Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 175 (2006) 296–317
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Heuristics and augmented neural networks for task scheduling with non-identical machines

Anurag Agarwal [a,*], Selcuk Colak [a], Varghese S. Jacob [b], Hasan Pirkul [b]

[a] *Department of Decision and Information Sciences, Warrington College of Business, University of Florida, Gainesville, FL 32611-7169, USA*
[b] *School of Management, University of Texas at Dallas, Richardson TX 75083-0688, USA*

Received 6 June 2003; accepted 14 March 2005
Available online 19 July 2005

Abstract

We propose new heuristics along with an augmented-neural-network (AugNN) formulation for solving the makespan minimization task-scheduling problem for the non-identical machine environment. We explore four task and three machine-priority rules, resulting in 12 combinations of single-pass heuristics. The task-priority rules are Highest-Level-First (HLF), Highest-Total-Remaining-Processing-Time-First (HTRPTF), Smallest-Latest-Finish-Time-First (SLFTF) and Minimum-Slack-First (MSF). For machine priority, we propose a greedy rule called Fastest-Available-Machine-First (FAMF), and two non-greedy rules: (1) Fastest-Available-Machine-First-With-Conditional-Wait-1 (FAMF-CW-1), and (2) Fastest-Available-Machine-First-With-Conditional-Wait-2 (FMF-CW-2). The AugNN approach integrates neural-network learning with domain and problem specific knowledge through heuristics, to produce good results. A single-pass heuristic is embedded in a neural network designed specifically for each problem. We give the AugNN formulation for each of the 12 heuristics and show computational results on 100 randomly generated problems of sizes ranging from 20 to 70 tasks and 2 to 5 machines. Results demonstrate that AugNN provides significant improvement over single-pass heuristics. The reduction in the gap between the obtained solution and the lower-bound due to AugNN over single-pass heuristics ranged from 24.4% to 50%. As far as heuristics, the non-greedy machine-priority rules performed significantly better than the greedy rule. The average gaps for the non-greedy rules ranged from 16.1% to 23.5% compared to 33.7% to 40.4% for greedy. For AugNN, for non-greedy rules, the gap ranged from 11.5% to 15.5% compared to 18.4% to 25.0% for greedy. The HLF and HTRPTF task priority rules performed better than the other two. The HTRPTF/FAMF-CW-1 combination gave the best results, closely followed by HLF/FAMF-CW-2 and HTRPTF/FAMF-CW-2 combinations.

* Corresponding author. Tel.: +1 3523927300; fax: +1 3523925438.
E-mail addresses: aagarwal@ufl.edu (A. Agarwal), scolak@ufl.edu (S. Colak), vjacob@utdallas.edu (V.S. Jacob), hpirkul@utdallas.edu (H. Pirkul).

0377-2217/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2005.03.045

Keywords: Scheduling; Neural networks; Heuristics

1. Introduction

The problem of optimally scheduling non-preemptive tasks with precedence constraints, on non-identical machines (or processors) belongs to the class of NP-Hard problems (De and Morton, 1980). The problem occurs in many computing environments. For example, in a multiprocessor environment, a computing job is broken down into smaller tasks, to be performed on different, and often, non-identical processors (Ibarra and Kim, 1977; El-Rewini et al., 1995). In network computing environment, a job may be processed on several idle processors available on a computer network (Saltzman, 1995). Processors on a network are likely to be non-identical. The problem also appears in an n-tier client–server environment where a job is distributed across servers of varying speeds. Similarly, on the Internet, file transfer occurs in packets, where packets go through various paths having processors of different speeds. Traditionally, the scheduling problem with non-identical machines is found in the manufacturing environment (Graves, 1981). There is therefore a need for good scheduling techniques for scheduling in the non-identical machine environment. Any improvements in the solution procedures for scheduling for this type of problem will have implications in both computing and manufacturing environments.

We consider such a task-scheduling problem in which a set of non-preemptive tasks with precedence constraints is to be scheduled on non-identical and non-uniform machines, to minimize makespan. Each task needs to be processed on only one machine and a machine can handle only one task at a time. The distinguishing feature of this problem is that the processing times of tasks depend on the machine. We assume non-uniformity of machines, i.e., a fast machine is not always faster than a slower machine for all tasks. For example, assume that machine A has a higher processing speed than machine B, but that machine B has higher memory. A CPU intensive task would run faster on machine A, while an I/O intensive task would run faster on machine B. The uniform machine case would be a special case of the non-identical machine case.

Due to its NP-Hard nature, there are not likely to be any polynomial-time algorithms to optimally solve this problem. Numerous heuristics are available for various scheduling problems involving the use of different dispatching rules for scheduling unexecuted ready tasks to machines or processors (Panwalker and Iskander, 1977; El-Rewini et al., 1995).

Several heuristics and other iterative search techniques exist for the task-scheduling problem for the case of identical machines (Kasahara and Narita, 1984; Agarwal et al., 2003, in press), or uniform machines (Epstein and Sgall, 2000; Gonzalez and Sahni, 1978), but, to the best of our knowledge, none for non-identical machines. Heuristics exist for non-identical machine scheduling problem for the case where all tasks are ready at time zero, i.e., the tasks do not follow precedence constraints (De and Morton, 1980). For the problem we consider in this paper, a heuristic would be a combination of a task-priority rule and a machine priority rule. While task-priority rules have been well studied, machine priority rules have not.

One of the difficulties with the non-identical machine problem is finding good quality lower-bounds for comparison of results. In the identical-machine case, relaxing the resource constraint gives a quick critical-path based lower-bound of fairly good quality. For the non-identical machine case, finding a lower-bound requires a second assumption that all tasks are assigned to the fastest machine. If the differences in processing times between a faster and a slower machine are substantial, a likely scenario, this second assumption leads to poor quality lower bounds. Therefore, the gaps between the obtained solution and the lower-bound are likely to be much wider for the non-identical machine case than those for the identical-machine case. In other words, the performance ratio (solution to lower bound) for search techniques for this type of problem will inherently be poor. Further, it will be hard to show the optimality of any reasonably sized problem. The

only way to show the effectiveness of an iterative search technique would be to show significant reductions in the gaps, compared to single-pass heuristics.

In this paper, we propose 12 heuristics and augmented neural networks (AugNN) for solving this non-identical machine scheduling problem. As mentioned earlier, while heuristics for the identical-machine case involve only task-priority rules, those for the non-identical-machine case are a composite of a task-priority rule and a machine-priority rule. We explore four rules for task priority and three for machine priority. The task-priority rules we consider are routinely found in scheduling literature (Hu, 1961; Panwalker and Iskander, 1977; Agarwal et al., 2003, in press). Same heuristics often appear in different studies with different names or acronyms. We consider Highest-Level-First (HLF), Highest-Total-Remaining-Processing-Time-First (HTRPTF), Smallest-Latest-Finish-Time-First (SLFTF) and Minimum-Slack-First (MSF). Machine-priority rules for the non-identical machine case are not found in the literature. We propose three rules, one greedy and two non-greedy. The greedy heuristic is called Fastest-Available-Machine-First (FAMF). The non-greedy rules are (1) Fastest-Available-Machine-First-with-Conditional-Wait-1 (FAMF-CW-1), and (2) Fastest-Available-Machine-First-With-Conditional-Wait-2 (FAMF-CW-2). Twelve heuristics result from various combinations of these four task and three machine priority rules.

We also formulate the AugNN approach for this problem and show significant improvements over each of the single-pass heuristics. The AugNN approach is shown to have worked well on the task-scheduling problem with identical machines (Agarwal et al., 2003, in press). In this approach, a given scheduling problem is framed as a neural network, where the task and machine nodes act as processing elements of a neural network. Input, output and activation functions for these nodes are defined in a manner that the constraints of the problem are enforced and a heuristic is applied to produce a feasible solution. There are weights associated with the connections between the task and machine nodes. These weights are the same for all the links during the first iteration, therefore the first-iteration solution is no different from a single-pass heuristic solution. In subsequent iterations, however, the weights are modified using a certain learning algorithm to produce different solutions in the search space. The weight change essentially amounts to a perturbation, and thus a non-deterministic local neighborhood search mechanism. We propose input, output and activation functions for the task and machine nodes and also propose a learning strategy. We apply the AugNN approach in conjunction with each of the twelve heuristics and report results on randomly created problems of various sizes.[1]

In Section 2, we discuss the literature briefly. Section 3 describes the various task and machine priority rules with the help of two example problems. Section 4 describes the AugNN framework. The exact mathematical formulation appears in Appendix 1. Computational results are reported and discussed in Section 5. We conclude the paper with summary and conclusions in Section 6.

2. Relevant literature

Research on the use of neural networks for optimization started with the seminal paper by Hopfield and Tank (1985), who first applied neural networks to the traveling-salesman problem. Haldun et al. (1994) provide a review of machine-learning techniques used in scheduling problems. They explore the use of expert systems, rote learning, inductive learning and case-based learning. They only briefly discuss neural-network learning. Sabuncuoglu (1998) has focused his review paper on the use of various neural-networks-based techniques in scheduling. He classifies neural-networks research for scheduling into three types—Hopfield networks based, competitive networks based and multi-layer perceptrons and back-propagation networks

[1] These problems are available on http://bear.cba.ufl.edu/agarwal/taskscheduling/nonidenticalmachines. The best solutions obtained so far are also reported.

based. Sabuncuoglu and Gurgun (1996) propose an enhancement of the Hopfield network by including an external processor that monitors and controls the evolution of the network. This enhancement significantly reduced the inherent complexity of the Hopfield network. Smith et al. (1996) propose a hybrid of Hopfield network and self-organizing networks to overcome the deficiencies of each of these types of network. Smith et al. (1998) explain the Hopfield network, an improved Hopfield network and the self-organizing network for a general optimization problem. Smith (1999) also reviews the state of the art on the use of Hopfield networks to the various optimization problems, including scheduling, in particular job-shop scheduling. Foo and Takefuji (1988) and Satake et al. (1994) apply Hopfield networks to solve the job-shop scheduling problem. Researchers in the field of neural networks for optimization problems are trying to develop a meta-heuristic paradigm to compete with genetic algorithms and simulated annealing.

However, the Hopfield approach has its shortcomings; in particular, the performance deteriorates exponentially with the problem size. Agarwal et al. (2003) proposed an alternative neural-network approach, which they called the augmented-neural-network approach in which use is made of domain and problem specific knowledge. It employs single-pass heuristics, and improvements are sought iteratively through a learning strategy involving weights modification. This approach differs completely from the Hopfield networks or other prior neural-network approaches. It overcomes the shortcoming of performance on large-sized problems.

For a review of the various dispatching rules used in heuristics for scheduling problems, see Adam et al. (1974) and Panwalker and Iskander (1977). De and Morton (1980) give a heuristic for scheduling jobs on equal, uniform and non-identical machines for the case where all jobs are ready at time $t = 0$. Gonzalez and Sahni (1978) and Horowitz and Sahni (1976) discuss issues with uniform machine scheduling and non-identical machine scheduling.

3. Priority rules with examples

For purposes of discussing the priority rules we will use a small example problem. Fig. 1 shows a sample task graph of nine tasks (T1–T9) and four non-identical machines (M1–M4). The nine tasks follow precedence constraints, represented by the given acyclic directed graph. Each task can be processed on any of the four machines, but needs to be processed on only one of them. The tasks are non-preemptive. The objective is to minimize the makespan or the completion time of the last task in the task graph. Processing times for each task on each machine are known and deterministic and shown next to each task node. Each task shows four processing times, in order, for each of the four machines.

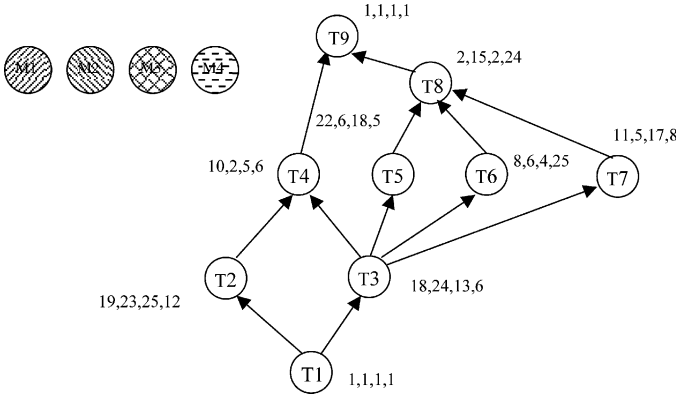

Fig. 1. A sample task graph.

3.1. Task priority rules

The first task-priority rule we use is Highest-Level-First (HLF). In this rule, priority is given to the task with the highest level. The level of a task is defined as the most number of tasks in the remaining path, including the task in question. This heuristic was first proposed by Hu (1961) for the task-scheduling problem. It is a very simple, yet effective rule. The levels of each task in our example are listed in Fig. 2.

The second rule we use is the Highest-Total-Remaining-Processing-Time-First rule (HTRPTF). This rule gives priority to the task with the highest total remaining processing time (TRPT). TRPT for a task is defined as the highest sum of the processing times of all the tasks in the remaining paths till the final task and including the task in question. Computing TRPT in the non-identical machine environment assumes that tasks will be assigned to a certain machine. We assume that tasks will be assigned to the fastest machine. This heuristic gives very good results for the identical machine case (Agarwal et al., 2003). In the literature, HTRPTF is also sometimes known as Most-Work-Remaining-First. Fig. 2 lists the TRPT of each task.

The third rule we use is the Smallest-Latest-Finish-Time-First heuristic (SLFTF). This rule gives priority to the task with the smallest latest finish time (LFT). Again, the computation of LFT requires the assumption that each task is assigned to the fastest machine. Fig. 2 also lists the LFT for each task.

The last task-priority rule we consider is called Minimum-Slack-First (MSF). This rule gives priority to the task with the smallest slack. Again, we assume that each task will be assigned to the fastest machine. The HLFTF and the MSF heuristics have both given good results in the resource constrained project scheduling problem. Fig. 2 lists the LFT for each task.

3.2. Machine priority rules

We consider three priority rules for machines. The first one is a greedy rule called Fastest-Available-Machine-First (FAMF). Whenever a task becomes ready to start, the fastest of the available machines is assigned to the task. Although this rule is effective, it fails to give good results for problems in which the speed differential between machines is significant. In such situations, assigning a task to a slower machine might actually prolong a particular path unnecessarily, slowing down the entire project. This difficulty can be overcome by some non-greedy rules. The first non-greedy rule we propose uses some problem-specific knowledge, in which the fastest available machine is assigned only if the following condition holds:

$$W_{\mathrm{FM}} + \mathrm{PT}_{\mathrm{FM}} > \mathrm{PT}_{\mathrm{FAM}},$$

Task-Priority Rule	Task								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Level	5	3	4	2	3	3	3	2	1
TRPT	16	15	14	3	8	7	8	3	1
LFT	1	13	8	15	13	13	13	15	16
Slack	0	0	1	0	1	2	1	1	0

a

Level	1	3	2	4	3	3	3	4	5
TRPT	1	2	3	6	4	5	4	6	7
LFT	1	3	2	4	3	3	3	4	5
Slack	1	1	2	1	2	3	2	2	0

b

Fig. 2. Task-priority parameters for the example problem. (a) Absolute numbers for Level, TRPT, LFT and Slack. (b) Ranked numbers for each task for Level, TRPT, LFT and Slack.

where W_{FM} is the wait for the fastest machine, PT_{FM} is processing time on the fastest machine and PT_{FAM} is processing time on the fastest available machine.

If the condition does not hold, we wait for the fastest machine. This condition prevents assignment of a task to a machine that is too slow. We call this rule the Fastest-Available-Machine-First-With-Conditional-Wait-1 (FAMF-CW-1). A further improvement may be possible in some problems by modifying the wait

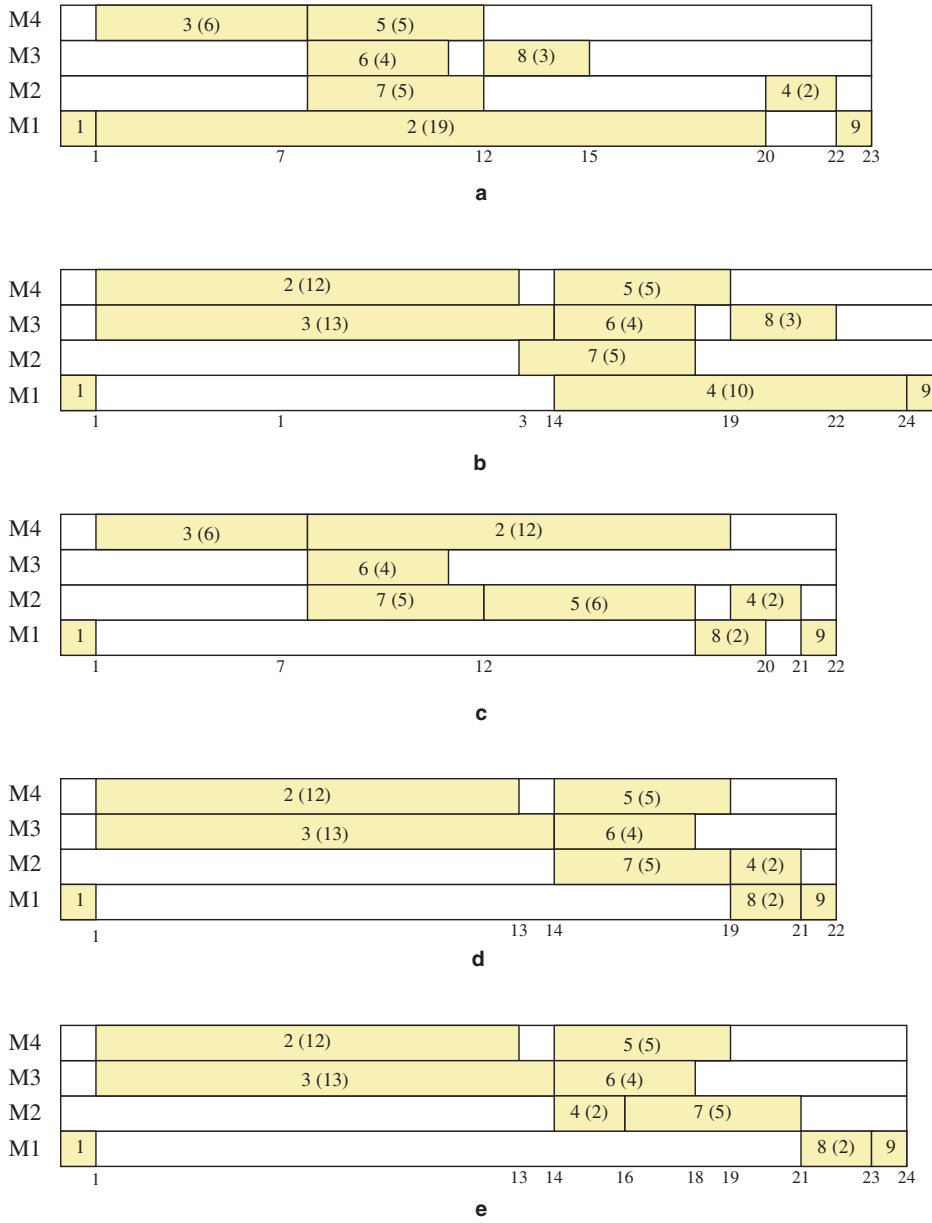

Fig. 3. Gantt charts for various heuristics. (a) HLF/FAMF and SLFTF/FAMF. (b) HTRPTF/FAMF and MSF/FAMF. (c) HLF and HLFTF/FAMF-CW-1 and CW-2. (d) HTRPTF/FAMF-CW-1 and CW-2. (e) MSF/FAMF-CW-1 and CW-2.

condition slightly. A task need not wait for just the fastest machine, but could wait for the second or third (or more) fastest machine as long as a similar condition is satisfied. The second non-greedy rule, called Fastest-Available-Machine-First-With-Conditional-Wait-2 (FMF-CW-2) considers this condition. In this rule, if the fastest machine for a given task is not available, assign the task to the fastest of the available machines only if

$$\min_{i \in B}(W_i + \mathrm{PT}_i) > \mathrm{PT}_{\mathrm{FAM}},$$

where B is the set of machines currently busy; other notation is already described above.

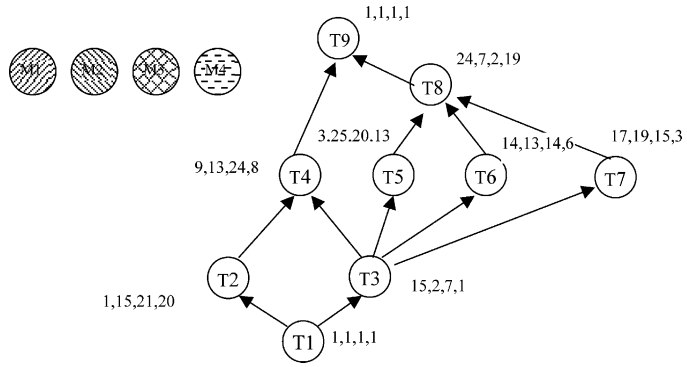

Fig. 4. Second sample task graph.

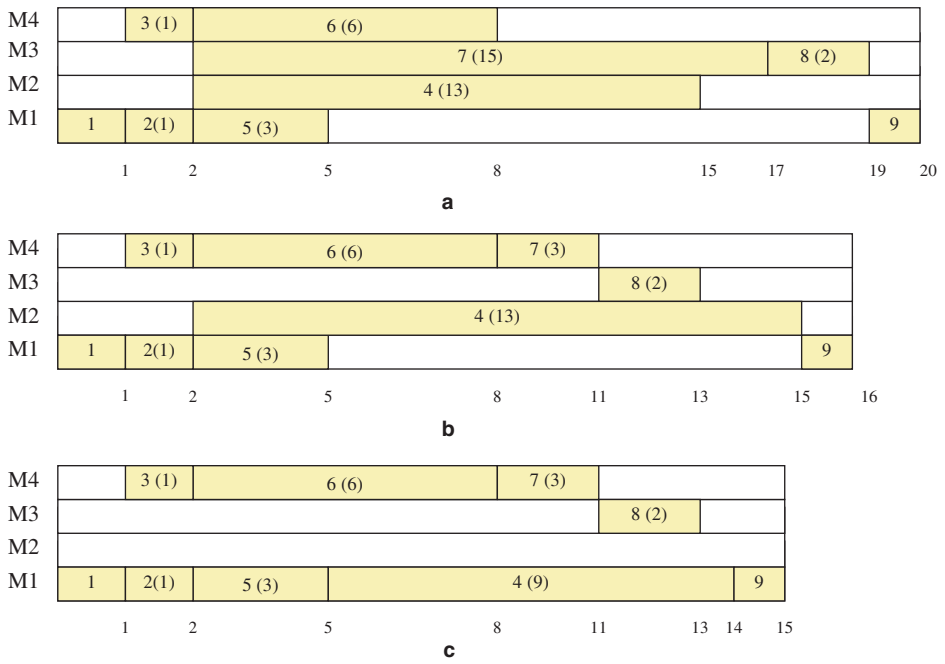

Fig. 5. Gantt charts for the task graph of Fig. 4 for HLF task-priority rule and different machine-priority rules. (a) HLF/FAMF. (b) HLF/FAMF-CW-1. (c) HLF/FAMF-CW-2.

Fig. 2 gives the task-priority rule parameters and the rankings due to different task-priority rules for the example problem in Fig. 1. For the same problem, the solutions range from a makespan of 22–25, giving five different solutions between the twelve algorithms. Fig. 3(a)–(e) give the five different Gantt charts for the various heuristics for this problem. For example, HLF and SLFTF in conjunction with FAMF (Fig. 3a) give a makespan of 23.

In Fig. 3(a) task 3 gets priority over task 2, because the ranked level (or LFT) of task 3 is higher than task 2. In 3(b) task 2 gets the priority of task 3 since it is ranked higher under the TRPT and MSF rules.

For the greedy machine-priority rule, the makespan for the four task-priority rules range from 23 (for HLF and SLFTF) to 25 (for TRPT and MSF). For the non-greedy rules, the makespans range from 22 to 24. Note that although HLF and HTRPT both give a makespan of 22, the schedules are different from each other (Fig. 3(c) and (d)).

Fig. 4 shows a second example, for which Gantt Charts are drawn in Fig. 5(a)–(c). In the second example, different makespans are obtained using the three different machine-priority rules, for the same task

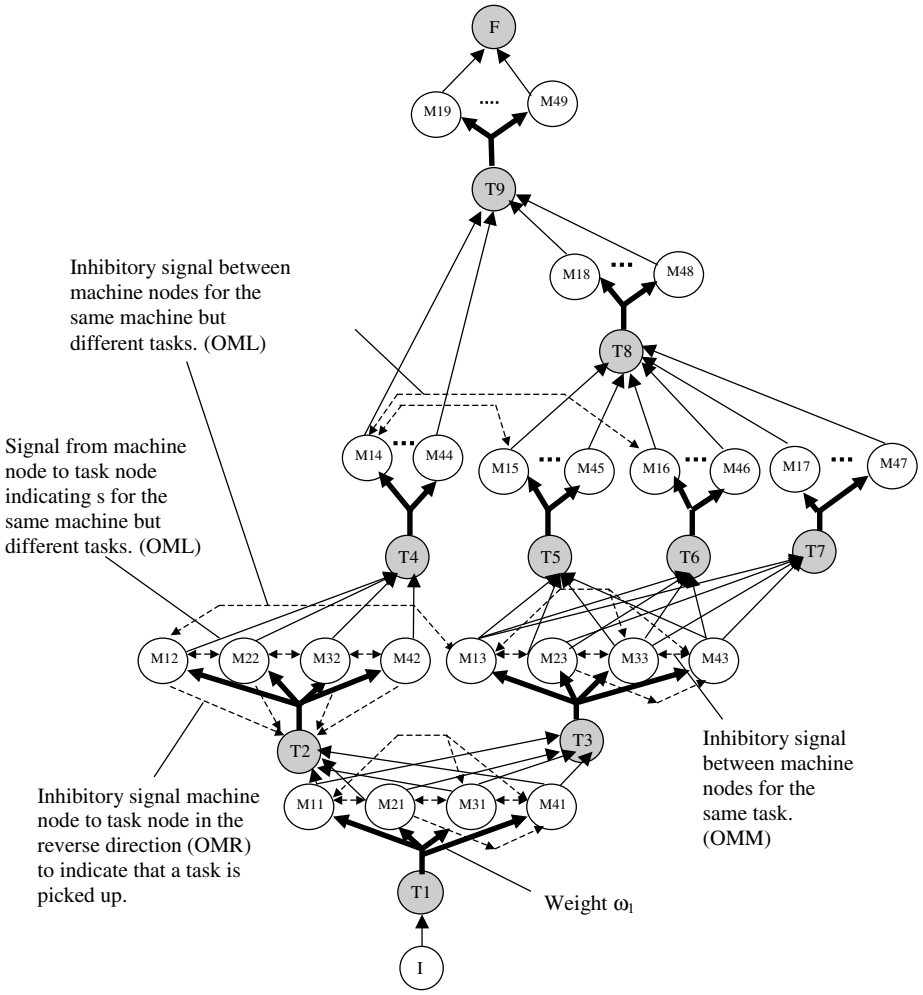

Fig. 6. AugNN architecture for the task graph of Fig. 1.

priority rule (HLF). In this example, the assignment of task 4 is the main difference between the two non-greedy task-priority rules. At time $t = 2$, task 4 has the last priority after tasks 5, 6 and 7. Task 4's first choice is machine 4, which is taken by task 6 for 6 time units. Under the FAMF-CW-1 rule, it is not worth waiting 6 units of time for machine 4 on which the processing time is 8, because the fastest available machine, machine 2 takes only 13 units of time, and $6 + 8 > 13$. However, under FAMF-CW-2 rule, we look at all busy machines, and find that it might be worth waiting for machine 1, because machine 1 is busy for only 3 time units and the processing time on machine 1 is 9. So for machine 1, $3 + 9 < 13$, therefore it is

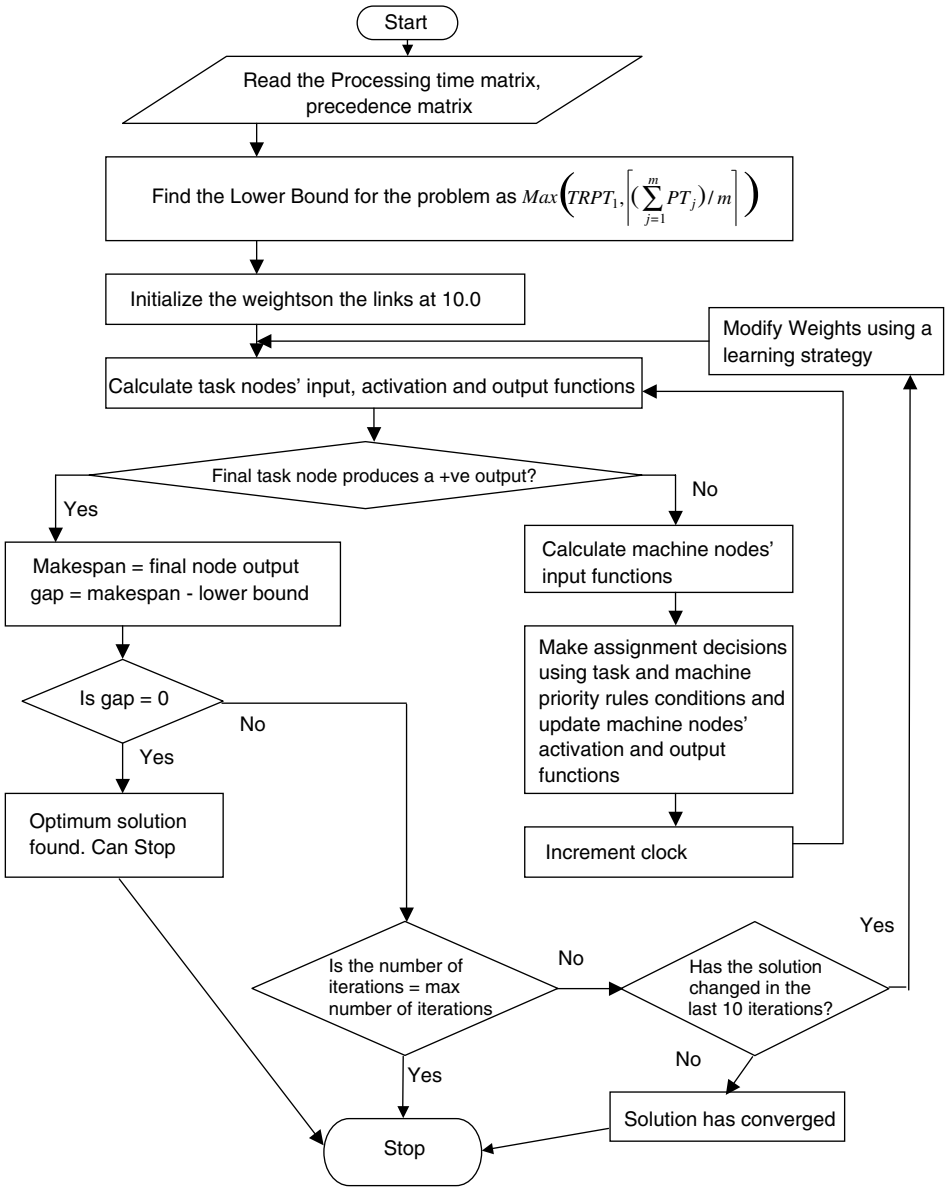

Fig. 7. Flow chart for the AugNN algorithm.

worth waiting for machine 1, which is not the fastest machine for task 4. As a result, we get a better makespan using the FAMF-CW-2 rule.

4. Augmented neural network formulation

In the AugNN formulation, the task graph is converted into a neural network of processing elements (PEs). Each task node in the task graph becomes a PE and each of the task-node PEs is connected to a machine-node PE, one for each machine. The task-node PEs of AugNN maintain the precedence relation of the original task graph. See Fig. 6 for an AugNN representation of the task graph of Fig. 1. We then add an initial node and a final node for ease of formulation. Following each task-node PE are four machine-node PEs corresponding to the four non-identical machines. The first subscript (on the machine node) represents the machine number while the second represents the task to which it is connected from. There are multiple layers of PEs. Each PE has an input, an activation function and an output function. These functions allow us to apply a particular heuristic, such as HLF, HTRPTF etc., to produce a feasible solution. The mathematical details of these functions are given in Appendix 1.

When a task node is ready to start, it sends a signal to the machine nodes it is connected to indicating that it is ready to start. When multiple tasks send signals to multiple machines at the same time, task-priority and machine-priority rules are applied through the activation functions. Assignment takes place based on competition. The assignment changes the state of the various PEs. When a machine node picks up a task, it sends signals to other machine nodes indicating that it has picked up a task. When a machine is done processing a task, it sends a signal to the task ahead that it is done processing, and also to other machines that it can compete for assignments. A task knows it is ready to start if it receives as many signals from machines as there are tasks preceding it. This simple rule helps ensure the precedence constraints of the problem.

There are weights associated with the links from task nodes to machine nodes. The weights are kept the same for the first iteration, so the first iteration gives the same solution as the single-pass heuristic. The weights are then modified in subsequent iterations, using an appropriate learning strategy, such that a neighboring solution is obtained. When applying the heuristic such as HTRPTF for example, the product of weight and TRPT is used to determine priority instead of TRPT alone. Note the presence of two sets of

Table 1
Summary of results for the 12 heuristics for the 100 test problems

Task priority rule	Machine priority rule	Lower bound	Heuristic single-pass solution	AugNN solution	Percent improvement by AugNN over single-pass	Percent gap from LB for single-pass	Percent gap from LB for AugNN
HLF	FAMF	21491	29168	25518	12.51	35.7	18.7
HLF	FAMF-CW-1	21491	25424	24035	5.46	18.3	11.8
HLF	FAMF-CW-2	21491	25483	23989	5.86	18.6	11.6
HTRPTF	FAMF	21491	29409	25443	13.49	36.8	18.4
HTRPTF	FAMF-CW-1	21491	25574	23974	6.26	18.9	11.5
HTRPTF	FAMF-CW-2	21491	25471	23989	5.82	18.5	11.6
SLFTF	FAMF	21491	28734	26285	8.52	33.7	22.3
SLFTF	FAMF-CW-1	21491	25039	24126	3.65	16.5	12.6
SLFTF	FAMF-CW-2	21491	24963	24030	3.74	16.1	11.8
MSF	FAMF	21491	30165	26867	10.93	40.4	25.0
MSF	FAMF-CW-1	21491	26537	24814	6.49	23.5	15.5
MSF	FAMF-CW-2	21491	26491	24744	6.59	23.3	15.1

dotted links between machine nodes (Fig. 6). One set of dotted links is between machines for the same task (shown in the Fig. 6 for tasks T1, T2 and T3, not shown for the rest to avoid cluttering.). The other set of dotted links is between the same machine on different tasks (shown in Fig. 6 between M12 and M13, M14 and M15 and M14 and M16 and not shown for the rest.). These links are used to send inhibitory signals to other nodes to enforce certain constraints. The first set of links is used to ensure that the same task is not assigned to more than one machine. The second set of links is used to ensure that the same machine does not get assigned to more than one task at the same time. For instance, suppose task 2 is assigned to machine 1, while task 3 is not yet assigned, then node M12 will send an inhibitory signals to nodes M22, M32 and M42 to ensure that task 2 does not also get assigned to machines 2, 3 or 4. M12 will also send inhibitory

Table 2A
Makespans for problem sizes 20, 25 and 30 for the three best heuristics

# of tasks	# of m/c	Problem #	Lower bound	Upper bound	HTRPTF/FAMF-CW1		HLF/FAMF-CW-2		HTRPTF/FAMF-CW-2	
					Single-pass heuristic	AugNN	Single-pass heuristic	AugNN	Single-pass heuristic	AugNN
20	2	1	176	191	201	*191*	235	*192*	201	*191*
20	2	2	196	211	237	*230*	259	*230*	237	*230*
20	2	3	215	228	255	*228*	239	*228*	255	*228*
20	2	4	189	202	203	*202*	230	*202*	203	*202*
20	3	1	90	105	114	*109*	128	*109*	114	*109*
20	3	2	97	112	127	*112*	128	*112*	127	*112*
20	3	3	72	88	97	*88*	105	*88*	97	*88*
20	3	4	115	142	145	*142*	145	*142*	145	*142*
20	4	1	84	109	110	*109*	118	*109*	110	*109*
20	4	2	64	76	76	76	107	107	76	76
20	4	3	72	99	101	101	105	*101*	101	101
20	4	4	89	89	90	*89*	89	89	90	89
25	2	1	220	225	249	*225*	237	*225*	249	*225*
25	2	2	185	191	207	*191*	219	*191*	207	*192*
25	2	3	218	249	289	*258*	281	*257*	289	*257*
25	2	4	280	295	332	*295*	295	295	332	*295*
25	3	1	89	103	116	*106*	125	*103*	116	*103*
25	3	2	106	117	121	*117*	132	*117*	121	*120*
25	3	3	155	155	170	*155*	160	*155*	170	*155*
25	3	4	117	135	152	*135*	147	*135*	152	*135*
25	4	1	91	100	108	*100*	108	*100*	108	*100*
25	4	2	58	68	77	*70*	88	*68*	69	69
25	4	3	94	105	111	*105*	106	106	111	*105*
25	4	4	101	112	123	*112*	115	*112*	113	*112*
30	2	1	201	222	227	*222*	249	*225*	227	*225*
30	2	2	284	306	323	*308*	342	*307*	323	*307*
30	2	3	287	302	313	*302*	311	*306*	313	*302*
30	2	4	186	213	219	*213*	213	213	219	*213*
30	3	1	159	179	195	*179*	214	*179*	195	*179*
30	3	2	126	155	173	*159*	176	*155*	173	173
30	3	3	138	155	176	*156*	168	*159*	174	*156*
30	3	4	145	169	198	*169*	186	*172*	198	*172*
30	4	1	80	98	122	*101*	110	*98*	110	*98*
30	4	2	115	149	157	*151*	153	153	157	*153*
30	4	3	76	93	95	93	93	93	95	*93*
30	4	4	167	175	182	182	177	177	182	182

Note: Italicized entries show improvement in makespan by AugNN over one-pass heuristic.

signal to M13 so that task 3 is not assigned to machine 1. Once task 2 is completed on say machine 1, node M12 will send another signal withdrawing the inhibitory signal from M13. In addition to the dotted links between machine nodes, there is a set of dotted links from the machine nodes back to the task node (shown in Fig. 6 from M12, M22, M32 and M42 to T2). This link sends a signal to the task indicating that the task is being processed by that machine.

One pass (iteration) from node I to F generates a feasible solution. After each iteration, the weights are modified using a learning strategy. The change in weights is a function of error. The error is calculated as the difference between the obtained makespan and the lower bound solution. The solution is arrived at after several iterations. The stopping rule is—either a lower-bound solution is reached or that the solution remains unchanged for 10 consecutive iterations or that the total number of iterations is 1000. Fig. 7 gives a flowchart of the AugNN algorithm.

Agarwal et al. (2003, in press) had first proposed the AugNN framework for the task-scheduling problem with identical machines and had obtained good results. In this paper, we adapt the original formulation

Table 2B
Makespans for problem sizes 40 and 50 for the three best heuristics

# of tasks	# of m/c	Problem #	Lower bound	Upper bound	HTRPTF/FAMF-CW1		HLF/FAMF-CW-2		HTRPTF/FAMF-CW-2	
					Single-pass heuristic	AugNN	Single-pass heuristic	AugNN	Single-pass heuristic	AugNN
40	2	1	396	405	443	*405*	426	*405*	443	*405*
40	2	2	351	398	466	*404*	438	*404*	466	*405*
40	2	3	299	305	340	*316*	325	*308*	340	*316*
40	2	4	303	324	345	*324*	354	*325*	345	*327*
40	3	1	166	193	214	*193*	212	*193*	214	*193*
40	3	2	188	251	264	*251*	257	*253*	264	*253*
40	3	3	151	187	208	*188*	196	*188*	208	*188*
40	3	4	189	202	231	*205*	225	*206*	231	*202*
40	4	1	94	121	129	*121*	142	*136*	129	*121*
40	4	2	141	157	177	*157*	177	*157*	177	*157*
40	4	3	152	191	203	*192*	192	*191*	203	*191*
40	4	4	126	159	159	159	167	*163*	161	161
40	5	1	131	133	133	133	144	*138*	138	138
40	5	2	142	160	181	*169*	161	161	161	161
40	5	3	150	169	175	*171*	175	*171*	175	*171*
40	5	4	112	122	122	*122*	146	*128*	127	127
50	2	1	433	445	469	*446*	463	*445*	469	*446*
50	2	2	519	528	557	*528*	548	*530*	557	*530*
50	2	3	409	416	441	*416*	440	*431*	441	*419*
50	2	4	463	485	525	*495*	517	*494*	525	*493*
50	3	1	192	213	237	*213*	250	*215*	237	*213*
50	3	2	200	224	235	*226*	251	*224*	235	*229*
50	3	3	189	213	230	*219*	231	*224*	235	*221*
50	3	4	226	246	263	*247*	271	*246*	263	*246*
50	4	1	135	174	184	*179*	200	*175*	184	*177*
50	4	2	140	167	180	*167*	208	*169*	185	*167*
50	4	3	136	174	190	*175*	204	*174*	184	*176*
50	4	4	134	157	200	*158*	185	*157*	199	*158*
50	5	1	154	189	199	*189*	203	*189*	201	*189*
50	5	2	104	133	152	*138*	144	*134*	152	*135*
50	5	3	104	134	163	*142*	159	*134*	155	*136*
50	5	4	126	146	149	*147*	155	*148*	149	*148*

Note: Italicized cells show improvement in makespan by AugNN over one-pass heuristic.

for the much harder, non-identical machine case. The main adaptation is introducing machine-priority rules on top of task-priority rules. This change is reflected in the machine-node activation function (see Appendix 1). The assignment condition is more complex for this problem. Different assignment conditions are proposed for different priority rules.

5. Computational experience

5.1. Problem generation and experimental setting

Problems were generated for sizes 20–70 tasks and 2–5 machines. For each task size, four precedence relationships were developed at random. For problem sizes of 20, 25 and 30 tasks, 2–4 machines were used. For larger problems, 2–5 machines were used. The processing times were generated randomly from uniform

Table 2C
Makespans for problem sizes 60 and 70 for the three best heuristics

# of tasks	# of m/c	Problem #	Lower bound	Upper bound	HTRPTF/FAMF-CW1		HLF/FAMF-CW-2		HTRPTF/FAMF-CW-2	
					Single-pass heuristic	AugNN	Single-pass heuristic	AugNN	Single-pass heuristic	AugNN
60	2	1	483	493	516	*494*	493	493	516	*494*
60	2	2	573	586	604	*588*	604	*593*	604	*586*
60	2	3	513	521	562	*524*	549	*529*	562	*525*
60	2	4	577	589	643	*592*	605	*598*	643	*598*
60	3	1	302	323	345	*323*	358	*325*	345	*324*
60	3	2	283	309	332	*317*	341	*318*	334	*318*
60	3	3	332	354	379	*359*	377	*356*	379	*356*
60	3	4	282	317	338	*325*	336	*322*	338	*322*
60	4	1	148	174	190	183	184	*183*	190	*185*
60	4	2	183	231	250	*236*	247	*233*	250	*233*
60	4	3	141	176	203	*176*	198	*178*	190	*185*
60	4	4	148	179	183	183	185	*179*	183	183
60	5	1	145	178	186	*181*	184	*183*	186	*184*
60	5	2	108	133	164	*138*	140	*138*	164	*140*
60	5	3	136	161	182	*169*	191	*169*	182	*169*
60	5	4	132	156	169	*159*	173	*161*	173	*158*
70	2	1	678	678	722	*678*	695	*679*	722	*678*
70	2	2	576	609	671	*609*	641	*610*	671	*609*
70	2	3	580	599	624	*609*	622	*607*	624	*599*
70	2	4	592	601	622	*601*	603	*601*	622	*601*
70	3	1	324	344	359	*344*	378	*346*	359	*345*
70	3	2	345	396	412	*401*	427	*404*	412	*400*
70	3	3	316	327	352	*334*	344	*336*	352	*333*
70	3	4	321	338	359	*358*	355	*347*	359	*354*
70	4	1	195	228	251	*229*	231	*228*	231	*228*
70	4	2	189	235	270	*235*	258	*238*	258	*235*
70	4	3	206	230	262	*234*	257	*231*	262	*233*
70	4	4	197	230	247	247	282	*237*	242	242
70	5	1	131	163	188	*166*	169	*167*	177	*165*
70	5	2	151	208	232	*215*	230	*220*	219	*218*
70	5	3	178	207	219	*208*	219	*210*	213	*211*
70	5	4	134	168	183	183	179	*175*	197	*181*

Note: Italicized cells show improvement in makespan by AugNN over one-pass heuristic.

distribution in the range of 1–50. A total of 100 problems were thus generated. This approach of generating problems is consistent with Kasahara and Narita's (1984) approach, and adequately represents real-world problems. Coding was done in Visual Basic 6.0 and run on a Pentium 4 machine. We ran AugNN for 1000 iterations. The initial weights were set at 10.00 and learning rate at 0.1. These parameters were arrived at after some initial testing.

5.2. Discussion of results

Table 1 summarizes the results of all 100 problems for all twelve heuristic combinations. The greedy machine-priority rule (FAMF) performed worse than the non-greedy (FAMF-CW-1 and CW-2), for each of the four task-priority rules. The gaps from the lower bound for the greedy rule for single-pass heuristics ranged from 33.7% (SLFTF) to 40.4% (MSF), compared to 16.1% to 23.5% for non-greedy. For three of the four task-priority rules, FAMF-CW-2 gave marginally better results than FAMF-CW-1. AugNN reduced the gaps significantly. For example, for HTRPTF/FAMF, the gap was reduced from 36.8% down to 18.4%, a reduction of 50%. For HTRPTF/FAMF-CW-1, the gap was reduced from 18.9% down to 11.5%, a 39.2% reduction. The reductions in gap due to AugNN ranged from 24.4% to 50% for the twelve heuristic combinations. The improvement in makespan due to AugNN ranged from 3.65% to 13.49%. Depending on the application, an improvement of 13.49% in makespan or a reduction in the gap by 50% may be considered significant and worth the effort of implementing the AugNN procedure. For example, in the manufacturing environment, an improvement in makespan of 12% translates into an increased capacity by 12% without increasing any fixed costs. Assuming adequate demand for the product, an increased capacity would translate into higher profits. It only takes a few seconds to apply the AugNN procedure. If the makespan for a one-time scheduling problem is in seconds or milliseconds, then using a single-pass heuristic would be preferred.

The three best heuristic combinations out of 12 were HTRPTF/FAMF-CW1, HLF/FAMF-CW-2 and HTRPTF/FAMF-CW-2. Tables 2A, 2B and 2C give the results for these three heuristics for each of the 100 problems. The table lists the makespans obtained from the single-pass heuristic as well as from AugNN. The problems where AugNN improved upon the makespan are italicized. Table 3 summarizes

Table 3
(i) Cases where improvement may be possible, (ii) cases where ANN improved over heuristic and (iii) percent cases where ANN improved over heuristic

	Number of machines				Total
	5	4	3	2	
Total problems	16	28	28	28	100
Improvement possible in (number of problems)					
HTRPTF/FAMF-CW-1	16	28	28	28	100
HLF/FAMF-CW-2	16	27	28	28	99
HTRPTF/FAMF-CW-2	16	28	28	28	100
Number of cases where improvement occurred					
AugNN HTRPTF/FAMF-CW-1	15	20	28	28	91
AugNN HLF/FAMF-CW-2	15	22	28	25	90
AugNN HTRPTF/FAMF-CW-2	13	21	27	28	89
Percent of cases where improvement occurred					
AugNN HTRPTF/FAMF-CW-1	81.3	78.5	100	100	93
AugNN HLF/FAMF-CW-2	93.8	81.5	100	89.3	90.9
AugNN HTRPTF/FAMF-CW-2	75	75	100	100	88

Table 4
Computational times[a] (in seconds)

	Total for all 100 problems		Average per problem	
	Heuristic	ANN	Heuristic	ANN
AugNN HTRPTF/FAMF-CW-1	1.56	143.07	0.0156	1.43
AugNN HLF/FAMF-CW-2	1.56	142.18	0.0156	1.42
AugNN HTRPTF/FAMF-CW-2	1.56	140.23	0.0156	1.40

[a] Includes IO and processing time.

the improvements in terms of number of problems for the three best heuristic combinations. Our best heuristic combination (HTRPTF/FAMF-CW1) found improvement in 91% of the problems. The next two heuristics found improvement in 90% and 89%, respectively.

Table 4 summarizes the computing times for single-pass and AugNN runs for our three best heuristic combinations. In roughly 1.4 seconds, on average, AugNN is able to provide the solutions. Single-pass heuristics are of course extremely fast, at 0.0156 seconds per problem.

It is to be noted that AugNN consistently gave significant improvements upon each of the single-pass heuristic combinations. So, if one comes across a better base heuristic than the ones proposed in this paper, we can count on AugNN to provide some improvement over that heuristic. We could not compare our results with other meta-heuristics due to lack of application of other meta-heuristics to this type of problem.

6. Summary and conclusions

In this paper, the augmented-neural-network (AugNN) approach is applied to the task-scheduling problem with non-identical machines. The AugNN approach was first applied to task-scheduling problem for the identical machines case (Agarwal et al., 2003). This approach applies a base single-pass heuristic and tries to improve upon the single-pass solution through a weight modification learning strategy using neural-network principles. We propose 12 base heuristics for this problem and show improvements on each of the 12 through AugNN. The 12 heuristics are essentially combinations of four task-priority rules and three machine-priority rules. The three machine-priority rules used in this paper have been proposed for the first time. One of them is a greedy rule, while the other two are non-greedy. The non-greedy rules performed significantly better than the greedy rule. We propose a modification to the original AugNN formulation (Agarwal et al., 2003) which allows implementation of a combination of task-priority and machine-priority rules. We also provide empirical results on 100 randomly generated problems. We found that AugNN reduces the single-pass heuristics gaps from the lower bound by 24.4–50%. The best single-pass heuristic gave a gap of 16.1% from the lower bound. The best AugNN results gave us a gap of 11.5%.

This research makes two-fold contribution. First is the development of machine-priority rules for this type of problem. Second, the AugNN formulation has been modified to handle a combination of task and machine priority rules. Two main results emerged. One, non-greedy machine-priority rules are significantly better than the greedy ones and two, AugNN approach shows significant reduction in gaps from single-pass heuristics, in a very short time.

A number of future research issues emerge from this study. Since the AugNN approach seems to work well for the makespan minimization problem, it can be applied to problems with other objectives, such as minimizing tardiness, and for other constraints, such as preemption of tasks. The multi-mode resource-constrained project scheduling problem can also be attempted using the AugNN approach, since the problem bears some resemblance with the problem in this paper. Further, since other meta-heuristics such as genetic

algorithms and tabu search have not been applied to this problem, they can be applied and their results compared against the results obtained in this work.

Appendix 1

In this appendix, we describe the input, output and activation functions for the task and machine nodes. We first list the notation used in our functions.

Notation

n	number of tasks
m	number of machines
k	iteration number
T	set of tasks $= \{1, \ldots, n\}$
M	set of machines $= \{1, \ldots, m\}$
T_j	jth task node, $j \in T$
M_{ij}	node for machine i connected from T_j, $i \in M$, $j \in T$
RT_j	remaining time of task T_j, $j \in T$
ω_j	weight on the outgoing link from T_j
ω_{m}	weight on the links between machine nodes
α	learning coefficient
ε_k	error in iteration k
t	elapsed time in the current iteration
I	initial dummy task node
F	final dummy task node
τ_j	threshold value of T_j = # of tasks immediately preceding task j, $j \in T \cup F$
ST_j	start time of T_j, $j \in T$
PT_{ij}	processing time of machine node M_{ij}, $i \in M$, $j \in T$
LST_j	latest start time of T_j, $j \in T$
PR_j	set of tasks that immediately precede task j, $j \in T \cup F$
NPR	set of tasks with no preceding tasks $\{T_j \mid \mathrm{PR}_j$ is an empty set$\}$, $j \in T$
SU_j	set of tasks that immediately succeed task j, $j \in T$
Win_j	winning status of T_j, $j \in T$

Following are all functions of elapsed time t:

$\mathrm{IT}_j(t)$	input function value of task nodes, $j \in I \cup T \cup F$
$\mathrm{IM}_{ij}(t)$	input function value of machine nodes, $i \in M$, $j \in T$
$\mathrm{OT}_j(t)$	output function value of task nodes, $j \in I \cup T \cup F$
$\mathrm{OMF}_{ijp}(t)$	output of machine node M_{ij} to task T_p in the forward direction, $i \in M$, $j \in T$, $p \in \mathrm{SU}_j$
$\mathrm{OMR}_{ij}(t)$	output of machine node M_{ij} to task T_j in reverse direction, $i \in M$, $j \in T$
$\mathrm{OML}_{ijp}(t)$	output of machine node M_{ij} to M_{ip} in lateral direction, $i \in M$, $j, p \in T$, $j \neq p$
$\mathrm{OMM}_{ioj}(t)$	output of machine node M_{ij} to M_{oj} in lateral direction, $i, o \in M$; $i \neq o$, $j \in T$
$\theta T_j(t)$	activation function of task nodes, $j \in T$
$\theta M_{ij}(t)$	activation function of machine nodes M_{ij}, $i \in M$, $j \in T$
$\mathrm{assign}_{ij}(t)$	$=1$ if machine Mi assigned to task T_j
$S(t)$	set of tasks that can start at time t. $S(t) = \{T_j$—$\mathrm{OT}_j(t) = 1\}$
$\mathrm{MA}(t)$	set of machines available at time t

MAB$j(t)$ set of available machines whose processing times on job j is less than or equal to the sum of (waiting time for the fastest machine and the processing time on the fastest machine)

The input to the algorithms is the precedence matrix and the task processing times (PT_{ij}) matrix. As mentioned earlier, the solution is obtained iteratively. Before the first iteration, however, there are some preliminary steps to be performed. These will be described now followed by a description of the input functions, output functions, activation functions and the learning strategies.

A.1. The preliminary steps

The following steps are performed before proceeding with the iterations.

1. Total remaining processing times ($TRPT_j$), are calculated for every task. Calculation of TRPTs assumes that each task will get the fastest machine.
2. Lower bound is established as

$$\text{Lower bound} = \max\left(\text{TRPT}_1, \left\lceil \sum_{j=1}^{m} \text{PT}_j \Big/ m \right\rceil\right).$$

3. Weights (ω_j) are initialized at 10.00. The initial value is not very critical as long as it is the same for all links in the beginning. The value 10 was used because it seems to work well. A choice of 100 would be equally good.
4. Calculate the threshold of each task τ_j. The threshold of task j is defined as the number of tasks immediately preceding task j. For tasks with no preceding tasks, the threshold is assigned to be 1 because a dummy initial task with zero processing time is used. The threshold value is used to determine when a task is ready to start.

After these preliminary steps, iterations are performed. The input, activation and output functions of the PE's determine the flow of each iteration. End of iteration routines will be discussed later.

A.2. The input, activation and output activation functions

The neural network algorithm can be described with the help of the learning strategy and the input, activation and output functions for the task nodes and the machine nodes. The formulation for all the heuristics is the same except for the assignment rule or the Activation state of machine nodes. The following functions are described for the HRTFFM heuristic.

A.3. Task nodes

We first describe the input functions, activation states and output functions of the T nodes:

A.3.1. Input function

We describe the initial values, i.e. at time $t = 0$.

$$\text{IT}_j(0) = 0 \quad \forall j \in I \cup T \cup F.$$

For nodes with no preceding tasks

$$\text{IT}_j(0) = \text{OT}_I(0) = \text{IT}_I(0) = 1 \quad \forall j \in \text{NPR}.$$

All tasks with no preceding constraints get a starting signal.
Threshold of all tasks in NPR is 1 because we have added the I node.
For all other tasks i.e. $\forall j \notin \text{NPR}\ \Lambda t > 1$

$$\text{IT}_j(t) = \text{IT}_j(t-1) + \sum_i \sum_q \text{OMF}_{iqj}(t) \quad \forall i \in M,\ \ q \in PR_j,\ \ j \in T \cup F.$$

OMF in the above equation, described later, represents a signal of 1 sent by a machine node to a succeeding task after completing its current task. $\text{IT}_j(t)$, is thus a measure of how many tasks preceding T_j are completed at time t.

A.3.2. Activation states

The activation states of the task nodes tell us about the ready state or the state of completion of a task. A task is either unable to start due to precedence constraints (state 1), or is ready to start (state 2) or is currently processing (state 3) or is complete (state 4). These states are a function of elapsed time t. These are now mathematically defined.

Task nodes' Initial Activation State (i.e. at $t = 0$) is 1. $\forall i \in M, j \in T$,

$$\theta T_j(t) \begin{cases} 1 \text{ if } \text{IT}_j(t) < \tau_j & \text{cannot start,} \\ 2 \text{ if} (\theta T_j(t-1) = 1 \vee 2) \Lambda \text{IT}_j(t) = \tau_j & \text{can start,} \\ 3 \text{ if} (\theta T_j(t-1) = 2 \vee 3) \Lambda \sum_i \text{OMR}_{ij}(t) < 0 & \text{processing,} \\ 4 \text{ if } \theta T_j(t-1) = 4 \vee (\theta T_j(t-1) = 3 \Lambda \sum_i OMR_{ij}(t) = 0) & \text{task completed.} \end{cases}$$

Note: $\forall j \in \text{NPR}$, $\tau_j = 1$.

Since $\text{IT}_j(t)$ can be interpreted as the number of preceding tasks (of T_j) completed at time t and τ_j is the total number of preceding tasks (of T_j), when they equal, the task T_j is ready to start. As long as $\text{IT}_j(t) < \tau_j$, T_j cannot start. Once the assignment takes place the machine to which the task is assigned sends a negative signal OMR. So when OMR is negative, we know that the task has been assigned. When the negative signal is withdrawn, we know that the task is completed.

A.3.3. Output function

$$\text{OT}_j(t) = \begin{cases} 1 & \text{if } \theta T_j(t) = 2, \\ 0 & \text{otherwise.} \end{cases}$$

When a task is ready to start (state 2), the task sends a unit signal to the machines it is connected to.
F-Node

$$\text{OT}_F(t) = \begin{cases} t & \text{if } \text{IT}_F(t) = \tau_F, \\ 0 & \text{otherwise.} \end{cases}$$

When task F's input equals its threshold, the scheduling for the task graph is complete and the time at which this condition occurs is the makespan of the schedule.

A.4. Machine nodes

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

A.4.1. Input

$$\mathrm{IM}_{ij}(t) = \mathrm{OT}_j(t)^*\omega_j + \sum_{q\varepsilon S(t)} \mathrm{OML}_{iqj}(t)^*\omega_{\mathrm{m}} + \sum_{i^*} \mathrm{OMM}_{i^*ij}(t)^*\omega_{\mathrm{m}} \quad \forall i \in M,\ j \in T,\ i^* \neq i$$

ω_{m} is fixed weight link between machines and is large to suppress output of task if machine is busy or assigned or task is assigned to other machine.

A.4.2. Activation states

The activation states of the machine nodes tell us the state of availability, assignment and processing. If a machine is available it is in state 1. If it gets assigned it is in state 2, if it is processing a task, it is state 3. When it finishes processing a task, it is in state 4. If a machine node of the same machine is assigned to another task it is in state 5. For example, if M12 is in state 3 (i.e. machine M1 is currently processing task 2), then M13 and M14 are in state 5 because they also represent machine 1, which is busy with task 2. When a machine node is done processing a task it reaches state 6. Before we explain the states mathematically, we describe the assignment process first.

$$\text{Let } \chi_{ij}(t) = \mathrm{IM}_{ij}(t)^*\mathrm{TaskPriorityRuleParameter}_j \quad \forall i \in M, j \in T.$$

The assignment process depends on the task-priority rule and the machine-priority rule.

For FAMF machine-priority rule,

$$\mathrm{assign}_{ij}(t) = \begin{cases} 1 & \text{if } \chi_{ij}(t) = \max[\chi_{it}(t)|\forall T_j \in S(t)] \wedge \chi_{ij}(t) > 0 \wedge j = \arg\min(\mathrm{PT}_{ij}), j \in MA(t), \\ 0 & \text{otherwise.} \end{cases}$$

For FAMF-CW-1 priority rule, define

$\mathrm{PT}_{\mathrm{FM},i}$ is processing time of the fastest machine for task i.
$W_{\mathrm{FM},i}$ is wait on the fastest machine for task i.
$\mathrm{PT}_{\mathrm{FAM},i}$ is processing time on the fastest machine for task i.

$$\mathrm{assign}_{ij}(t) = \begin{cases} 1 & \text{if } \chi_{ij}(t) = \max[\chi_{it}(t)|\forall T_j \in S(t)] \wedge \chi_{ij}(t) > 0 \wedge (W_{\mathrm{FM}} + \mathrm{PT}_{\mathrm{FM}}) > \mathrm{PT}_{\mathrm{FAM}}, \\ 0 & \text{otherwise.} \end{cases}$$

For FAMF-CW-2 priority rule, define

$B(t)$ is set of busy machines at time t.
W_j is wait on a busy machine j.

$$\mathrm{assign}_{ij}(t) = \begin{cases} 1 & \text{if } \chi_{ij}(t) = \max[\chi_{it}(t)|\forall T_j \in S(t)] \wedge \chi_{ij}(t) > 0 \wedge \min\limits_{j \in B(t)}(W_j + \mathrm{PT}_{ij}) > \mathrm{PT}_{\mathrm{FAM}}, \\ 0 & \text{otherwise.} \end{cases}$$

The task-priority rule part of the heuristic is enforced by the condition $\chi_{ij}(t) = \max[\chi_{ij}(t)|\forall T_j \in S(t)] \Lambda \chi_{ij}(t) > 0$.

The machine-priority part of the heuristic is enforced by the last part of the assignment condition. For example, for FAFM, it is $j = \mathrm{argmin}(\mathrm{PT}_{ij}), j \in \mathrm{MA}(t)$ and for FAFM-CW-1, it is $W_{\mathrm{FM}} + \mathrm{PT}_{\mathrm{FM}} > \mathrm{PT}_{\mathrm{FAM}}$.

If $\mathrm{assign}_{ij}(t) = 1$, then $\mathrm{ST}_j = t$.

If $|S(t)| > |\mathrm{MA}(t)|$ then if $\mathrm{assign}_{ij}(t) = 1$ then $\mathrm{Win}_j = 1$.

If competition takes place, we need to keep track of which task won the competition.

Task-priority-rule parameter depends on the rule. For example, it is Level for HLF, TRPT for HTRPTF, Slack for MSF and LFT for SLFTF rule. Note that for MSF and SLFTF rules, the task parameter should be considered negative because the assignment is based on $\max \chi_{ij}(t)$.

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

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

$$\theta M_{ij}(t) = \begin{cases} 1 & : \text{machine available,} \\ 2 \ \text{ if } (\theta M_{ij}(t-1) = 1 \vee \theta M_{ij}(t) = 1) \wedge \text{assign}_{ij}(t) = 1 & : \text{machine busy(just assigned)} \\ 3 \ \text{ if } (\theta M_{ij}(t-1) = 2 \vee 3) \wedge (t < \text{ST}_i + \text{PT}_i) & : \text{machine busy(in process),} \\ 4 \ \text{ if } (\theta M_{ij}(t-1) = 3) \wedge (t = \text{ST}_i + \text{PT}_i) & : \text{machine just released,} \\ 5 \ \text{ if } (\theta M_{ij}(t-1) = 1) \wedge \left(\sum_{q \in S(t)} \text{OML}_{iqj}(t)^* \omega_{\text{m}} < 0 \right) & : \text{assigned to another task,} \\ 6 \ \text{ if } (\theta M_{ij}(t-1) = 4) & : \text{machine } i \text{ finished process on task } j, \\ 1 \ \text{ if } (\theta M_{ij}(t-1) = 1 \vee 5) \wedge \left(\sum_{q \in S(t)} \text{OML}_{iqj}(t)^* \omega_{\text{m}} = 0 \right) & : \text{machine } i \text{ not assigned to any other task,} \\ 1 \ \text{ if } (\theta M_{ij}(t-1) = 1) \wedge \left(\sum_{i^*} \text{OMM}_{i^* ij}(t)^* \omega_{\text{m}} < 0 \right) & : \text{task assigned to another machine.} \end{cases}$$

A.4.3. Output

The output of the machine node indicates a signal to the tasks ahead that their preceding tasks are complete and also a signal to other machine nodes that they are now available.

$$\text{OMF}_{ijp} = \begin{cases} 1 & \text{if } \theta M_{ij}(t) = 4 \\ 0 & \text{if } \theta M_{ij}(t) = 1,2,3,5,6 \end{cases} \quad \forall i, j \in T, \quad \forall p \in SU_i,$$

$$\text{OMR}_{ij} = \begin{cases} -1 & \text{if } \theta M_{ij}(t) = 2,3 \\ 0 & \text{if } \theta M_{ij}(t) = 1,4,5,6 \end{cases} \quad \forall i, j \in T,$$

$$\text{OML}_{ijp} = \begin{cases} -1 & \text{if } \theta M_{ij}(t) = 2,3 \\ 0 & \text{if } \theta M_{ij}(t) = 1,4,5,6 \end{cases} \quad \forall i, j \in T, \forall p \in S(t), p \neq j,$$

$$\text{OMM}_{ii^*j} = \begin{cases} -1 & \text{if } \theta M_{ij}(t) = 2,3 \\ 0 & \text{if } \theta Mij(t) = 1,4,5,6 \end{cases} \quad \forall i, j \in T, i^* \neq i.$$

The output of F represents the makespan and the $\text{assign}_{ij}(t)$ 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 period.

A.4.4. 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 different machines should be winners during subsequent iteration. Since the machine with the highest value of χ_j, is the winner, an increase of weights will make the machine more likely to win

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 the level of the task. Keeping these points in mind, the following learning strategy for the links that determine priority is used.

$$\text{Winning tasks: If } \mathrm{Win}_j = 1 \text{ then } (\omega_j)_{k+1} = (\omega_j)_k - \alpha^* \mathrm{RT}_j^* \varepsilon_k \quad \forall j \in T,$$

$$\text{Non-winning tasks: If } \mathrm{Win}_j = 0 \text{ then } (\omega_j)_{k+1} = (\omega_j)_k + \alpha^* \mathrm{RT}_j^* \varepsilon_k \quad \forall j \in T.$$

A.4.5. End of iteration routines

1. Calculate the error i.e., the difference between obtained makespan and the lower bound. The lower bound was obtained during the preliminary steps.
2. Store the best solution so far.
3. Sense convergence i.e., if the solution has not changed in the last 10 iterations, stop the program. The value of 10 was also arrived at after some computational experience.
4. If the number of iterations is greater than 1000, stop the program. Most solutions were obtained in less than 100 iterations. So, there is enough margin of safety in running the program for up to 1000 iterations.

If continuing with the next iteration, modify weights using the learning strategy.

References

Adam, T.L., Chandy, K.M., Dickson, J.R., 1974. A comparison of list schedules for parallel processing systems. Communications of the ACM 17 (12), 685–690.

Agarwal, A., Jacob, V.S., Pirkul, H., 2003. Augmented neural networks for task scheduling. European Journal of Operational Research 151 (3), 481–502.

Agarwal, A., Jacob, V.S., Pirkul, H., in press. Improved augmented neural networks for scheduling problems. INFORMS Journal on Computing.

De, P., Morton, T.E., 1980. Scheduling to minimize makespan on unequal parallel processors. Decision Sciences 11, 586–602.

El-Rewini, H., Ali, H.H., Lewis, T., 1995. Task scheduling in multiprocessing systems. Computer Decisions, 27–37.

Epstein, L., Sgall, J., 2000. A lower bound for online scheduling on uniformly related machines. Operations Research Letters 26 (1), 17–22.

Foo, Y.P.S., Takefuji, Y., 1988. Stochastic neural networks for solving job-shop scheduling: Part 1, problem representation. Proceedings of Joint International Conference on Neural Networks, Vol. 2, pp. 275–282.

Graves, S.C., 1981. A review of production scheduling. Operations Research 29 (4), 646–675.

Gonzalez, T.F., Sahni, S., 1978. Preemptive scheduling of uniform processor systems. Journal of ACM 25, 92–101.

Haldun, A., Bhattacharya, S., Koehler, G.J., Snowdon, J.L., 1994. A review of machine learning in scheduling. IEEE Transactions on Engineering Management 41 (2), 165–171.

Hopfield, J.J., Tank, D.W., 1985. Neural computation of decisions in optimization problems. Biological Cybernetics 52, 141–152.

Horowitz, E., Sahni, S., 1976. Exact and approximate algorithms for scheduling non-identical processors. Journal of ACM 23, 317–327.

Hu, T.C., 1961. Parallel sequencing and assembly line problem. Operations Research 9, 841–848.

Ibarra, O.H., Kim, C.E., 1977. Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of ACM 24, 280–289.

Kasahara, H., Narita, S., 1984. Practical multiprocessor scheduling algorithms for efficient parallel processing. IEEE Transactions on Computers C-33 (11), 1023–1029.

Panwalker, S.S., Iskander, W., 1977. A survey of scheduling rules. Operations Research 25, 45–61.

Sabuncuoglu, I., Gurgun, B., 1996. A neural network model for scheduling problems. European Journal of Operational Research 93, 288–299.

Sabuncuoglu, I., 1998. Scheduling with neural networks: A review of the literature and new research directions. Production Planning and Control 9 (1), 2–12.

Saltzman, M.J., 1995. Trends in computing technology. OR/MS Today (Dec.), 32–36.
Satake, T., Morikawa, K., Nakamura, N., 1994. Neural network approach for minimizing the makespan of the general job-shop. International Journal of Production Economics 33, 67–74.
Smith, K.A., 1999. Neural networks for combinatorial optimization: A review of more than a decade of research. INFORMS Journal On Computing 11 (1), 15–34.
Smith, K.A., Palaniswamy, M., Krishnamoorthy, M., 1996. A hybrid neural network approach to combinatorial optimization. Computers & Operations Research 23 (6), 597–610.
Smith, K.A., Palaniswamy, M., Krishnamoorthy, M., 1998. Neural techniques for combinatorial optimization with applications. IEEE Transactions on Neural Networks 9 (6), 1301–1318.