INFORMS Journal on Computing
Vol. 18, No. 1, Winter 2006, pp. 119–128
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1801 | 0119

DOI 10.1287/ijoc.1040.0108
© 2006 INFORMS

An Improved Augmented Neural-Network Approach for Scheduling Problems

Anurag Agarwal
Department of Decision and Information Sciences, Warrington College of Business Administration,
University of Florida, Gainesville, Florida 32611-7169, USA, aagarwal@ufl.edu

Varghese S. Jacob, Hasan Pirkul
School of Management, University of Texas at Dallas, Richardson, Texas 75083-0688, USA
{vjacob@utdallas.edu, hpirkul@utdallas.edu}

For the task-scheduling problem, we propose an augmented neural-network approach, which allows the integration of greedy as well as nongreedy heuristics (AugNN-GNG), to give improved solutions in a small number of iterations. The problem we address is that of minimizing the makespan of n tasks on m identical machines (or processors), where tasks are nonpreemptive and follow a precedence order. The proposed approach exploits the observation that a nongreedy search heuristic often finds better solutions than do their greedy counterparts. We hypothesize that combinations of nongreedy and greedy heuristics when integrated with an augmented neural-network approach can lead to better solutions than can either one alone. We show the formulation of such integration and provide empirical results on over a thousand problems. This approach is found to be very robust in that the results were not very sensitive to the type of greedy heuristic chosen. The new approach is able to find solutions, on average, within 1.8% to 2.8% of the lower bound compared to 2.0% to 8.3% for the greedy-only AugNN approach. This improvement is obtained without any increase in computational complexity. In fact the number of iterations used to find the solution decreased.

Key words: production scheduling; heuristics; neural networks
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received June 2002; revised April 2003; accepted September 2003.

1. Introduction and Motivation

The resource-constrained task-scheduling problem is at the heart of many scheduling problems. The problem involves finding the minimum makespan schedule for n tasks on m identical machines (or processors, or resources), where tasks have precedence constraints and cannot be preempted. Variations of this type of problem are routinely found in production and computing environments. For example, the open-shop, the flow-shop, and the job-shop scheduling problems are extensions of the basic resource-constrained task-scheduling problem. This type of problem also appears in a distributed computing environment. The resource-constrained project-management problem is also a variation of the task-scheduling problem. For a review of applications of task-scheduling problems, see Rinnooy Kan (1976) and Coffman (1976). Any improvements in the solution procedure of the task-scheduling problem will likely lead to improvements in a variety of other scheduling problems. We apply a hybrid approach of heuristics and neural networks to find improved solutions for this resource-constrained task-scheduling problem. The focus is on integrating greedy as well as nongreedy heuristics with neural networks to obtain improved solutions.

The scheduling literature is replete with a variety of solution procedures for a variety of scheduling problems. A host of single-pass heuristic approaches, as well as iterative approaches have been proposed. See Sabuncuoglu (1998) for a review of the use of neural networks in scheduling. More recently, Agarwal et al. (2003) proposed a framework called *augmented neural-network* (AugNN), which allows the integration of various greedy heuristics with neural networks. This framework exploits the advantages of both the heuristics and the neural-network approaches. The heuristic approach finds a good single-iteration solution quickly, while the neural-network approach allows iterative adaptive learning. The AugNN approach is shown to be effective even for large problems (100 tasks) and has the computational complexity of $O(n^2 + nm)$, which is the same as that of most greedy heuristics for this problem. Significant improvements over single-pass heuristics were reported in just 50 or so iterations. Most iterative approaches such as genetic algorithms, simulated annealing, traditional neural networks, tabu search etc., require thousands of iterations and therefore are effective only for small problems.

This study was motivated by the observation that for many problems, a nongreedy heuristic may sometimes find a better solution than a greedy heuristic.

The problem is how to integrate the two approaches, i.e., greedy and nongreedy, to obtain the best possible solution. Because iterative approaches generally provide better solutions, as long as the number of iterations is reasonable (less than 100), the extra computing time may be worth the improvement. Because AugNN gives good results in just a few iterations, we focus on integrating nongreedy heuristics, along with greedy heuristics to neural networks using the AugNN framework.

Using the proposed algorithms, we were able to obtain solutions, on average, within 1.8% to 2.8% of the lower bound for various heuristics, compared to 2.0% to 8.3% obtained from greedy-only AugNN for the same set of problems. While the greedy-only approach was sensitive to the type of greedy heuristic used, the new approach works quite well with any heuristic. Hereafter, we will call this approach AugNN-GNG (for AugNN with greedy and nongreedy heuristics combined).

This study contributes to the scheduling literature. First, it proposes near-optimal robust algorithms for an important class of scheduling problems. Second, it develops strategies for combining nongreedy and greedy heuristics and empirically demonstrates the effectiveness thereof. Third it extends the AugNN literature for scheduling by proposing (i) extensions to the architecture of neural networks, (ii) new learning rules, and (iii) new iteration schemes.

In Section 2, we describe the task-scheduling problem and analyze it. The scheduling problem is reduced to two decision problems—prioritization and wait. We discuss local and look-ahead strategies to tackle the two decision problems and set the stage for comparing greedy and nongreedy heuristics. A specific instance of the problem is used for illustration. In Section 3, we provide a literature survey for scheduling problems. Section 4 briefly describes the AugNN approach, enough to understand our extensions. The details of the AugNN approach are in the online supplement to this paper on the journal's website. We then describe how the AugNN approach is modified or extended in this paper. Computational complexity is also discussed. In Section 5, empirical results are reported and discussed. Section 6 discusses a few issues related to neural network training and extensions of this approach to other scheduling problems. Finally, Section 7 provides a summary, conclusions, and suggestions for further research.

2. The Problem and Its Analysis

We first briefly explain greedy versus nongreedy search heuristics. At every step, a greedy search heuristic selects the best choice available at that step without regard to future consequences (Gass and Harris 2001). Nongreedy heuristics entertain the idea that what is best at each step may not be the best for the entire problem. For example, a typical greedy heuristic for a traveling salesman problem is to visit the nearest neighboring unvisited city with no regard to the overall map of cities. While this strategy may generate an optimal subtour in the beginning, it may result in long paths towards the end of the tour. A nongreedy heuristic would not necessarily choose to visit the nearest city, realizing that other options might actually result in a better overall solution. Greedy algorithms are easier to implement, as they simply make a series of local decisions. Nongreedy heuristics need more problem-specific knowledge to make choices at each step.

The problem we address is scheduling n tasks on m identical machines (or processors). Tasks are not independent because they follow a precedence order. Further, tasks are nonpreemptive, which implies that once assigned, they must go through to completion. The objective is to minimize the makespan (or the completion time). An instance of such a problem is described in Figure 1. There are nine tasks and two machines. The task graph shows the precedence relationships. Processing times are shown inside the task nodes.

We now analyze the task-scheduling problem and reduce it to two decision problems. At any point, a task can be in one of four states: (1) "Not Ready" for assignment because of precedence constraints, (2) "Ready" for assignment, (3) "Assigned and In-process" and (4) "Completed." If a task is in one of states 1, 3 or 4, there is no assignment decision to

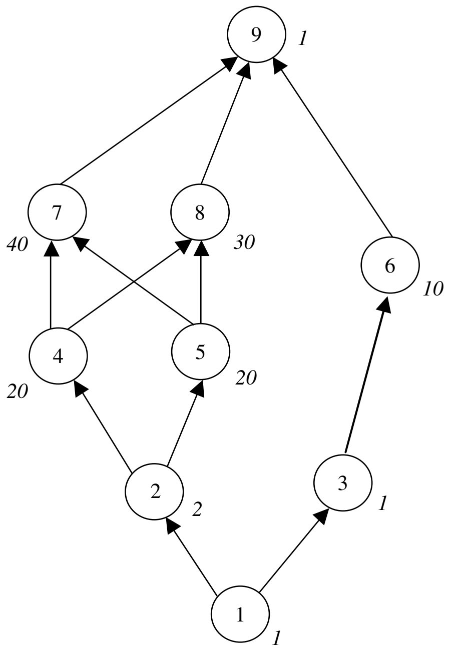

Figure 1 An Example Task Graph of a Task-Scheduling Problem

be made because either the decision has already been made or cannot be made yet. State 2 requires some decision making.

Two scenarios are possible related to state 2. In the first scenario, at any given time, there are fewer machines available than the number of "Ready" tasks (i.e., tasks in state 2). If the number of machines is zero, there is no decision to be made. If the number of machines is positive but less than the number of tasks ready, then clearly we need a priority rule, criterion, or heuristic, to assign tasks to a limited number of machines. This is the prioritization decision problem. Typically a greedy heuristic such as a *highest-level-first* or *longest-processing-time-first* heuristic is used to settle the priority.

In the second scenario, at any given time, there are at least as many machines available as the number of ready tasks. In other words, tasks do not have to compete for limited resources. Greedy heuristics would not treat this scenario as one requiring a decision, and would assign all ready tasks to available machines. However, a nongreedy approach would consider the following possibility. Even though certain tasks are in state 1 and therefore not competing for machines at a given time, had they been competing they might have been assigned a higher priority than some of the tasks currently in state 2. Should such a possibility exist, it would be prudent to treat a state 2 task of low priority as if it were still in state 1, and letting another state 1 task become a state 2 task and get assigned first. In other words, a task may sacrifice its "Ready" state and remain idle in spite of available resources for the sake of other more critical tasks, in the hope of improving the makespan. So we have the wait-decision problem, i.e., whether to force a state 2 task to wait or not.

This nongreedy approach of considering whether to force a task to wait would also help in the first scenario in which the number of machines available is less than the number of tasks ready to start. In this situation, a task may be asked to "wait" to allow for more critical tasks to become ready to start. We will apply the nongreedy heuristic to all situations, i.e., whether we have more machines, the same number, or fewer machines than the number of tasks ready to start. Once tasks are allowed to be assigned, if the number of machines is less than the number of tasks allowed to be assigned, we face the prioritization-decision problem. Intuitively, a task with a high slack would be a strong candidate for a "wait" decision especially if tasks with zero slack are about to become "Ready" soon.

By addressing the "wait" decision problem, which amounts to a nongreedy heuristic, for many problems we got results better than those from greedy heuristics. We hypothesize that an algorithm that addresses both the prioritization and the wait decision problems would provide better solutions than one that only addresses one. For the prioritization decision, we propose any one of the many greedy heuristics in the literature. For the wait-decision problem, we propose a specific nongreedy heuristic. We then propose two alternative strategies to combine the two approaches using the AugNN framework.

The task graph of Figure 1 can also be used to show how a nongreedy algorithm gives a better solution than a greedy one. This example is due to Ramamoorthy et al. (1972). The numbers inside the circles indicate task numbers while those outside indicate processing times in some unit of time (*ut*). Assuming infinite resources, paths 1-2-4-7-9 and 1-2-5-7-9 are critical. The minimum possible makespan is 64 *ut*s. Path 1-3-6-9 is obviously noncritical with a slack of 51 *ut*s. When task 3 is completed, task 6 is ready to start (state 2), but only one *ut* later, tasks 4 and 5, both of which are on critical paths and currently in state 1, will become ready to start. Hence, it is easy to see that it is advantageous to let task 6 wait and not tie up one of the two machines. Had there been more than two machines, however, assignment of task 6 without wait would not have made any difference. Gantt charts in Figure 2 illustrate the two schedules, with and without task 6 waiting. Schedule (b), in which task 6 waited, resulted in a better makespan than schedule (a), in which it did not wait. This example clearly illustrates the advantage of letting a noncritical task in state 2 wait.

3. Literature Review

Scheduling problems have been studied for a long time. For a classification and a review of methods for solving them see Graham et al. (1979) or Rinnooy Kan (1976). Cheng and Sin (1990) also provide an excellent summary of the results of algorithms for the various scheduling problems. Casavant and Kuhl (1988) provide a taxonomy of scheduling problems. Potts and Kovalyov (2000) provide a review of scheduling with batching.

Hu (1961) pioneered the idea of heuristic approaches for solving scheduling problems for the identical-machine case with a task precedence relationship. He developed "level-scheduling" or "list-scheduling," a critical-path approach for greedy heuristics. Panwalker and Iskander (1977) provide a survey of dispatching rules. Adam et al. (1974) also provide comparisons of various list schedules. Examples of dispatching rules include *SPT* (shortest path time), *LPT* (longest processing time), *HLF* (highest level first), *HLETF* (highest level with estimated time first), *LWKR* (least work remaining), *MWKR* (most work remaining), *FCFS* (first come first served), *MOR* (most operations remaining), RANDOM etc. Kasahara and Narita (1984, 1985) propose a variation of list scheduling called CP/MISF (critical path with most immediate successors first) and show that

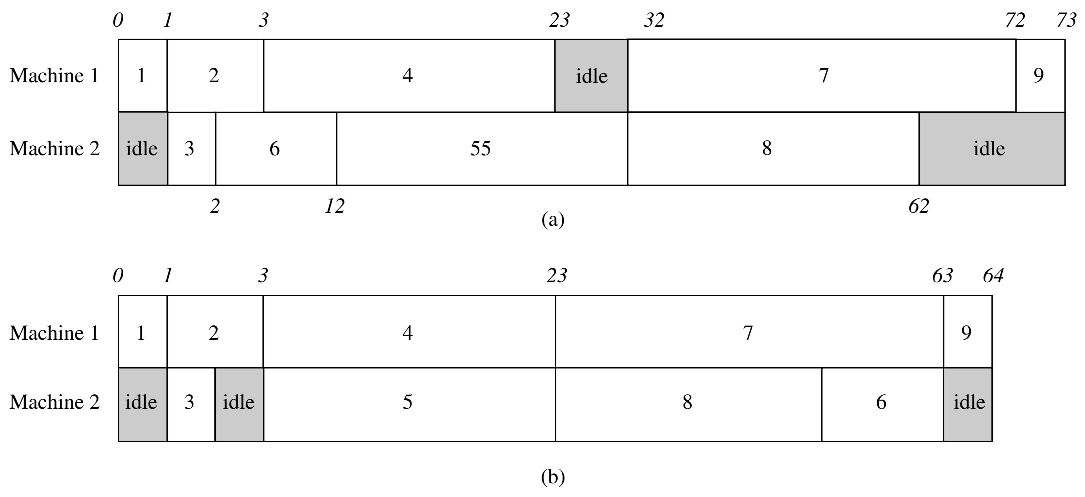

Figure 2 Gantt Chart Displaying Two Schedules for the Problem in Figure 1
Note. (a) Schedule where task 6 assigned right away—makespan 73 ut. (b) Schedule where task 6 waits—makespan 64 ut.

their method produces solutions very close to optimal. Rajendran and Holthaus (1999) do a comparative study of various dispatching rules in dynamic flowshops and job-shops.

Hopfield and Tank (1985) pioneered the application of neural networks for solving combinatorial optimization problems. Their approach is based on defining and minimizing "energy" functions that capture the constraints and the objective function of the problem. When the "energy" function converges to a minimum, a good solution is found. Using this novel approach, Hopfield and Tank (1985) solved the traveling-salesman problem. Dahl (1987) used a similar neural-network approach for solving the graph-coloring problem.

Foo and Takefuji (1988a, b) were the first to apply the Hopfield network for the job-shop-scheduling problem. The problems they worked on, however, were extremely small—a 2-job, 3-machine problem and the computational complexity was high ($O(m^2n^2 + mn)$). Satake et al. (1994) used a modified Hopfield network to solve bigger problems, up to 14 jobs and seven machines. The computation time remains poor at about 700 seconds for a 14 job, seven-machine problem. Most recently, Sabuncuoglu and Gurgun (1996) reported yet another variation of the Hopfield network. Computational times, however, remain very poor; for example, a 15-task 5-machine problem took about 2,300 seconds to reach a near optimum solution.

The Hopfield and Tank approach suffers from some limitations. First, it is not satisfactory for large problems because it often finds suboptimal local minima. Most reported results are for extremely small problems. The second limitation lies in the problem formulation itself. Formulating the energy function for an optimization problem incorporating the various constraints is not straightforward and still remains an art. The third difficulty is computational complexity. The existing Hopfield-neural-network approaches take significant processing time and thousands of iterations for convergence. 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 Aytug et al. (1994) and Sabuncuoglu (1998). Machine-learning techniques other than neural networks have also been proposed in the literature. For example Sakawa and Kubota (2000) use genetic algorithms for scheduling problems with fuzzy processing times and due dates. Candido et al. (1998) also use genetic algorithms for job-shop problems. Steinhofel et al. (1999) propose two simulated-annealing-based heuristics for the job-shop problem. Miller et al. (1999) propose a hybrid genetic-algorithm approach for a single-machine scheduling problem.

Agarwal et al. (2003), proposed an augmented neural-network approach that is a hybrid of greedy heuristics and the neural-network approach. This approach has been shown to work on large problems without any deterioration in the solution quality or the computational time taken. The AugNN approach makes use of the properties of neural networks, yet is different from the prior neural-network approaches used for solving optimization problems. A critical feature of this approach is the one-to-one correspondence between the problem structure and the neural-network structure, thus allowing the neural network to be augmented by the relevant domain-specific knowledge in solving the problem.

4. AugNN and AugNN-GNG

Because this paper extends the AugNN framework, we briefly describe the framework and then explain our GNG extension. Details of AugNN can be found in the online supplement to this paper on the journal's website or in Agarwal et al. (2003). The AugNN approach, as previously mentioned, is a hybrid of heuristic approaches and the neural-network approach. A brief description of neural networks follows.

Neural networks, in general, constitute a class of artificial-intelligence learning techniques for nonlinear transformation of some input (independent variables) to an output (dependent variable). The transformation function is not a simple algebraic function, but is embedded in the structure of the neural network, which consists of what are called *processing elements* (PEs). These PEs are arranged in three (or more) layers—an input layer, one (or more) hidden layer(s), and an output layer. PEs in one layer are linked (or connected) to PEs in other layers or in the same layer through links. These links have "weights" associated with them, which factor into the transformation. With the help of input, activation (or transfer), and output functions associated with PEs and weights associated with links, a neural network transforms an input to some output. The weights are modified using a learning rule, such that the output becomes closer to the desired value in subsequent iteration(s). The network thus adaptively learns the transformation function to produce the most desirable output.

The AugNN approach exploits this iterative adaptive learning scheme to find improved solutions to the task-scheduling problem. It basically sets up a neural network based on the task graph of the given problem so that each task and each machine becomes a PE. See Figure 3 for an NN structure for a task-scheduling problem. These PEs are assigned input, activation, and output functions, and the links between PEs are assigned weights in such a way that the network is able to find a feasible solution to the task-scheduling problem in one iteration. The heuristic is built into the input and activation functions of the machine nodes. Suitable "learning rules" are also proposed to modify weights such that better makespans are generated in subsequent iterations.

The mathematical formulation of the input, activation, and transfer functions for task and machine nodes are given in the online supplement to this paper. The input function of task nodes is designed to count the number of predecessor tasks completed. The activation function decides the state of the task. The initial state for all tasks is 1. A task reaches state 2 when its input-function value equals its threshold value, which equals the total number of predecessor tasks. When a task is assigned to a machine, state 3 is reached and when it is completed, it attains state 4. The output function of the task sends a positive unit signal whenever the task is completed. The output of the final node is the makespan.

The input function of the machine node is the sum of three components. The first is the weighted output from a ready task. Two others are inhibitory (large negative) signals from other machines. The inhibitory signals are designed to ensure that a machine cannot be assigned to another task if it is already processing a task, and that the task it is processing does not get assigned to another machine. A task node knows that it has been assigned to a machine when it receives an inhibitory signal from the machine node to which it is assigned. Absent such a signal, the task will assume it is ready to start and may look for available machines. The activation function of the machine node is in two parts. The first decides which machine is assigned to which task. The greedy heuristic is built into AugNN through this part of the activation function with the help of the *TaskHeuristicParameter.* For example, the *TaskHeuristicParameter* for the *highest-level-first* heuristic is the "level" of the task, while that for the *longest-processing-time* heuristic is the processing time of the task. Tasks compete for limited machines based on the value of the activation function, which in turn depends on the heuristic used. The second part of the activation function decides the state of a machine. A machine node can be in one of six states: (1) available, (2) busy (just assigned), (3) busy, (4) just released, (5) assigned to another task, and (6) finished processing a task. The output function of a machine node fires a signal to the forward task node whenever it is released, i.e., when its own state becomes 4. It fires inhibitory signals to other machines when it is assigned a task. With the help of these functions, a single pass (iteration) generates a feasible solution.

With this background on neural networks and the AugNN framework for solving task-scheduling problems, we will now describe the changes proposed in this paper. First we will describe how the nongreedy heuristic is implemented, followed by how the greedy and nongreedy heuristics are combined.

4.1. How Is the Nongreedy Heuristic Implemented?

We propose three changes to the mathematical formulation of the AugNN approach. First, we modify the conditions under which a task node goes from state 1 to state 2 (recall that state 1 means "not ready" while state 2 means "ready"). In the original AugNN framework, in the activation function of the task nodes, the only condition for a task to enter state 2 was that all its preceding tasks had to be completed. We add another condition as follows:

$$(\omega_{NG})_{ij} \cdot (t + LET_j) > LST_j + LET_j \quad \forall i \in M, j \in T$$

where $(\omega_{NG})_{ij}$ is some nongreedy weight factor from task j to machine i, with an initial value of 1, t is the elapsed time, LST_j is the *latest start time* for the task in question under an infinite-resource assumption, and LET_j is the "level with estimated time" or the length of the longest remaining path, including

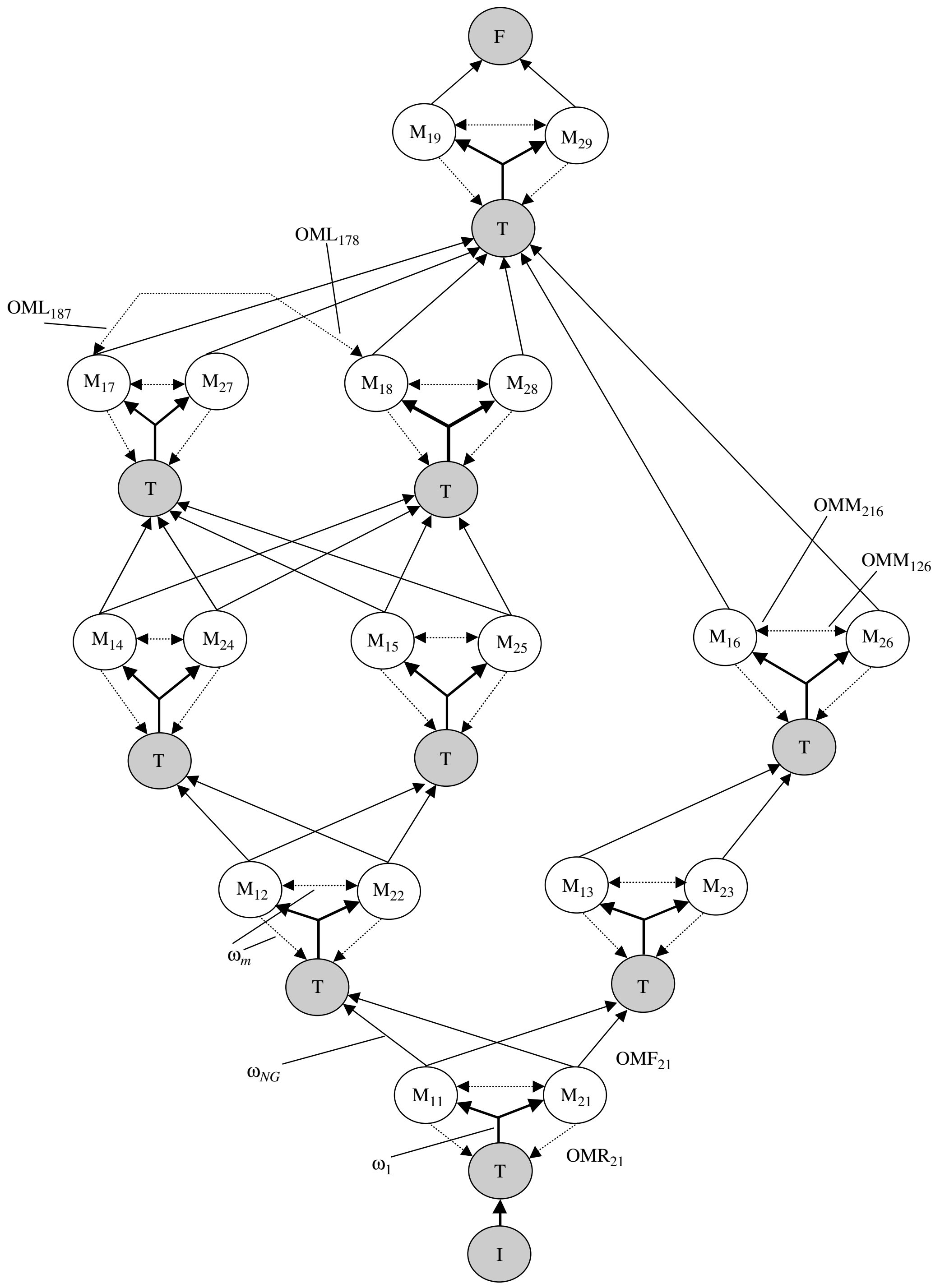

Figure 3 The AugNN Network for the Example Problem of Figure 1

the processing time of task j. This additional condition (i.e., in addition to the obvious one that the preceding tasks should be completed), allows a task to be assigned only if $(\omega_{NG})_{ij} \cdot (t + LET_j) > LST_j + LET_j$. To understand this condition, assume that $(\omega_{NG})_{ij} = 1$, and that task j's preceding tasks are complete and $t < LST_j$. This means that task j has some slack and can wait for some time without affecting the overall makespan. If $t = LST_j$, then the current job becomes critical and should not be allowed to wait. The role of $(\omega_{NG})_{ij}$ is to vary the likelihood of waiting in different iterations. If $(\omega_{NG})_{ij} > 1$, then task j need not wait till it becomes critical because $(\omega_{NG})_{ij} \cdot (t + LET_j)$ might exceed $LST_j + LET_j$ before t equals LST_j. So, the higher the $(\omega_{NG})_{ij}$, the lower the likelihood of forcing a wait decision on a task. This new condition is implemented by adding a weight factor to the links between machine nodes and task nodes and changing the activation function for task nodes.

The second change is in the *TaskHeuristicParameter* of the Activation function of the machine node. In the greedy heuristic the *TaskHeuristicParameter* was one of either *LET* (level with estimated time) or *L* (level) or *PT* (processing time), etc. In the nongreedy heuristic, we replace the *TaskHeuristicParameter* by $t + LET$. In the nongreedy approach, we do not want to try the various dispatching rules, but only one dispatching rule, *LET*. But we want to add the elapsed time t to *LET*. The addition of elapsed time helps in giving higher priority to tasks that have consumed more of their slack because as t approaches *LST*, the slack approaches zero.

The third change deals with the learning rule of the new weight factor. The weight factor is modified in such a way that if "waiting" does not help the solution, then in subsequent iterations the weight will be higher as a function of the error and processing time of the task. By doing so, the probability of "wait" in subsequent iteration is reduced. For each task,

$$(\omega_{NG})_{k+1} = (\omega_{NG})_k + Waited_k \cdot \alpha \cdot PT \cdot \varepsilon_k$$

where k is the iteration number, ε_k is the error in the kth iteration (between the obtained solution and the lower bound), *PT* is the processing time of the task, α is the learning rate, and $Waited_k$ is 1 if the task waited in iteration k, and is 0 otherwise. In other words, $Waited_k$ becomes 1 if the task remained in state 1 due to the added condition.

With the help of these three changes, a nongreedy algorithm can be implemented using the AugNN framework. Results using this nongreedy-only AugNN were better than five out of six greedy-only AugNN results. However, the results of the best greedy heuristic (*HLETF*) were far superior to the nongreedy approach. Can we do better than the best greedy heuristic by combining it with the nongreedy heuristic? Two alternative strategies are proposed here. Both strategies worked better than greedy or nongreedy alone, although one strategy appeared to be more robust than the other.

4.2. Combining Greedy and Nongreedy Heuristics

In the first strategy, we retain the extra condition (discussed above) for a task to become a state 2 task, but instead of using the *TaskHeuristicParameter* of the nongreedy type, use the one for the greedy type. This approach gave better results than a greedy or nongreedy heuristic alone, for all six greedy heuristics tried. The improvement, although statistically significant, was not very high in magnitude.

Our second strategy was guided by the observation that when the nongreedy approach was used alone, it gave the best solution in under 10 iterations on average compared to about 23 for the greedy-only heuristics. Recall that this best solution was better than five out of six greedy heuristics' best solution. So, for our second approach, we simply run the first 10 iterations as nongreedy-only AugNN, and use the best solution from these 10 iterations to run the greedy-only algorithm for the remaining iterations. Using the second strategy, the overall results were far superior to any of the previous results, although for one heuristic the first strategy worked better.

5. Empirical Work and Results

We use three sets of problems. There are 344 problems ranging in size from 10 tasks and 2 machines to 100 tasks and 5 machines in each set. The processing times for the first set were generated randomly using a uniform distribution. For the other two sets, the task graphs are the same but the processing times are drawn from exponential and normal distributions. So, there are a total of 1,032 problems. We will call the three sets of problems the "uniform set," the "normal set," and the "exponential set." Table 1 summarizes the distribution of these problems by size.

We ran several experiments on all three sets of problems. We used six greedy heuristics, namely, (1) *highest level first* (*HLF*), (2) *highest level with estimated time first* (*HLETF*), (3) *critical path with most immediate successors first* (*CP/MISF*), (4) *shortest path time* (*SPT*), (5) *longest processing time first* (*LPT*), and (6) *RANDOM*. These heuristics were implemented using the appropriate *TaskHeuristicParameter* in the

Table 1 Size Distribution of Problems Used

	Number of machines			
Number of tasks	2	3	4	5
10	24	0	0	0
11	24	0	0	0
12	24	0	0	0
13	24	0	0	0
14	24	24	0	0
15	24	24	0	0
16	24	24	0	0
20	24	24	0	0
25	24	24	0	0
30	24	24	0	0
40	24	24	24	0
50	24	24	24	24
60	24	24	24	24
70	24	24	24	24
80	24	24	24	24
90	24	24	24	24
100	24	24	24	24
Total	408	312	168	144

Note. The total number of problems used for empirical work is 1,032. A third of these have processing times from a uniform distribution, a third from a normal distribution, and a third from an exponential distribution.

Table 2a **Aggregate Results for the Uniform Set**

Greedy heuristic used	Learning rule parameter used	Gap[1] with single pass	Gap with greedy-only AugNN[2]	Gap with AugNN-GNG	Improvement AugNN-GNG over single pass (%)	Improvement AugNN-GNG over greedy-only AugNN (%)	Percent gap[3] with single pass (%)	Percent gap with greedy-only AugNN (%)	Percent gap with AugNN-GNG (%)	CPU time in seconds for greedy-only AugNN	CPU time in seconds for AugNN-GNG
HLF	*Level*	7101	4252	3246	54.3	23.7	5.0	3.0	2.3	916.4	790.7
	LET	7101	4440	3255	54.2	26.7	5.0	3.1	2.3	925.8	752.4
HLETF	*LET*	4304	2822	2531	41.2	10.3	3.0	2.0	1.8	886.1	772.4
CPMISF	*CPMISF*	8977	4657	3285	63.4	29.5	6.3	3.3	2.3	1187.7	864.3
	LET	8977	5422	3555	60.4	36.83	6.3	3.8	2.4	1241.1	752.4
SPT	*SPT*	27474	11809	4032	85.3	65.9	19.3	8.3	2.8	1079.7	864.3
	LET	27474	11706	4027	85.3	65.6	19.3	8.2	2.8	1018.6	752.4
LPT	*LPT*	15125	5900	3572	76.4	39.5	10.6	4.1	2.5	1170.3	791.3
	LET	15125	6325	3606	76.2	43.0	10.6	4.4	2.5	1102.3	757.8
RANDOM	*RANDOM*	13218	4489	3354	74.6	25.3	9.3	3.2	2.4	1394.9	853.5
	LET	13218	4466	3338	74.7	25.3	8.9	3.1	2.3	1315.0	760.2

Note. These results are aggregate over 344 problems where processing times were drawn from a uniform distribution.
[1]Gap between obtained solution and the lower bound solution. The lower bound solution for these problems was 142,490.
[2]Gap with nongreedy only AugNN was 4049 or 2.8%.
[3]Gap expressed as percentage of lower bound.

Table 2b **Aggregate Results for the Normal Set**

Greedy heuristic used	Learning rule parameter used	Gap[1] with single pass	Gap with greedy-only AugNN[2]	Gap with AugNN-GNG	Improvement AugNN-GNG over single pass (%)	Improvement AugNN-GNG over greedy-only AugNN (%)	Percent gap[3] with single pass (%)	Percent gap with greedy-only AugNN (%)	Percent gap with AugNN-GNG (%)	CPU time in seconds for greedy-only AugNN	CPU time in seconds for AugNN-GNG
HLF	*Level*	2482	1359	1273	48.7	6.3	4.4	2.4	2.2	683.7	613.7
	LET	2482	1386	1260	49.2	9.1	4.4	2.4	2.2	676.3	593.7
HLETF	*LET*	1781	1177	1140	36.0	3.2	3.1	2.1	1.9	729.8	610.3
CPMISF	*CPMISF*	3162	1831	1427	54.9	22.1	5.6	3.2	2.5	967.5	697.3
	LET	3162	1795	1372	56.6	23.6	5.6	3.2	2.4	994.6	597.3
SPT	*SPT*	11104	3835	1700	84.7	55.7	19.5	6.8	3.0	999.1	622.9
	LET	11104	3845	1694	84.7	55.9	19.5	6.8	3.0	936.2	593.5
LPT	*LPT*	6196	2219	1440	76.8	35.1	10.9	3.9	2.5	1011.9	639.0
	LET	6196	2252	1477	76.2	34.4	10.9	4.0	2.6	967.5	595.4
RANDOM	*RANDOM*	4811	1847	1398	70.9	24.3	8.5	3.3	2.5	1155.9	725.4
	LET	4979	1792	1376	72.4	23.2	8.5	3.2	2.4	1100.4	598.8

Note. These results are aggregate over 344 problems where processing times were drawn from a normal distribution.
[1]Gap between obtained solution and the lower bound solution. The lower bound solution for these problems was 56,804.
[2]Gap with nongreedy only AugNN was 1,624 or 3.0%.
[3]Gap expressed as percentage of lower bound.

Table 2c **Aggregate Results for the Exponential Set**

Greedy heuristic used	Learning rule parameter used	Gap[1] with single pass	Gap with greedy-only AugNN[2]	Gap with AugNN-GNG	Improvement AugNN-GNG over single pass (%)	Improvement AugNN-GNG over greedy-only AugNN (%)	Percent gap[3] with single pass (%)	Percent gap with greedy-only AugNN (%)	Percent gap with AugNN-GNG (%)	CPU time in seconds for greedy-only AugNN	CPU time in seconds for AugNN-GNG
HLF	*Level*	11013	6605	3825	65.3	42.1	7.4	4.4	2.6	932.7	727.6
	LET	11013	6802	3815	65.4	43.9	7.4	4.5	2.5	942.0	738.0
HLETF	*LET*	4109	2683	2404	41.5	10.4	2.7	1.8	1.6	845.6	756.1
CPMISF	*CPMISF*	10388	5041	3273	68.5	35.1	6.9	3.4	2.2	1246.4	688.1
	LET	10388	6375	3573	65.6	44.0	6.9	4.3	2.4	1277.6	742.0
SPT	*SPT*	34142	13093	4378	87.2	66.6	22.8	8.7	2.9	1063.0	805.7
	LET	34142	13312	4394	87.1	67.0	22.8	8.9	2.9	1004.5	737.3
LPT	*LPT*	16930	6913	3676	78.3	46.8	11.3	4.6	2.5	1167.1	759.2
	LET	16930	7097	3752	77.8	47.1	11.3	4.7	2.5	1093.9	740.5
RANDOM	*RANDOM*	15043	4571	3345	77.8	26.8	10.1	3.1	2.2	1388.5	781.7
	LET	15744	4579	3286	79.1	28.2	10.5	3.1	2.2	1303.9	742.5

Note. These results are aggregate over 344 problems where processing times were drawn from an exponential distribution.
[1]Gap between obtained solution and the lower bound solution. The lower bound solution for these problems was 149,672.
[2]Gap with nongreedy-only AugNN was 4,881 or 3.2%.
[3]Gap expressed as percentage of lower bound.

activation function of the machine nodes. We combined each of the six greedy heuristics with the nongreedy heuristic in two different ways, as discussed in Section 4.2. For each of these twelve combinations, we tried two different learning rules.

Results are tabulated in Tables 2 and 3. Table 2 is divided into three parts (a, b, and c), one for each set of problems. For the uniform set (Table 2a), the lower bound for all 344 problems combined is 142,490 ut. The best single-pass greedy heuristic (*HLETF)* gives a gap (obtained solution minus lower bound) of 4,304 while the worst (*SPT)* is 27,474. As a percentage of the lower bound these gaps range from 3% for *HLETF* to 19.3% for *SPT*. Greedy-only AugNN approach

Table 3 Cases with Known Optimum Solution[1] for the Uniform Set of 344 Problems

Greedy heuristic used	Learning rule parameter used	With single-pass greedy heuristic	Expressed as percentage (%)	With greedy-only AugNN	Expressed as percentage (%)	With AugNN-GNG	Expressed as percentage (%)
HLF	*Level*	6	1.74[2]	68	19.77	70	20.35
	LET	6	1.74	54	15.70	72	21.28
HLETF	*LET*	11	3.20	79	22.97	92	26.08
CP/MISF	*CP/MISF*	12	3.49	73	21.22	76	22.40
	LET	12	3.49	55	15.99	73	21.22
SPT	*SPT*	1	0.29	12	3.49	35	10.18
	LET	1	0.29	14	4.07	36	10.47
LPT	*LPT*	5	1.45	44	12.79	65	18.89
	LET	5	1.45	39	11.34	59	17.14
Random	*Random*	4	1.16	87	25.29	87	25.29
	LET	5	1.45	87	25.29	87	25.29

[1]A solution is known to be optimum if the solution equals the lower bound.
[2]6 as a percentage of 344 is 1.74%

reduces these gaps to 2.0% for *HLETF* to 8.3% for *SPT*. The nongreedy-only AugNN approach reduces the gap to 2.8%, which is better than five out of six greedy heuristics. AugNN-GNG further reduces the gap to 1.8% for *HLETF* to 2.8% for *SPT*. These results are extremely encouraging. We wanted to test the robustness of our algorithms on more problem sets.

For the normal set (Table 2b), the total lower bound for all problems is 56,804. Single-pass heuristic gaps range from 3.1% for *HLETF* to 19.5% for *SPT*. Greedy-only AugNN reduce these gaps to 2.1% for *HLETF* to 6.8% for *SPT*. The nongreedy-only AugNN approach reduces the gap to 3.0%, better than four out of six greedy heuristics. AugNN-GNG further reduces the gap to 1.9% for *HLETF* to 3.0% for *SPT*.

For the exponential set (Table 2c), the total lower bound for all problems is 149,672. Single-pass heuristics give gaps of 2.7% for *HLETF* to 22.8% for *SPT*. Greedy-only AugNN reduces these gaps to 1.8% for *HLETF* to 8.9% for *SPT.* Nongreedy-only AugNN approach reduces the gap to 3.2%, which is better than four out of six greedy heuristics. AugNN-GNG further reduces the gap further to 1.6% for *HLETF* to 2.9% for *SPT*. Results over all three sets are very similar and they sufficiently illustrate the robustness of the AugNN-GNG approach.

Another criterion used to compare AugNN-GNG with greedy-only AugNN and single-pass greedy heuristics is the "number of known cases with optimal solution" (Table 3). For the uniform set, while single-pass heuristics gave from one known optimal solution for *SPT* to 11 for *HLETF*, greedy-only AugNN gave 12 known optimal solutions for *SPT*, 79 for *HLETF* and 87 for *RANDOM*. AugNN-GNG gave as many as 35 known optimal solutions for *SPT* and as high as 92 for *HLETF*. Results for other two sets were similar and are not reported to conserve space.

The average number of iterations needed for the best solution is 20 for all three sets.

The computational complexity of the AugNN-GNG approach is $O(n^2 + mn)$, the same as that of AugNN and of single-pass heuristics. The only difference is that the AugNN approaches require a few iterations whereas single-pass heuristics, by definition, require just one. However, the number of iterations required are very few and do not increase with the problem size. The computational complexity of NN based on the 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 approach can range up to several thousands (up to 3,000 in the case of Lo and Bavarian) compared to ours, which averaged below 20.

6. Discussion

It is important to note that in the AugNN approach, the architecture of the neural network is tied to the structure of the given scheduling problem. So by design, the neural-network structure will change for each problem. The training, which affects the weights of the links, therefore applies only to a particular instance of the problem. In other words, the trained network may not be used to solve new and unseen problems. However, an automated tool, which is part of the implementation of our approach, creates a new structure for a problem in real time. Because the training lasts for fewer than 100 iterations, the training time is negligible.

While our approach is used for solving the resource-constrained task-scheduling problem with identical machines with the objective of minimizing the makespan, this framework can be extended for other types of scheduling problems. For example, the case of nonidentical machines can be considered by simply treating the machine nodes as those of nonidentical machines. For example, in Figure 3, machine nodes M_{11} and M_{21}, which represent nodes for machines 1 and 2 for task 1, respectively, could be treated as nonidentical. The different processing times will also play a part in the competition between which machine is assigned to job 1. The case of job-shop scheduling can also be handled by modifying the architecture of the links between the machine and task nodes to reflect the sequence of operations on various jobs. The approach can also be applied to other operations research problems such as the traveling salesman problem. Each machine node can be treated as a city and the distance to the city is equivalent to the processing time.

7. Summary and Conclusions

We have proposed very robust algorithms through an extension of the AugNN approach proposed by Agarwal et al. (2003) to solve the task-scheduling problem. The problem is to minimize the makespan for scheduling n tasks on m identical machines, where the tasks follow a predetermined precedence order and are nonpreemptive. The AugNN approach is a hybrid of the heuristic and the neural-network approaches. We propose a nongreedy heuristic in which tasks on noncritical paths with large slack are forced to wait in spite of machine availability, to give more critical tasks a chance to get assigned. Using the nongreedy heuristic with the AugNN approach, the results were better than most of the greedy heuristics used with AugNN. We then combine the nongreedy heuristic with one of many greedy heuristics (AugNN-GNG) and find solutions better than the best greedy heuristic.

We have demonstrated how a nongreedy and a greedy heuristic can be combined in the AugNN framework to get significantly improved results, irrespective of the greedy heuristic used. The proposed extensions were tested on over a thousand problems. Although the greedy-only AugNN approach has been shown to reduce the gap between the obtained solution and the lower-bound solution by as much as 66% compared to some single-pass heuristics, AugNN-GNG was able to reduce the gap further by as much as 65% for some heuristics. Also, the new approach is able to find solutions, on average, within 1.8% to 2.8% of the lower bound compared to 2.0% to 8.3% for the greedy-only AugNN approach and 3.0% to 19.3% for the single-pass greedy-heuristic approaches. This improvement is obtained without any increase in either computational complexity or time. The AugNN approach could be used for other problems such as the job-shop scheduling, open-shop scheduling, independent-tasks scheduling, etc.

References

Adam, T. L., K. M. Chandy, J. R. Dickson. 1974. A comparison of list schedules for parallel processing systems. *Comm. ACM* **17** 685–690.

Agarwal, A., V. S. Jacob, H. Pirkul. 2003. Augmented neural networks for task scheduling. *Eur. J. Oper. Res.* **151** 481–502.

Aytug, H., S. Bhattacharya, G. J. Koehler, J. L. Snowdon. 1994. A review of machine learning in scheduling. *IEEE Trans. Engrg. Management* **41** 165–171.

Candido, M. A. B., S. K. Khator, R. M. Barchia. 1998. A genetic algorithm based procedure for more realistic job shop scheduling problems. *Internat. J. Production Res.* **36** 3437–3457.

Casavant, T. L., J. G. Kuhl. 1988. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Software Engrg.* **14** 141–154.

Cheng, T. C. E., C. C. S. Sin. 1990. A state-of-the-art review of parallel-machine scheduling research. *Eur. J. Oper. Res.* **47** 271–292.

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

Dahl, E. D. 1987. Neural network algorithm for an NP-complete problem: Map and graph coloring. *Proc. 1st Joint Conf. Neural Network III*, San Diego, CA, 113–120.

Foo, Y. P. S., Y. Takefuji. 1988a. Stochastic neural networks for solving job-shop scheduling: Part 1. *Proc. Joint Internat. Conf. Neural Networks* **2** 275–282.

Foo, Y. P. S., Y. Takefuji. 1988b. Stochastic neural networks for solving job-shop scheduling: Part 2. *Proc. Joint Internat. Conf. Neural Networks* **2** 283–290.

Gass, S. I., C. M. Harris. eds. 2001. *Encyclopedia of Operations Research and Management Science*. Kluwer, Boston, MA.

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

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

Hu, T. C. 1961. Parallel sequencing and assembly line problems. *Oper. Res.* **9** 841–848.

Kasahara, H., S. Narita. 1984. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.* **C-33** 1023–1029.

Kasahara, H., S. Narita. 1985. Parallel processing of robot-arm control computation on a multimicroprocessor system. *IEEE J. Robotics Automation*. **RA-1** 104–113.

Lo, Z. P., B. Bavarian. 1993. Multiple job scheduling with artificial neural networks. *Comput. Electr. Engrg.* **19** 87–101.

Miller, D. M., H. C. Chen, J. Matson, Q. Liu. 1999. A hybrid genetic algorithm for the single machine scheduling problem. *J. Heuristics* **5** 437–454.

Panwalker, S. S., W. Iskander. 1977. A survey of scheduling rules. *Oper. Res.* **25** 45–61.

Potts, C. N., M. Y. Kovalyov. 2000. Scheduling with batching: A review. *Eur. J. Oper. Res.* **120** 228–249.

Rajendran, C., O. Holthaus. 1999. A comparative study of dispatching rules in dynamic flowshops and job shops. *Eur. J. Oper. Res.* **116** 156–170.

Ramamoorthy, C. V., K. M. Chandy, M. J. Gonzalez. 1972. Optimal scheduling strategies in a multiprocessor system. *IEEE Trans. Comput.* **C-21** 137–146.

Rinnooy Kan, A. H. G. 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague, The Netherlands.

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

Sabuncuoglu, I., B. Gurgun. 1996. A neural network model for scheduling problems. *Eur. J. Oper. Res.* **93** 288–299.

Sakawa, M., R. Kubota. 2000. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy due date through genetic algorithms. *Eur. J. Oper. Res.* **120** 393–407.

Satake, T., K. Morikawa, N. Nakamura. 1994. Neural network approach for minimizing the makespan of the general job shop. *Internat. J. Production Econom.* **33** 67–74.

Steinhofel, K., A. Albrecht, C. K. Wong. 1999. Two simulated annealing-based heuristics for the job shop scheduling problem. *Eur. J. Oper. Res.* **118** 524–548.

Zhang, H. C., S. H. Huang. 1995. Application of neural networks in manufacturing: a state of the art survey. *Internat. J. Production Res.* **33** 705–728.