ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

European Journal of Operational Research 151 (2003) 481–502

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Augmented neural networks for task scheduling

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

[a] *Department of Decision and Information Sciences, Warrington College of Business Administration, University of Florida, Gainesville, FL 32611-7169, USA*

[b] *School of Management, University of Texas at Dallas, Richardson, TX 75083-0688, USA*

Received 8 November 2000; accepted 26 June 2002

Abstract

We propose a new approach, called Augmented Neural Networks (AugNN) for solving the task-scheduling problem. This approach is a hybrid of the heuristic and the neural networks approaches. While retaining all the advantages of the heuristic approach, AugNN incorporates learning, to find improved solutions iteratively. This new framework maps the problem structure to a neural network and utilizes domain specific knowledge for finding solutions. The problem we address is that of minimizing the makespan in scheduling n tasks on m machines where the tasks follow a precedence relation and task pre-emption is not allowed. Solutions obtained from AugNN using various learning rules are compared with six different commonly used heuristics. AugNN approach provides significant improvements over heuristic results. In just a few iterations, the gap between the lower bound and the obtained solution is reduced by as much as 58% for some heuristics, without any increase in computational complexity. While the heuristics found solutions in the range of 5.8–26.9% of the lower bound, on average, AugNN found solutions in the range of 3.3–11.1%, a significant improvement.
© 2002 Elsevier B.V. All rights reserved.

Keywords: Scheduling; Neural networks; Heuristics

1. Introduction

Neural networks (NNs) have been applied extensively to a variety of problems in various disciplines such as data mining, pattern recognition and classification (Bishop, 1995). They have also been used for solving certain optimization problems. For example Hopfield and Tank (1985) used NNs for solving the TSP. Foo and Takefuji (1988a,b,c), Sabuncuoglu and Gurgun (1996), and Satake et al. (1994) have used the Hopfield and Tank approach for solving the job-shop scheduling problem. Other iterative search techniques such as Simulated Annealing (Steinhofel et al., 1999), Tabu Search (Pezzella and Merelli, 2000; Thomas and Salhi, 1998) and Genetic Algorithms (Candido et al., 1998; Miller et al., 1999; Sakawa and Kubota, 2000) have also been used for various scheduling problems. However, these approaches fail to provide good solutions

* Corresponding author. Tel.: +1-35-239-27300; fax: +1-35-239-25438.
E-mail addresses: aagarwal@ufl.edu (A. Agarwal), hpirkul@utdallas.edu (H. Pirkul), vjacob@utdallas.edu (V.S. Jacob).

0377-2217/$ - see front matter © 2002 Elsevier B.V. All rights reserved.
doi:10.1016/S0377-2217(02)00605-7

as the problem size grows and require thousands of iterations. The CPU times, as a result, tend to be high, in the order of 100's of seconds per problem for small problems of about 10 tasks and 1000's of seconds for problems of about 100 tasks (Sabuncuoglu and Gurgun, 1996).

In this paper, we propose an alternative to the Hopfield and Tank approach of using neural networks for the scheduling problem, in which the behavior of the solution procedure does not deteriorate as the problem size grows. This approach, which we call the Augmented Neural Network (AugNN) approach, is a hybrid of the heuristic and the neural network approaches. The approach maps the problem structure on the network and utilizes domain specific knowledge to improve performance. In less than 100 iterations, excellent results are obtained. To demonstrate this new framework we apply it to the task-scheduling problem. The approach, however, can be used for other more complex scheduling problems such as the open shop and the job-shop scheduling problems.

The scheduling problem has been stimulated by the increasingly competitive world markets for efficiently manufactured goods, as well as for efficiency in distributed computing environments. Lo and Bavarian (1993) argue that there is a great need for better scheduling algorithms and heuristics. The task-scheduling problem occurs in a variety of situations, ranging from project management to scheduling of tasks in a multiprocessor environment. In fact, the task-scheduling problem is at the heart of many scheduling problems, both in manufacturing and computing. Improvements in the task-scheduling problem will easily translate to other types of scheduling problems.

The problem we address is that of minimizing the makespan in scheduling n tasks on a set of m identical machines. The tasks follow precedence constraints and cannot be pre-empted. This problem has been studied extensively by a number of researchers (Hu, 1961; Coffman, 1976; Graham et al., 1979; Kasahara and Narita, 1985). Since the problem is NP-Hard, solution procedures require excessive computational effort for larger problem instances. Therefore, the problem is generally solved using heuristics, such as *Highest Level First* (*HLF*), *Highest Level with Estimated Time First* (*HLETF*), *Critical Path/Most Immediate Successor First* (*CP/MISF*), etc. No single heuristic works best for all problems. On average, *HLETF* and *CP/MISF* (see Kasahara and Narita, 1985) are known to work the best. Our experience during this research was that HLETF worked the best, followed by HLF. Neural network approaches based on Hopfield networks have been applied to the job-shop scheduling problems but not for the task-scheduling problem. With little modification, our approach can handle the job shop problem as well.

The proposed AugNN approach makes use of the characteristics of NNs, yet is different from the prior neural network approaches used for solving optimization problems. A critical feature of our approach is the one-to-one correspondence between the problem structure and the NN structure, thus, allowing the NN to be augmented by the relevant domain specific knowledge in solving the problem. We use six different heuristics and two different learning rules to test the effectiveness and robustness of the AugNN approach. For each of the six heuristics, significant improvements were obtained using AugNN. In some cases, the gap between the lower bound solution and the heuristic solution was reduced by as much as 58%. Also, while the heuristics found solutions in the range of within 5.8–26.9% of the lower bound, on average, AugNN found solutions in the range of 3.3–11.1%, a significant improvement.

This research, therefore, makes a twofold contribution. First, it presents a new approach for solving the task-scheduling problem. Second, it presents a new way of using neural networks, one that allows embedding of relevant domain specific knowledge and makes use of the network structure to represent the problem itself. This framework can be applied to other optimization problems especially those that lend themselves to network structures.

The rest of the paper is organized as follows. In Section 2, the literature review for the task-scheduling problem is provided. Existing neural network approaches are also discussed. Section 3 describes the task-scheduling problem in order to better understand the proposed framework. Section 4 describes the proposed framework in detail. Section 5 includes the computational experience and a discussion of the results. Section 6 provides the summary and conclusions.

2. Literature review

Sabuncuoglu (1998) provides an excellent review of literature in the use of neural networks in Scheduling. They point out the limitations of the current approaches based on backpropagation neural networks for optimization problems and encourage research on alternative approaches. Hu (1961) had pioneered the idea of a heuristic approach for solving scheduling problems for the identical machine case with precedence constraints. He developed "Level-Scheduling", a critical path approach for greedy heuristics. Most of the heuristic scheduling procedures described in the literature are based on "priority dispatching" rules (Adams et al., 1988). Panwalker and Iskander (1977) provide a survey of dispatching rules. For a more recent comparison study of dispatching rules for job shop scheduling, see Rajendran and Holthaus (1999).

Adams et al. (1988) were amongst the first to propose an iterative (or multi-pass) procedure for solving scheduling problems. They proposed a shifting bottleneck procedure for the job shop problem. They applied their shifting bottleneck heuristic to 59 problems with size ranging from 20 operations (5 jobs, 4 machines) to 500 operations (50 jobs, 10 machines). Uzsoy and Wang (2000) extend the study of shifting bottleneck procedure.

Hopfield and Tank (1985) pioneered the application of neural networks for solving combinatorial optimization problems. They solved the traveling salesman problem by defining an 'Energy Function', which captured the constraints of the problem. A neural network was used to minimize the value of this 'Energy Function', thus giving the solution. The Hopfield network has been applied to the job-shop scheduling problem by several researchers. Foo and Takefuji (1988a,b,c) were amongst the first. They, however, use a very small 2-job, 3-machine problem. The computational complexity is $O(m^2n^2 + mn)$. Lo and Bavarian (1993) use an extension of the Hopfield network to solve the scheduling problem with deadline requirements. They also considered a small problem with 10 tasks and 3 machines. Their computational complexity was $O(nmk + \alpha)^2$, where k is the processing time. The number of iterations required for convergence was as high as 3000. Satake et al. (1994) have used a modified Hopfield network to solve bigger problems, up to 14-job, 7-machines. They change the threshold values at each transition of neurons in order to make a non-delay schedule in addition to incorporating job and shop-related constraints. The computation time remains poor at about 700 s for a relatively small, 14-job, 7-machine problem. Lo and Hsu (1993) propose a modified Hopfield network to improve computational time. Still, a 35-job 4-machine problem took an average of 150 seconds. Computational complexity is not reported. Sabuncuoglu and Gurgun (1996) have reported yet another variation of the Hopfield network. Computational times have not improved. For example, a 15-job 5-machine problem took about 2300 s to reach a near optimum solution.

The energy function approach therefore, has the following problems. They do not work well on large problems. Most reported results are for extremely small problem sizes. The problem formulation itself is still an art. The computational complexity and thereby CPU times remain high. For a complete review of the application of neural networks in manufacturing, see Zhang and Huang (1995). For a review of machine learning in scheduling, see Haldun et al. (1994) and Sabuncuoglu (1998). Biskup (1999) and Chen et al. (1999) have suggested learning based scheduling. Machine learning techniques other than neural networks have also been used for some scheduling problems. For example Steinhofel et al. (1999) use simulated annealing for the job shop problem and Pezzella and Merelli (2000) and Thomas and Salhi (1998) use Tabu Search in conjunction with the shifting bottleneck procedure for certain job shop problems. Solution times remain in thousands of seconds per problem even on fast machines. The number of iterations needed for solutions remains very high at about 10,000.

The following section describes the problem and briefly discusses the six heuristics used in our paper. Discussion of heuristics will help understand the proposed neural network framework better.

3. The problem

In this paper, we address the problem of scheduling n tasks on m identical machines (or processors). The task graph is directed acyclic, which means there are precedence constraints. Pre-emption of tasks is not allowed. There are an arbitrary number of machines. The objective is to minimize the makespan. Additional constraints are that the same machine may not be assigned to more than one task at the same time and that the same task is not assigned to more than one machine at a time. An instance of a scheduling problem with identical machines is described below (see Fig. 1). There are seven tasks and two machines. This example will be used to explain the proposed AugNN framework in Section 4.

This research uses six different heuristics to test if the hybrid approach makes a difference. They are: (i) *Highest Level First* (*HLF*), (ii) *Highest Level with Estimated Time First* (*HLETF*), (iii) *Critical Path with Most Immediate Successors First* (*CP/MISF*), (iv) *Shortest Path Time* (*SPT*), (v) *Longest Processing Time* (*LPT*), and (vi) Random.

In the *HLF* dispatching rule, task with the highest *Level* is given preference. Ties, if any, are broken at random. *Level* of a task is the number of tasks in the longest remaining path up to the final task, including the task in question. So, for example, the *Level* of the final task is 1 and of the task preceding it is 2 and so on. Although a simple heuristic, *HLF* gives surprisingly good results. The idea is that tasks with higher levels are more likely to be on the critical path. The *HLETF* dispatching rule is a minor variation of the *HLF* rule. 'Level with estimated time', hereafter *LET*, is the estimated processing time of the longest remaining path, including the task in question. Ties are rare under *HLETF* rule, but if they arise they are broken at random. In our experience, the *HLETF* heuristic provides better results than *HLF*.

The *CP/MISF* dispatching rule, as the name suggests, gives priority to tasks on the critical path. In case of ties, priority is given to the task with the most number of immediate successors. In our experience, *CP/MISF* did not provide as good results as *HLF* and *HLETF*, although it performed better than the other three heuristics. The *SPT* rule is the opposite of *HLETF*. *SPT* gives priority to the task with the shortest

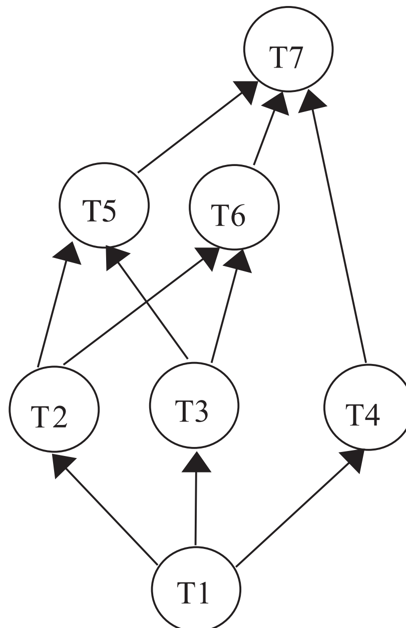

Fig. 1. An example task graph used to illustrate the AugNN approach (this is a 7-task, 2-machine problem).

remaining path. This rule did not provide good results. In the *LPT* rule, priority is given to the task with the longest processing time, regardless of the time of the remaining path. This rule worked better than *SPT*, although not as good as some of the other heuristics. Finally, the *RANDOM* dispatching rule, assigns priority at random. Surprisingly, this approach often gave better results than some of the other heuristics, in particular *SPT* and *LPT*. It is not the purpose of this research to compare these heuristics but to see if AugNN can improve upon each of these heuristics, by using adaptive learning.

4. The augmented neural network framework

The AugNN framework first translates the given scheduling problem to a NN architecture. We present here a very brief description of NNs so the reader is familiar with certain terms we use in discussing the AugNN framework. For a detailed discussion of NNs, see Rumelhart and McClelland (1989). A typical NN consists of a network of interconnected processing elements (PEs) or neurons, arranged in layers (input, hidden and output layers). PEs are connected from one layer to the next through links. These links are characterized by weights. With the help of input, activation (or transfer) and output functions on the PEs and weights on the links between PEs, a given input is transformed to an output. Better outputs are obtained in subsequent iterations by intelligently modifying the weights (using a learning rule) on the links.

We build a NN for the given task-scheduling problem, with weights on links between PEs and with input, activation and output functions on the PEs such that the output of the final PE gives the makespan of a feasible solution. We then use this adaptive machine-learning paradigm to find improved solutions in subsequent iterations by modifying the weights using an intelligent learning strategy. The input, activation and output functions can be designed to implement a given heuristic and to enforce the constraints of the problem. Various learning strategies can be devised. This approach thus offers the flexibility of using any established heuristic with any learning strategy.

The power of this approach comes from the combined use of heuristic and learning approach. The heuristic gives a good starting point. The approach therefore cannot do worse than the single-pass heuristic. The improvement in subsequent iterations comes from the learning involved. If a particular task when given priority over others, results in a better makespan, it is rewarded with higher priority weight and vice versa if it results in a worse makespan, it is penalized. Simulated Annealing and Tabu Search could also give good results, but by their nature, require excessive number of iterations.

We now describe how a given scheduling problem is translated into a NN architecture, using the example problem of Fig. 1. Next we will describe the general procedure, followed by the mathematical formulation.

4.1. AugNN applied to a special case

The proposed neural network framework is explained with the help of Fig. 2, using the example problem of Fig. 1. There are seven tasks and two machines in the problem. A network of task nodes (T) and machine nodes (M) is constructed as in Fig. 2. Node I acts as the initial node and F as the final node. Note that the task nodes in Fig. 2 follow the same precedence relationship as the tasks in Fig. 1. For each task node there are two machine nodes. In naming a machine node, the first subscript identifies the machine while the second identifies the task linked with it. Each task or machine node is viewed as a PE of a NN, each with its own input function, activation function and output function. The nodes are connected by links with weights associated with them. These weights help determine the assignment of tasks to machines. Within the same layer, tasks compete for assignment.

The input function, the activation function and the output functions are designed such that the network preserves the precedence order of the tasks. These functions also make sure that the same task is not assigned to more than one machine at the same time and that the same machine is not assigned to more than

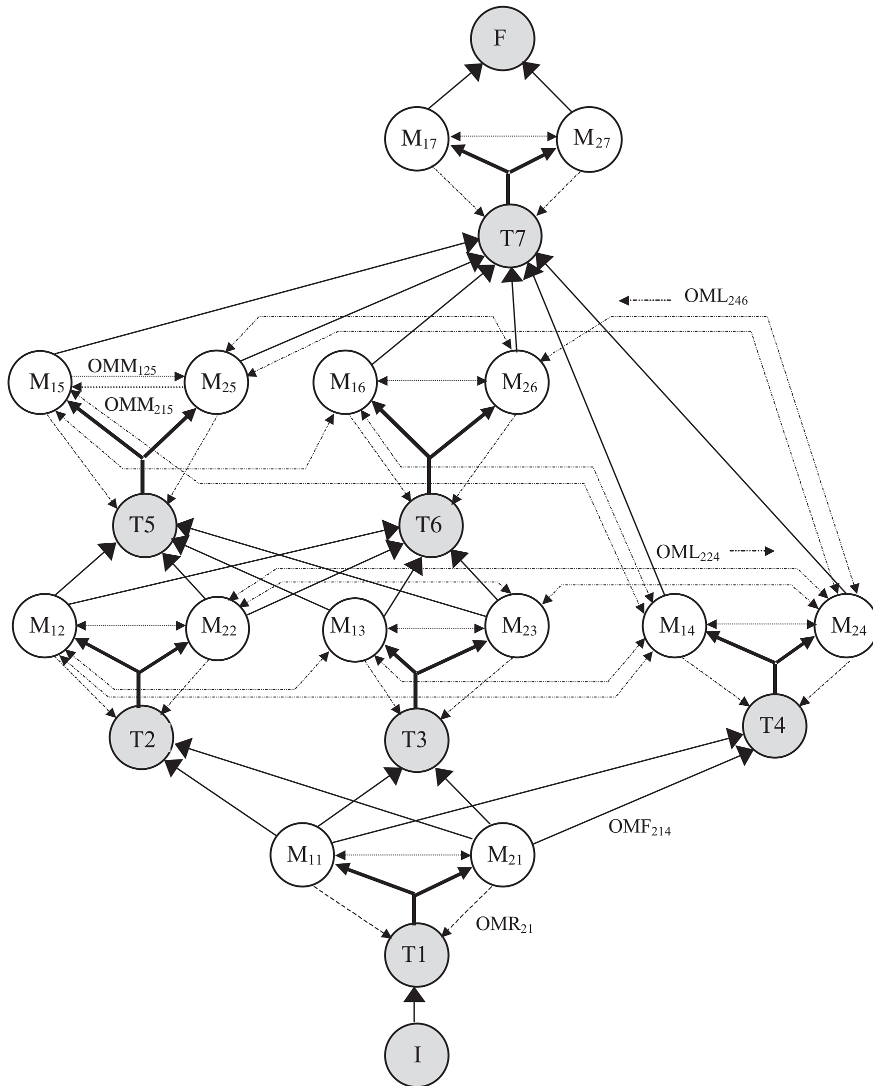

Fig. 2. Neural network architecture for the AugNN approach for the example problem of Fig. 1.

one task at the same time. These functions are described in detail in Appendix A. As can be seen, this network has all the elements of a typical NN, yet it is slightly different in that the network is not fully connected and there are many "hidden" layers. The connections are based on precedence of tasks and the number of hidden layers depends on the given problem. Also, PEs are linked to other PEs within the same layer, with those in the next layer, as well with those in the previous layer.

Note the presence of two sets of links (shown dotted in Fig. 2) between machine nodes. One set of dotted links is between two machines for the same task while the other is between the same machines on different tasks within the same layer. These links are used to send inhibitory signals to other nodes to enforce certain constraints. For example, the first set of links is used to ensure that the same task is not assigned to more than one machine while 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 and tasks 3 and 4 are not yet assigned, then node M_{12} will send an inhibitory signal to node M_{22} to ensure that task 2 does not also get assigned to machine 2 and also to nodes M_{13} and M_{14} to ensure that tasks 3 or 4 are not assigned to machine 1. Once task 2 is completed on machine 1, node M_{12} will send another signal withdrawing the inhibitory signal from M_{13} and M_{14}, thus allowing tasks 3 and 4 to be assigned to machine 1. 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. This link sends a signal to the task indicating that it is currently being processed.

One pass (or iteration) from node *I* through node *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 (gap between lower bound and the obtained solution), the learning rate and some parameter such as task processing time or *Level* of the task. 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 certain number of consecutive iterations.

4.2. The general AugNN architecture

The general procedure to construct the AugNN architecture from a given task-scheduling problem is now explained. The task nodes of the task graph become the task node of the AugNN structure. An initial node '*I*' and a final node '*F*' are added at the respective ends. *I* is the only PE in the input layer while *F* is the only PE in the output layer. The remaining tasks are represented with PEs in hidden layers. For each task node, as many machine nodes are added as there are machines in the problem. The machine nodes are then linked to all successor task nodes (according to the precedence constraints of the problem). These links help enforce the precedence constraint. Machine nodes for the same task are connected with a dotted link for inhibitory signal, to enforce the constraint that a task is not assigned to two machines. Same machine nodes on different tasks on the same layer are connected with dotted links for inhibitory signals to ensure that two tasks are not assigned to the same machine at the same time. Machine nodes are also linked to the task nodes in the reverse direction to indicate if a task is currently in process with a particular machine. Programmatically, these links are identified by matrices of connections between nodes. These matrices are discussed under Appendix A. The input is in the form of two matrices, an $n \times n$ precedence matrix and an $n \times m$ processing times matrix.

Flowcharts in Figs. 3 and 4 explain the general underlying AugNN solution procedure. As noted in Fig. 3, after reading the processing time and precedence matrix, we perform some preliminary steps such as finding the lower bound of the problem, initializing the weights and calculating the thresholds of tasks. A single iteration is then carried out using the input, activation and output functions of the task and machine nodes. See Fig. 4 for details. The mathematical treatment of the aforementioned functions is included in Appendix A.

At the end of an iteration, we decide whether to stop or continue with the next iteration. If the obtained solution is a lower bound solution then we know that the solution is optimal and we can stop. If not, then if we have performed a predetermined number of maximum iterations then we can stop. Or if the solution has not changed in the past ten iterations, we can stop. Otherwise we modify our weights using a learning strategy and start a fresh iteration.

5. Results and computational experience

The six heuristics discussed earlier and the AugNN approach were implemented in Visual Basic 6.0® environment, running on IBM compatible Pentium III 450 machine with 128 MB RAM.

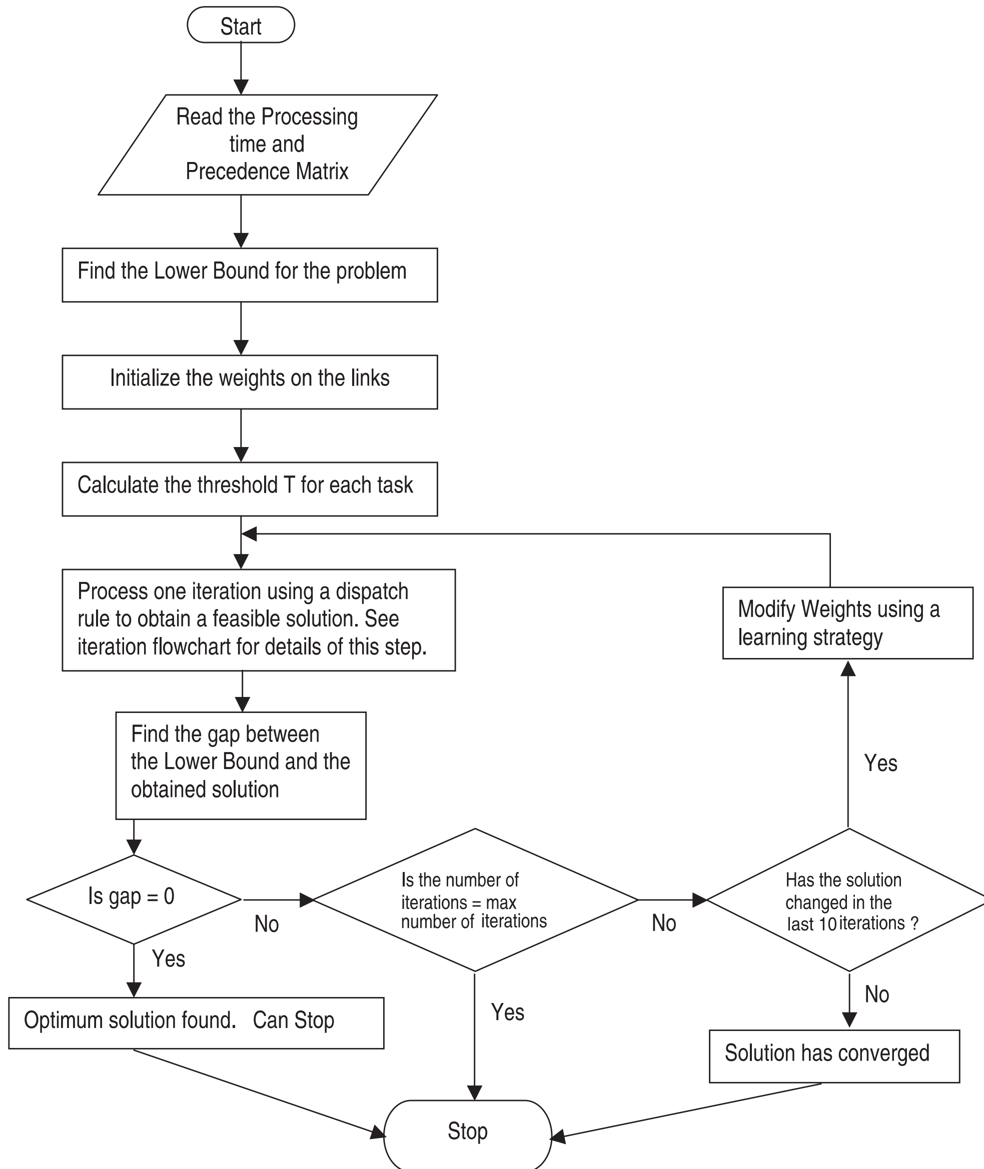

Fig. 3. Flowchart explaining the AugNN solution procedure.

5.1. Problems

Five hundred and seventy problems of various task/machine sizes were generated with random processing times (using uniform distribution) and random precedence matrices the same way such problems have been generated in previous studies such as Kasahara and Narita (1985). The number of tasks ranged from 10 to 100 and the number of machines from 2 to 5. For smaller number of tasks, fewer machines were used, and as the number of tasks increased, so did the number of machines. For each problem size, 30 instances of problems were generated. Table 1 summarizes the distribution of problems by size.

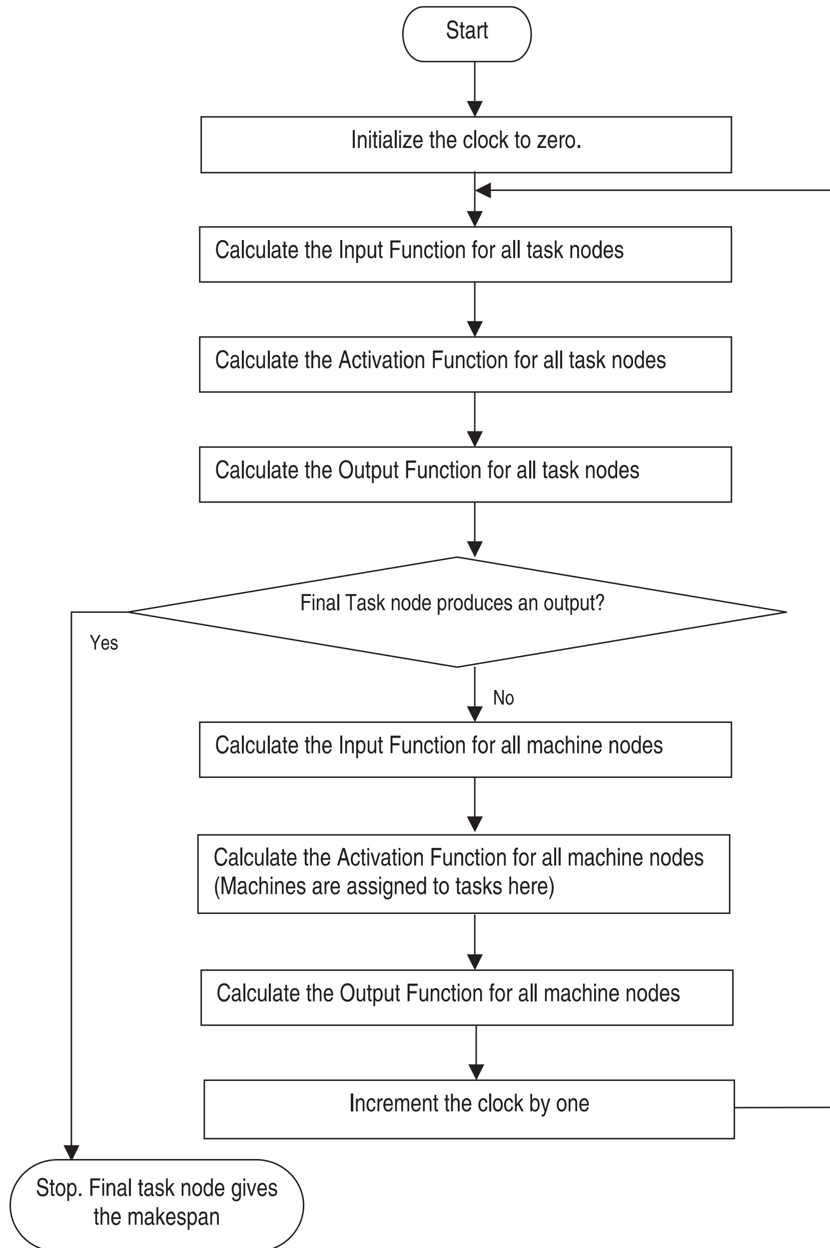

Fig. 4. Iteration flowchart.

5.2. Sensitivity analysis

The results obtained using AugNN depend upon various training parameters such as the learning rate (α), the learning rule, initial weight, and the number of maximum iterations used in the stopping rule. In order to ascertain the best values for these parameters we conducted a sensitivity analysis. We performed

Table 1
Number of problems for each size (# of tasks vs. # of machines) used for empirical work total of 570 problems are used

Number of tasks	Number of machines			
	2	3	4	5
10	30			
20	30	30		
30	30	30		
40		30	30	
50		30	30	
60			30	30
70			30	30
80			30	30
90			30	30
100			30	30
Total problems	90	120	210	150

the sensitivity analysis on the HLETF since this dispatch rule gave the best results. We first determined how many iterations were enough to find good solutions. So we ran all the 570 problems using various number-of-maximum-iterations in the stopping rule. We tried up to 200 iterations. Table 2(a) and Fig. 5 summarize the results.

We found that the most reduction in gap occurs in the first 10 iterations (28.3%). Thereafter, for each increment of 10 iterations, up to 40 or so iterations, the improvement is several percent points. Beyond 50 iterations, the rate of improvement slows down considerably. While the first 100 iterations give an improvement in gap of 42.5%, the next 100 iterations only improve the gap by about 1.4%. Since the bulk of the improvement occurred in the first 100 iterations, we ran our experiments with 100 iterations. However, if the problem requires a more accurate solution, the max number of iterations can be improved to 200 or higher.

We next performed the sensitivity analysis on the learning rate (or α). We tried α of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0 and 2.0. Table 2(b) and Fig. 6 show the results. We found that very small α (0.001) did not perform that well. Beyond an α of 0.01, the results were not very sensitive to changes in α. The best makespan was obtained for α of 0.1. So we used 0.1 as our α for the rest of the experiments.

Initial weights are another parameter, which may affect learning, although in theory it should not make much difference. To see what level of initial weight to use, we performed the sensitivity analysis on this parameter too. We tried initial weights of 1, 10, 50 and 100. The results are displayed in Table 2(c) and Fig. 7. There were not any significant differences. An initial weight of 10 performed the best, with no significant differences for other initial weights.

5.3. Results with different heuristics

We then tested how well each of the six heuristics and the AugNN using these heuristics performed on our problems. We used a learning rate of 0.1, initial weight of 10 and we ran 100 maximum iterations. Twelve combinations of six heuristics and two learning rules were run on 570 problems. Results are reported in Tables 3–7. Three criteria are used for the performance evaluation of AugNN. The first and the most important criterion is—Reduction (or improvement) in the makespan, which is equivalent to the reduction in gap between the lower bound (LB) solution and the heuristic solution. Hereafter, the gap between obtained solution and LB solution will simply be referred to as the 'gap'. Reduction in gap can be expressed in absolute terms as well as in percent terms (i.e. percent of gap obtained by the heuristic). Table 3 provides aggregate results for all 570 problems for this criterion, for all heuristics/learning rule combinations.

Table 2
Sensitivity analysis for the parameter

	Gap using AugNN with HLETF	Improvement over single pass (%)
(a) *Number of maximum iterations in the stopping rule*[a]		
5	4105	19.8
10	3672	28.3
20	3346	34.6
30	3181	37.9
40	3095	39.6
50	3044	40.5
60	3018	41.0
70	2996	41.5
80	2977	41.8
90	2962	42.2
100	2946	42.5
150	2908	43.2
200	2876	43.8
(b) *Learning rate*[b]		
0.001	3210	37.3
0.005	3016	41.1
0.01	2990	41.6
0.05	2991	41.6
0.1	2946	42.5
0.5	3007	41.3
1.0	2998	41.5
2.0	2997	41.5
(c) *Initial weight*		
1	3003	41.4
10	2946	42.5
50	2989	41.6
100	2977	41.9

[a] Other parameters used were learning rate: 0.1; initial weight: 10; min weight: 0.001.
[b] Other parameters used were max iterations: 100; initial weight: 10; min weight: 0.001.

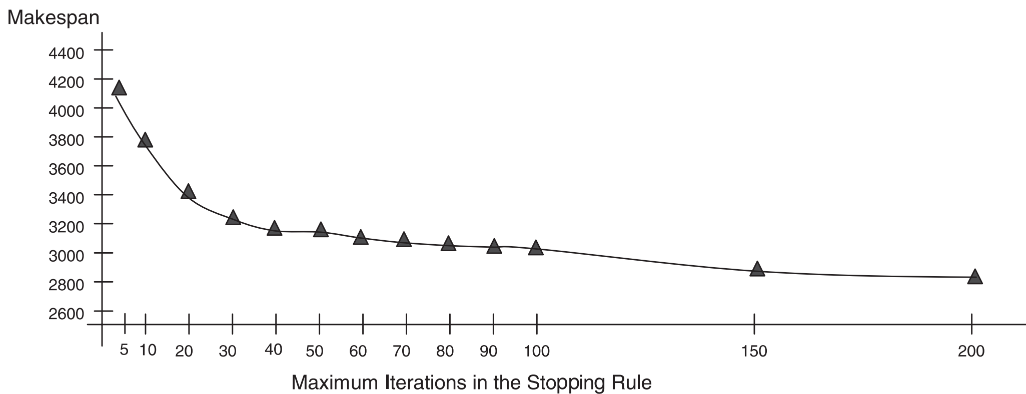

Fig. 5. Sensitivity analysis for maximum iterations in the stopping rule.

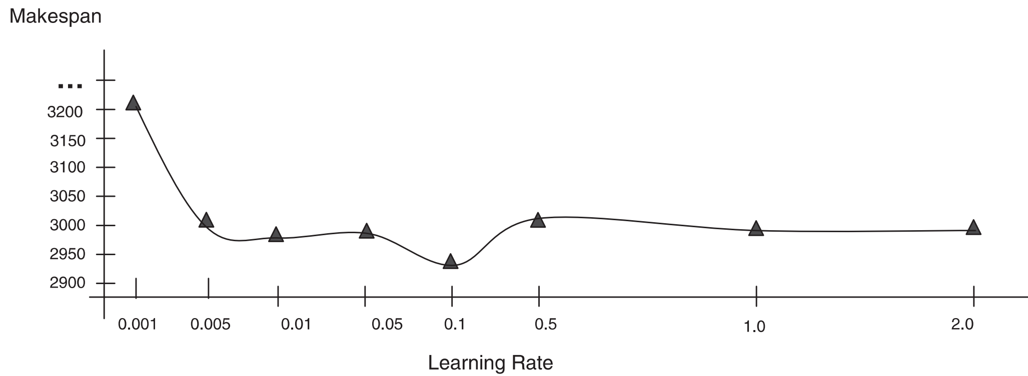

Fig. 6. Sensitivity analysis for the learning rate.

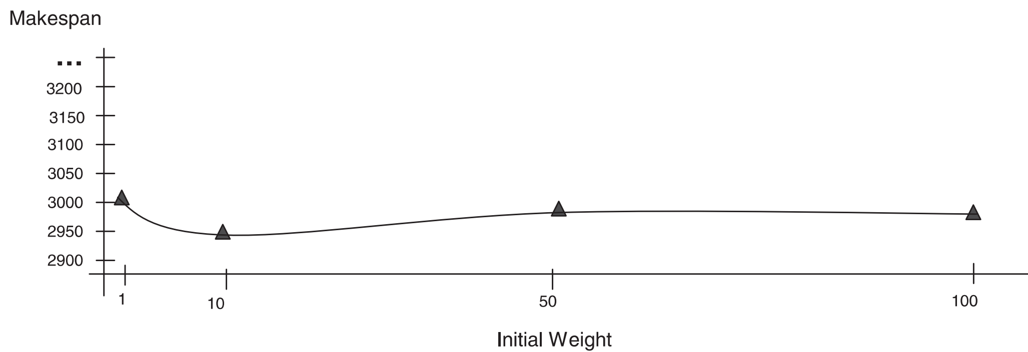

Fig. 7. Sensitivity analysis for the initial weight.

The second criteria is—Number of cases with known optimum solutions, i.e. cases with LB solutions. This can also be expressed in both absolute or percentage (of total problems) terms. Table 4 summarizes the results for this criterion. While this criterion takes into account known optimum solutions, it ignores cases where the AugNN approach improves upon the heuristic solution but does not necessarily reach a known optimum solution. So we have a third criterion—Cases where improvement in makespan occurs over heuristic. This can be expressed as a percentage of cases where improvement may (or may not) be possible. If the heuristic finds a LB solution, then it is also an optimum solution and no improvement is possible. If the heuristic finds a non-LB solution, it may or may not be an optimum solution. So an improvement may not be possible (if the solution is optimum), or possible (if it is not optimum). Table 5 summarizes the results for the third criterion.

For each of the 570 problems, AugNN solution is better than or equal to the heuristic solution. Consider the first criteria, i.e., Reduction in gap. The gaps obtained from the various single pass heuristics range from 5.8% of LB (for *HLETF*) to as much as 26.9% of LB (for *SPT*) (see Table 3). The gaps obtained from AugNN range from 3.3% (for *HLETF*) to 11.1% (for *SPT*). Ignoring the SPT heuristic, the worst gap given by AugNN is only 6.9%. The reduction in gap by AugNN over the gap obtained by the heuristic, ranges from 38.6% to as high as 58.7%. Considering that the heuristics considered in this paper are regarded as 'very good' in the scheduling literature, improvements of this order are quite impressive, especially because they are obtained in just a few iterations. Although it is not the purpose of this paper to compare heuristics with each other, it may be noted that the heuristic based on *HLETF* dispatching rule performs significantly

Table 3
Aggregate results for 570 problems for AugNN and six single pass heuristic algorithms, for different learning rules

Heuristic used	Task parameter used in learning rule	Makespan with single pass heuristic	Makespan with AugNN	Gap[a] from LB[b] with heuristic	Gap from LB with AugNN	Reduction in makespan (or gap) by AugNN[c]	Percent reduction in gap by AugNN over heuristic	Gap[a] using heuristic, expressed as percentage of LB	Gap[a] using AugNN, expressed as percentage of LB
HLF	L	94473	91629	7283	4439	2844	39.0	8.3	5.0
	LET	94473	91665	7283	4475	2808	38.6	8.3	
HLETF	LET	92311	90136	5121	2946	2175	42.5	5.8	3.3
CP/MISF	CP	96003	92283	8813	5093	3720	42.2	10.1	5.8
	LET	96003	92495	8813	5305	3508	39.8	10.1	
SPT	SPT	110649	96889	23459	9699	13760	58.7	26.9	11.1
	LET	110649	96943	23459	9753	13706	58.4	26.9	
LPT	LPT	101435	93290	14245	6100	8145	57.2	16.3	6.9
	LET	101435	93296	14245	6106	8139	57.1	16.3	
Random	Random	99816	92657	12626	5467	7159	56.7	14.5	6.2
	LET	100066	92540	12876	5350	7526	58.4	12.7	

[a] Gap between obtained solution and the lower bound.
[b] The LB for these 570 problems is 87,190.
[c] The reduction refers to reduction by AugNN with respect to single pass heuristic. Reduction in makespan is equivalent to reduction in the gap between obtained solution and LB.

Table 4
Cases with known optimum solution[a] for all 570 problems obtained using single pass heuristics and with AugNN

Heuristic used	Task parameter used in learning rule	With single pass heuristic	Expressed as percentage	With AugNN	Expressed as percentage
HLF	L	11	1.93[b]	66	11.58
	LET	11	1.93	58	10.17
HLETF	LET	18	3.16	91	15.96
CP/MISF	CP	13	2.28	63	11.05
	LET	13	2.28	60	10.52
SPT	SPT	1	0.17	21	3.68
	LET	1	0.17	23	4.03
LPT	LPT	10	1.75	50	8.77
	LET	10	1.75	48	8.42
Random	Random	7	1.22	73	12.88
	LET	4	1.22	75	13.16

[a] A solution is known to be optimum if the solution equals the lower bound.
[b] 11 as a percentage of 570 is 1.93.

Table 5
Cases where improvement over heuristic solution occurs using AugNN

Heuristic used	Task parameter used in learning rule	Cases where improvement may be possible[a]	Cases where improvement occurs	Percentage of cases improved to total possible
HLF	L	559[b]	527	94.2
	LET	559	525	93.9
HLETF	LET	552	518	93.8
CP/	CP	557	516	92.6
MISF	LET	557	521	93.5
SPT	SPT	569	561	98.6
	LET	569	562	98.7
LPT	LPT	560	538	96.1
	LET	560	531	94.8
Random	Random	563	543	96.4
	LET	563	549	97.5

[a] We say "may be possible" because even though the single pass heuristic does not give a lower bound solution, it could give an optimal solution, which cannot be improved upon.
[b] 559 is (570 minus 11), because 11 solutions were known to be optimal.

better than others. In fact, the range of results of the six heuristics is very high (gaps of 5121–23,459). AugNN on the other hand, provides results in a much narrower range (gaps of 2946–9753), hence AugNN may be considered quite robust.

The second criterion looks at the number of cases for which a known optimum is achieved. While the best heuristic gives a known optimum solution for 18 out of 570 problems (3.16%), AugNN gives a known optimum for as many as 91 problems (or 16%) (see Table 4). For each heuristic, the improvement in the number of known cases is significant. Results for third criterion are also very good. See Table 5 for a

Table 6
Number of iterations and CPU times

Heuristic used	Task parameter used in learning rule	Number of iterations using AugNN to find the best solution (average of all 570 problems)	Total iterations needed by AugNN (average of all 570 problems)	CPU time (in second) using single pass heuristic for all 570 problems	CPU time (in second) using AugNN for all 570 problems
HLF	L	35.7	90	32[a]	833
	LET	33.8	90		835
HLETF	LET	27.1	86	34	818
CP/MISF	CP	50.6	84	35	830
	LET	34.6	91		880
SPT	SPT	36.5	92	32	760
	LET	34.4	96		747
LPT	LPT	37.1	97	33	796
	LET	36.3	93		790
Random	Random	40.4	91	33	793
	LET	40.9	91		838

[a] For single pass heuristic, the CPU time is independent of the learning rule.

Table 7
Results by problem size for HLF heuristic (panel A) and HLETF heuristic (panel B)

No. of tasks	LB	Makespan with single pass heuristic	Makespan with AugNN (HLF)	Gap by single pass heuristic	Gap by AugNN	Improvement over single pass	Percent improvement	Avg CPU time per problem
Panel A								
10	1443	1571	1511	128	68	60	46.8	0.047
20	5032	5390	5133	358	101	257	71.8	0.309
30	7516	7939	7711	423	195	228	53.9	0.732
40	7442	8217	7945	775	503	272	35.1	1.328
50	9144	9916	9600	772	456	316	40.9	2.087
60	8523	9580	9259	1057	736	321	30.3	2.609
70	9954	10959	10629	1005	675	330	32.8	3.386
80	11469	12585	12228	1116	759	357	31.9	5.147
90	12670	13499	13180	829	510	319	38.5	6.242
100	13997	14817	14433	820	436	384	46.8	6.869
			Makespan with AugNN (HLETF)					
Panel B								
10	1443	1613	1503	170	60	110	64.7	0.046
20	5032	5339	5119	307	87	220	71.7	0.285
30	7516	7908	7665	392	149	243	61.9	0.596
40	7442	8047	7807	605	365	240	39.7	1.261
50	9144	9689	9457	545	313	232	42.6	2.126
60	8523	9181	8965	658	442	216	32.8	2.603
70	9954	10628	10380	674	426	248	36.8	3.361
80	11469	12244	12008	775	539	236	30.5	5.144
90	12670	13194	12985	524	315	209	39.9	6.265
100	13997	14468	14247	471	250	221	46.9	6.546

summary of number of cases where a positive improvement occurs. AugNN provides an improvement over the heuristic solution on a very high percentage of problems (92.6–98.7%).

We also show the results by problem size, for the various heuristics. These results are summarized in Table 7a and b for HLF and HLETF dispatching rules. We find that problems with 20 tasks showed the maximum improvement (71.8% and 71.7% for HLF and HLETF respectively). For all other size tasks we did not observe any significant variance and the solution procedure seems to work equally well for all size problems. Computational complexity and computational times are discussed in the following subsection.

5.4. Computational complexity

The computational complexity for our algorithm is $O(n^2 + mn)$ per iteration. This is also the computational complexity of each of the all heuristics considered in the study. The only difference is that AugNN requires a few iterations whereas all the heuristics are single pass heuristics. On an average, AugNN gives the best solution within 40 iterations for the random heuristic and less than 37 for most of other heuristics. The total number of iterations tried on average ranged from 84 to 97. Table 6 summarizes the results of the number of iterations taken. The computation time of course increased as the problem size increased. A 50-task problem was solved in an average of 2 seconds, a 100 task problem on average took about 6.5 seconds. The average CPU time taken per problem is in the range of 1.3–1.5 seconds. In comparison to these results, the computational complexity of NN based on Hopfield and Tank paradigm is an order of magnitude higher (worse) than our computational complexity. Lo and Bavarian (1993) show a complexity of $O(nmk)^2$, where k is the makespan. Also, the number of iterations required in the Hopfield and Tank paradigm approach can range upto several thousands (upto three thousand in case of Lo and Bavarian) compared to ours which averaged below 37 for most heuristics.

6. Summary and conclusions

In this paper we propose a neural network framework for solving the task-scheduling problem. The problem considered is that of minimizing the makespan for scheduling n tasks on m identical machines, where the tasks follow a predetermined precedence order and pre-emption of tasks is not permitted. This problem belongs to the class of NP-Hard problems. For such problems, many heuristic based algorithms exist. The proposed neural network approach, called Augmented Neural Network (AugNN), is significantly different from the hitherto known neural network approaches, which are based on Hopfield and Tank paradigm. The proposed approach is a hybrid of the traditional heuristic approach and the adaptive machine learning approach. AugNN uses a network architecture that resembles the task graph of the scheduling problem. Through weights assigned to the links between tasks and machines, and by adjusting the weights using an appropriate learning strategy, a significantly improved schedule is found in just a few iterations. Any known heuristic can be used with the AugNN approach, in conjunction with a suitable learning strategy. We use six different heuristics and two different learning strategies to show the effectiveness of the AugNN approach.

We use three criteria to compare our results with the six single pass heuristics—(i) reduction in gap between LB solution and heuristic solution, (ii) number of cases with known optimum solutions, and (iii) number of cases where an improvement in makespan occurs over the heuristic. Empirical tests performed over 570 problems, with size ranging from 10 to 100 tasks and 2 to 5 machines, show that the proposed AugNN approach far outperforms the single pass heuristics in all the three criteria. These improvements occur at the expense of some extra CPU time because of the iterations involved in the AugNN approach. Nevertheless, the number of iterations needed to find the solution was reasonably low, at an average of 37 for most heuristics.

This study makes a contribution in the ongoing work on the application of neural networks for optimization problems. The proposed AugNN algorithm's scope is not limited to task-scheduling problems. With a slight modification, the approach can be applied to the general job-shop scheduling problem. The different variations of the scheduling problem, i.e., those with different objective functions (such as minimizing tardiness, meeting deadlines) and those with different constraints (such as set up time, pre-emption allowed) can also be solved by this approach. Although at this time the augmented networks seem to be specifically suited to precedence constrained problems, their generality is yet to be explored. Future research can explore the possibility of applying the augmented NN approach for other types of optimization problems.

Appendix A. Mathematical formulation and algorithm details

Notation used

n	number of tasks
m	number of machines
k	current iteration
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, $i \in M$, $j \in T$
L_j	level of T_j, $j \in T$ (number of tasks in the remaining path till the final node)
LET_j	level with estimated time of T_j, $j \in T$ (processing rime of the remaining path)
ω_j	weight on the link from T_j to machine nodes
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 T_j, $j \in T \cup F$
ST_j	start time of T_j, $j \in T$
PT_j	processing time of T_j, $j \in T$
LST_j	latest start time of T_j, $j \in T$
CP_j	whether task j is on the critical path, $j \in T$
NIS_j	number of immediate successor tasks of 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 $\{j \mid \mathrm{PR}_j$ 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 node j, $j \in I \cup T \cup F$

$\mathrm{IM}_{ij}(t)$ input function value of machine node i from task node j, $i \in M$, $j \in T$

$\mathrm{OT}_j(t)$ output function value of task node j, $j \in I \cup T \cup F$

$\mathrm{OMF}_{ijp}(t)$ 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 node j, $j \in T$
$\theta M_{ij}(t)$ activation function of machine node M_{ij}, $i \in M$, $j \in T$
$assign_{ij}(t)$ machine i 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

Preliminary steps:

1. Calculate the lower bound. This requires that we first calculate Levels with estimated time, or (LET_j). LET_j is the length (measured in time units) of the longest path from task j up to the final node, including the processing time of task j. Lower bound for the makespan is calculated as follows:

$$\text{Lower Bound} = \max_j \left(\mathrm{LET}_1, \quad \left\lceil \sum (\mathrm{PT}_j | m) \right\rceil \right)$$

   The first quantity is the LET of task 1. Since there cannot be a feasible solution with a makespan less than LET_1, LET_1 represents a lower bound. The second quantity above indicates that no machine is ever idle. The higher of these two quantities makes for an excellent lower bound.
2. Weights (ω_j) are initialized at 10.00. The value 10 was arrived at after some computational experience. The value of the initial weights should be such that after subsequent modification to weights, the value should remain positive. The choice of the value of initial weight therefore also depends on the value of the learning coefficient used.
3. Calculate the threshold of each task τ_j. The threshold of task j is defined as the number of tasks immediately preceding task j. The threshold value is used to determine when a task is ready to start.

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

A.1. AugNN functions

A.1.1. Task nodes

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

Input function

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

For nodes with no preceding tasks

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

All tasks with no preceding constraints get a starting signal.
Threshold of all tasks in NPR is 1 due to the I node.
For all other tasks, i.e., $\forall j \notin \mathrm{NPR} \wedge t > 1$

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

IT_j helps to enforce precedence constraint. When IT_j becomes equal to τ_j, the task can be assigned to a free machine.

Activation function

Task nodes' initial activation state (i.e. at $t = 0$) is 1. For all $i \in M, j \in T$,

$$\theta T_j(t) = \begin{cases} 1 & \text{if } \mathrm{IT}_j(t) < \tau_j \\ 2 & \text{if } (\theta T_j(t-1) = 1 \vee 2) \wedge \mathrm{IT}_j(t) = \tau_j \\ 3 & \text{if } (\theta T_j(t-1) = 2 \vee 3) \wedge \sum_i \mathrm{OMR}_{ij}(t) < 0 \\ 4 & \text{if } \theta T_j(t-1) = 4 \vee (\theta T_j(t-1) = 3 \wedge \sum_i \mathrm{OMR}_{ij}(t) = 0) \end{cases}$$

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

State 1 above implies that task j is not ready to be assigned because input to task j is less than its threshold τ. State 2 implies that task j is ready to be assigned because its input equals its threshold. State 3 implies that the task is in process because it is receiving a negative signal from a machine i that it is currently being processed. State 4 implies that the task is complete and the negative signal from machine i is no longer there.

Output function

$$\mathrm{OT}_j(t) = \begin{cases} 1 & \text{if } \theta T_j(t) = 4 \\ 0 & \text{otherwise} \end{cases}$$

If a task is ready to start but not assigned yet, it sends a unit signal to each machine node.

F-node

$$\mathrm{OT}_F(t) = \begin{cases} t-1 & \text{if } \mathrm{IT}_F(t) = \tau_F \\ 0 & \text{otherwise} \end{cases}$$

The final node outputs the makespan ($t - 1$), the moment its threshold point is reached.

Note: $t - 1$ is the makespan because the first assignment can occur at $t = 0$ but we are making the first assignment at $t = 1$ for ease of notation.

A.1.2. Machine nodes

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

Input

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

There are three components of $\mathrm{IM}(t)$. The first component is the weighted output from Task node j. Whenever it is positive, it means that machine i is being requested by task j for assignment. The second and third components are either zero or large negative. The second component becomes large negative whenever machine i is already busy with another task. This is the inhibitory signal discussed earlier. The third component becomes large negative whenever task j is assigned to another machine. ω_m is a fixed weight link between machines and is large to suppress the output of a task if the machine is busy or assigned or the task is assigned to another machine.

Activation function

Let $\chi_{ij}(t) = \mathrm{IM}_{ij}(t) * \mathit{TaskHeuristicParameter}_j$

$$\mathit{assign}_{ij}(t) = \begin{cases} 1 & \text{if } \chi_{ij}(t) = \mathrm{Max}[\chi_{ij}(t) \mid \forall T_j \in S(t)] \wedge \chi_{ij}(t) > 0 \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in M,\ j \in T$$

The assignment takes place if the product of Input of the machine node and the Heuristic dependent task parameter 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. *TaskHeuristicParameter* is a task parameter dependent on the chosen heuristic.

$$TaskHeuristicParameter = \begin{cases} L_j & \text{for } HLF \text{ heuristic} \\ \text{LET}_j & \text{for } HLETF \text{ heuristic} \\ \text{CP}_j * \text{NIS}_j & \text{for } CP/MISF \text{ heuristic} \\ \text{LET}_1 - \text{LET}_j & \text{for } SPT \text{ heuristic} \\ \text{PT}_j & \text{for } LPT \text{ heuristic} \\ \text{RND} & \text{for Random heuristic} \end{cases}$$

If $assign_{ij}(t) = 1$, then $\text{ST}_j = t$. If $|S(t)| > |\text{MA}(t)|$ then if $assign_{ij}(t) = 1$ then $\text{Win}_j = 1$. Machine nodes' initial activation state (i.e. at $t = 0$) is 1. For all $i \in M$, $j \in T$,

$$\theta M_{ij}(t) = \begin{cases} 1 & \text{machine available} \\ 2 \quad \text{if } (\theta M_{ij}(t-1) = 1 \vee \theta M_{ij}(t) = 1) \wedge assign_{ij}(t) = 1 & \text{machine busy (just assigned)} \\ 3 \quad \text{if } (\theta M_{ij}(t-1) = 2 \vee 3) \wedge t < \text{ST}_j + \text{PT}_j & \text{machine busy (processing)} \\ 4 \quad \text{if } \theta M_{ij}(t-1) = 3 \wedge t = \text{ST}_j + \text{PT}_j & \text{machine just released} \\ 5 \quad \text{if } \theta M_{ij}(t-1) = 1 \wedge \sum_{q \in S(t)} \text{OML}_{iqj}(t) * \omega_m < 0 & \text{assigned to another task} \\ 6 \quad \text{if } \theta M_{ij}(t-1) = 4 & \text{machine } i \text{ is finished processing task } j \\ 1 \quad \text{if } (\theta M_{ij}(t-1) = 1 \vee 5) \wedge \sum_{q \in S(t)} \text{OML}_{iqj}(t) * \omega_m = 0 & \text{released by other task or not assigned to any other task} \\ 1 \quad \text{if } \theta M_{ij}(t-1) = 1 \wedge \sum_{i^*} \text{OMM}_{i^*ij}(t) * \omega_m < 0 & \text{task assigned to another machine} \end{cases}$$

Output function

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

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

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

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

The output of F represents the makespan and the $assign_{ij}(t)$ 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.

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 some task parameter. This parameter could be the same parameter used in the heuristic or it could be a different parameter. Keeping these points in mind, the following learning strategy for the links.

Winning tasks	If $\text{Win}_j = 1$ then $(\omega_j)_{k+1} = (\omega_j)_k - \alpha * \mathit{TaskParameter}_j * \varepsilon_k \; \forall j \in T$
Non-winning tasks	If $\text{Win}_j = 0$ then $(\omega_j)_{k+1} = (\omega_j)_k + \alpha * \mathit{TaskParameter}_j * \varepsilon_k \; \forall j \in T$

We test each heuristic with two learning strategies. In the first learning strategy, we use the *TaskParameter* the same as the *TaskHeuristicParameter* used for the heuristic. For example if we use L_j as the *TaskHeuristicParameter* in the Input Function of Machine Nodes, then L_j is also used as the *TaskParameter* in the learning strategy. The second learning strategy we use is called breadth-first strategy. In this strategy, the *TaskParameter* is LET_j. Using LET_j we modify the weights of links near the initial node more than those near the final node, thus simulating a breadth-first strategy. We tried a third strategy that simulated depth-first search but the results were not very good and are not reported.

End of iteration routines:

1. Calculate the gap, i.e., the difference between obtained makespan and the lower bound.
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 100, stop the program. Most solutions were obtained in less than 50 iterations. So, there is enough margin of safety in running the program for upto 100 iterations. Of course, for some problems, a better solution was obtained at closer to 200 iterations. But such problems were few and far in between.
5. If continuing with the next iteration, modify weights using the learning strategy.

References

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedures for job shop scheduling. Management Science 34 (3), 391–401.

Bishop, C.M., 1995. Neural Networks for Pattern Recognition. Oxford University Press, New York.

Biskup, D., 1999. Single-machine scheduling with learning considerations. European Journal of Operations Research 115, 173–178.

Candido, M.A.B., Khator, S.K., Barchia, R.M., 1998. A genetic algorithm based procedure for more realistic job shop scheduling problems. International Journal of Production Research 36 (12), 3437–3457.

Chen, C.C., Yih, Y., Wu, Y.C., 1999. Auto-bias selection for developing learning-based scheduling systems. International Journal of Production Research 37 (9), 1987–2002.

Coffman, E.G., 1976. Computer and Job Shop Scheduling Theory. Wiley, New York.

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

Foo, Y.P.S., Takefuji, Y., 1988b. Stochastic neural networks for solving job-shop scheduling: Part 2, architecture and simulations. In: Proceedings of Joint International Conference on Neural Networks, vol. 2, pp. 283–290.

Foo, Y.P.S., Takefuji, Y., 1988c. Integer linear programming neural networks for job-shop scheduling. In: Proceedings of Joint International Conference on Neural Networks, vol. 2, pp. 341–348.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326.

Haldun, A., Bhattacharya, S., Koehler, G.L., 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.
Hu, T.C., 1961. Parallel sequencing and assembly line problem. Operations Research 9, 841–848.
Kasahara, H., Narita, S., 1985. Parallel processing of robot-arm control computation on a multi-microprocessor system. IEEE Journal of Robotics and Automation RA-1 (2), 104–113.
Lo, C.C., Hsu, C.C., 1993. A parallel distributed processing technique for job-shop scheduling problems. In: Proceedings of International Joint Conference on Neural Networks, pp. 1602–1605.
Lo, Z.P., Bavarian, B., 1993. Multiple job scheduling with artificial neural networks. Computers and Electrical Engineering 19 (2), 87–101.
Miller, D.M., Chen, H.C., Matson, J., Liu, Q., 1999. A hybrid genetic algorithm for the single machine scheduling problem. Journal of Heuristics 5, 437–454.
Panwalker, S.S., Iskander, W., 1977. A survey of scheduling rules. Operations Research 25, 45–61.
Pezzella, F., Merelli, E., 2000. A tabu search method guided by shifting bottleneck for the job-shop scheduling problem. European Journal of Operational Research 120, 297–310.
Rajendran, C., Holthaus, O., 1999. A comparative study of dispatching rules in dynamic flowshops and job shops. European Journal of Operational Research 116, 156–170.
Rumelhart, D., McClelland, J.L., 1989. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, MA.
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.
Sakawa, M., Kubota, R., 2000. Fuzzy programming for multi-objective job shop scheduling with fuzzy processing time and fuzzy due date through genetic algorithms. European Journal of Operational Research 120, 393–407.
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.
Steinhofel, K., Albrecht, A., Wong, C.K., 1999. Two simulated annealing-based heuristics for the job shop scheduling problem. European Journal of Operational Research 118, 524–548.
Thomas, P.R., Salhi, S., 1998. A tabu search approach for the resource constrained project scheduling problem. Journal of Heuristics 4, 123–139.
Uzsoy, R., Wang, C.S., 2000. Performance of decomposition algorithms for job shop scheduling problems with bottleneck machines. International Journal of Production Research 38, 1271–1286.
Zhang, H.C., Huang, S.H., 1995. Application of neural networks in manufacturing: A state of the art survey. International Journal of Production Research 33 (3), 705–728.