems from the literature and also on some new, larger problems. The computational tests show excellent results on all the problems, making this new AugNN approach a strong contender to techniques such as tabu search simulated annealing, ant-colonies, and genetic algorithms, for solving the OSSP and other types of scheduling problems. The approach, which combines the benefits of single-pass heuristics approach and the iterative learning approach, is able to give very good results in a short time, making it the technique of choice especially for tackling large instances. During the course of this study, we also developed a new single-pass heuristic (DS/LTRPAM). Non-greedy versions of these heuristics were also developed. The non-greedy DS/LTRPAM in conjunction with AugNN gave better results than other heuristics. We also formalized some new learning strategies such as reinforcement and backtracking to improve the learning process and the solution quality. Large problem instances of sizes 25×25, 30×30, 50×50, and 100×100 were also generated.

Since the AugNN approach is still in its nascent state of development, more work needs to be done. Better learning strategies can be developed in future studies. Also, the approach can be attempted on other variations of the OSSP in which the objective is other than minimization of the makespan. Since larger problems can now be solved in reasonable time, industry should make use of this approach for solving the OSSP.

Table 7. Comparison of average CPU time in seconds for Taillard's 15×15 and 20×20 problem sets.

	AugNN[a]	Dorndorf B&B[b]	Taillard TS	Liaw GA[c]	Liaw SA[d]	Liaw TS[e]	Prins GA	Blum ANT[f]
tai15 $\times$ 15	17.3	600	NR	138	372	150	NR	137
tai20 $\times$ 20	28.6	3600	NR	465	1632	218	NR	349

[a]Used Celeron 900 MHz machine.
[b]Used Pentium-II 333 MHz machine.
[c]Used Pentium-II 266 MHz machine.
[d]Used Pentium-II machine.
[e]Used Pentium 133 MHz machine.
[f]Used AMD Athlon 1100 MHz machine.
NR: Not reported.

REFERENCES

[1] A. Agarwal, V.S. Jacob, H. Pirkul, Augmented neural networks for task scheduling, European J Oper Res 151(3) (2003), 481–502.

[2] A. Agarwal, V.S. Jacob, and H. Pirkul, Improved augmented neural networks for scheduling problems, INFORMS J Comput, forthcoming.

[3] D. Alcaide, J. Sicilia, and D. Vigo, Heuristic approaches for the minimum makespan open shop problem, Trab Invest Oper 5(2) (1997), 283–296.

[4] C. Blum, An ant-colony optimization algorithm to tackle shop scheduling problems, Technical report, TR/IRDIA/2003-01, IRDIA, Université Libre de Bruxelles, Belgium, 2003.

[5] C. Blum, Beam-ACO-hybridizing ant colony optimization with beam search: An application to open shop scheduling, Comput Oper Res 32 (2005), 1565–1591.

[6] H. Brasel, T. Tautenhahn, and F. Werner, Constructive heuristic algorithms for the open shop problem, Computing 51 (1993), 95–110.

[7] P. Brucker, J. Hurink, B. Jurish, and B. Wostmann, A branch and bound algorithm for the open-shop problem, Discrete Appl Math 76 (1997), 43–59.

[8] U. Dorndorf, E. Pesch, and T. Phan-Huy, Solving the open shop scheduling problem, J Sched 4 (2001), 157–174.

[9] Y.P.S. Foo and Y. Takefuji, Stochastic neural networks for solving job-shop scheduling, Proc Joint Int Conf Neural Networks 2 (1988), 275–290.

[10] T. Gonzalez and S. Sahni, Open shop scheduling to minimize finish time, ACM 23(4) (1976), 665–679.

[11] C. Gueret, N. Jussien, and C. Prins, Using intelligent backtracking to improve branch-and-bound methods: An application to open shop problems, European J Oper Res 127 (2000), 344–354.

[12] C. Gueret and C. Prins, A new Lower Bound for the open-shop problem, Ann Oper Res 92 (1998), 165–183.

[13] C. Gueret and C. Prins, Classical and new heuristics for the open shop problem: A computational evaluation, European J Oper Res 107 (1998), 306–314.

14. C. Gueret and C. Prins, Efficient heuristic black boxes for the open-shop: Comparison with other methods on three benchmarks. Research Report 98/11/AUTO, École des Mines de Nantes, France, 1998.

[15] J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, Biol Cybernet 52 (1985), 141–152.

[16] S. Khuri and S. Miryala, Genetic algorithms for solving open shop scheduling problems, Progress in artificial intelligence, Lecture Notes in Artificial Intelligence, Springer, Berlin, 1999, pp. 357–369.

[17] C.F. Liaw, An iterative improvement approach for the non-preemptive open shop scheduling problem, European J Oper Res 111 (1998), 509–517.

[18] C.F. Liaw, A tabu search algorithm for the open shop scheduling problem, Comput Oper Res 26 (1999), 109–126.

[19] C.F. Liaw, Applying simulated annealing to the open shop scheduling problem, IIE Trans 31 (1999), 457–465.

[20] C.F. Liaw, A hybrid genetic algorithm for the open shop scheduling problem, European J Oper Res 124 (2000), 28–42.

[21] M. Pinedo, Scheduling: Theory, algorithms, and systems, Prentice Hall, Upper Saddle River, NJ, 1995.

[22] C. Prins, Competitive genetic algorithms for the open shop scheduling problem, Math Methods Oper Res 52 (2000), 389–411.

[23] I. Sabuncuoglu and B. Gurgun, A neural network model for scheduling problems, European J Oper Res 93 (1996), 288–299.

[24] E. Taillard, Benchmarks for basic scheduling problems, European J Oper Res 64 (1993), 278–285.