Selecting Scheduling Heuristics Using Neural Networks

Jatinder N. D. Gupta
Department of Management, Ball State University
Muncie, IN 47306, USA; E-mail: jgupta@bsu.edu

Randall S. Sexton
Computer Information Systems, Southwest Missouri State University
Springfield, MO 65804, USA; E-mail: rss000f@mail.smsu.edu

Enar A. Tunc
Department of Management, Ball State University
Muncie, IN 47306, USA; E-mail: etunc@bsu.edu

Received: September 1998; revised: May and November 1999;
accepted: January 2000

This paper discusses the application of neural networks to select the best heuristic algorithm to solve a given scheduling problem. The two-stage hybrid flowshop with multiple identical parallel machines at the second stage is used as an example to discuss the process of selecting a scheduling heuristic through a neural-network approach. This paper uses the genetic-algorithm-based approach for training the neural network and shows that the suggested neural-network approach is quite effective and efficient for selecting the best heuristic algorithm for solving a given scheduling problem.

Subject Classification: Production Scheduling, Neural Networks, Artificial Intelligence.

Other key words: Two-tier neural networks, genetic algorithms, hybrid flowshop scheduling, parallel machines, heuristic selection.

Most combinatorial optimization problems, such as scheduling problems, are NP-hard in the strong sense, implying that it is highly unlikely that a polynomial algorithm can be designed to solve these problems. Realizing this, heuristic algorithms are developed and used to solve combinatorial optimization problems representing real-world applications. In the development and testing of these heuristic algorithms, statistical-inference techniques, like the analysis of variance, are used to conclude that one heuristic is superior to another. An ideal result of such statistical analysis would be to have evidence of significant differences among the heuristic algorithms in terms of the desired performance measure. However, when two-or three-way interactions of problem characteristics significantly affect these desired performance measures, it is impossible to identify the best heuristic algorithm for a specific instance of these combinations of problem characteristics. Thus, such deductions based on statistical-inference techniques, while valid for an aggregation of several problem instances, are ineffective in selecting the best heuristic algorithm to solve a specific instance of a given combinatorial problem since a heuristic that performs best on average may perform quite poorly for a specific instance. Therefore, we need to explore different methodologies for the selection of a heuristic algorithm to solve a specific instance of a combinatorial optimization problem.

The problem of selecting the best heuristic algorithm, when several of them exist for solving a hard combinatorial optimization problem, is a prediction problem. Since neural networks are most effective in solving classification and prediction problems[2], the emerging technology of neural networks may be well-suited to solve this problem. Even though this is an important problem in many practical applications, only a few papers explore the use of neural networks to select heuristic algorithms. Nygard, Juell, and Kabada[13] and Tuzun, Magent, and Burke[17] report on a successful application of a neural-network system in identifying the best algorithm to solve a class of vehicle-routing problems.

Gupta and Tunc[8] describe the application of neural networks to select a heuristic algorithm to solve a two-stage-hybrid-flowshop problem to minimize makespan. However, their approach requires the construction of a neural network for each problem size. That is, for a seven-job problem, one needs to develop a different neural network than a neural network for a five-job problem. Consequently, it is not practical to use their approach

to solve scheduling problems with various number of jobs. Further, Gupta and Tunc [8] used backpropagation to train the neural networks, which has the drawback of producing poor-quality results in several instances due to its inability to escape from local minima[1].

In this paper, we propose to use neural-network systems for evaluating and selecting the best algorithms for solving a well-defined NP-hard scheduling problem. As an illustrative example of our proposed approach, we consider the two-stage-hybrid-flowshop-scheduling problem where the first stage contains only one machine and the second stage consists of multiple identical parallel machines, and the objective is to minimize the total number of jobs delivered after their due dates. Our proposed approach differs from existing approaches in three ways. First, contrary to Gupta and Tunc's[8] approach, we represent a scheduling problem by a few chosen problem characteristics to create only one neural network for any size problem, enabling us to solve problems with *any* number of jobs and machines. Second, we develop a *two-tier neural network* that enables us to select the best heuristic algorithm in stages. Third, we train the proposed networks using the genetic-algorithm-based approach that avoids the pitfalls of backpropagation.

The rest of the paper is organized as follows: Section 1 briefly describes the two-stage-hybrid-flowshop-scheduling problem with the objective of minimizing the total number of jobs delivered after their due dates. This section also describes various heuristic algorithms to find an approximate solution to this problem. The transformation of an algorithm-selection problem to a prediction problem is discussed in section 2, where the neural network (NN) architecture, the manner in which it is applied to the selection of heuristic algorithms, and the genetic algorithm (GA) configuration used to train the NN are also outlined. The results of the experiments in training and testing the effectiveness of the proposed neural-network system are reported in section 3. Finally, section 4 concludes the paper with some recommendations for future research in using neural networks for algorithm selection.

1 Heuristics for the Hybrid Flowshop Problem

In this section, we describe the two-stage-hybrid-flowshop problem and heuristic algorithms available to solve this problem.

1.1 The Hybrid Flowshop Problem

The two-stage-hybrid-flowshop-scheduling problem considered in this paper may be described as follows: a given set $N = \{1, 2, \ldots, n\}$ of n jobs is to be processed on two stages, with only one machine at stage 1 and m identical parallel machines at the second stage. Each job $i \in N$ is to be processed first on stage 1 and then, on stage 2, requires a_i and b_i time units of processing at stage 1 and 2 respectively, and is due by its due date d_i. If job i is completed after its due date, it is called *tardy*, otherwise it is considered *early*.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a sequence of jobs at the first stage and $\pi_h = (\pi_h(1), \pi_h(2), \ldots, \pi_h(k_h))$ be an assignment and sequence of jobs at machine h of the second stage such that $\sum_{h=1}^{m} k_h = n$. Then, the completion times of job $\pi(i)$ at the first and second stage in the schedule $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, $C_{1,\pi(i)}$ and $C_{2,\pi(i)}$, are given by the following expressions:

$$C_{1,\pi(i)} = \sum_{s=1}^{i} a_{\pi(s)} \tag{1}$$

$$C_{2,\pi(i)} = \max\Big\{C_{1,\pi_h(q)}; C_{2,\pi_h(q-1)}\Big\} + b_{\pi(i)} \tag{2}$$

where $\pi_h(q) = \pi(i)$ and $C_{2,\pi_h(0)} = 0$ for all $h = 1, \ldots, m$.

Now, define a binary variable $U_{\pi(i)}$ which is 1 if the completion time of job $\pi(i)$ calculated by using equations (1) and (2), $C_{2,\pi(i)} > d_{\pi(i)}$ (i. e., if job $\pi(i)$ is tardy), and is 0 otherwise. Then, following the three-parameter notation of scheduling problems described by Lawler *et al.*[10] , we designate this problem as a $F2|P(1,m)|\sum U_i$ problem, where $F2$ indicates that it is a two-stage flowshop, $P(1,m)$ indicates that stage 1 contains only one machine while stage 2 contains m identical parallel machines, and $\sum U_i$ indicates that the objective is the minimization of the total number of tardy jobs. Specifically, the $F2|P(1,m)|\sum U_i$ problem is one of finding a schedule $\pi = (\pi(1), \ldots, \pi(n))$ at the first stage and schedule $\pi_h = (\pi_h(1), \pi_h(2), \ldots, \pi_h(k_h))$ at each machine h of the second stage such that $\sum_{i=1}^{n} U_{\pi(i)}$ is minimum.

1.2 Heuristic Algorithms

The $F2|P(1,m)|\sum U_i$ problem is clearly NP-hard in the strong sense since Lenstra *et al.*[11] have shown that its special case, the $F2||\sum U_i$ problem is NP-hard in the strong

sense. Realizing this, Gupta and Tunc[7] propose several heuristic algorithms to find near-optimal schedules for the $F2|P(1,m)|\sum U_i$ problem. These heuristic algorithms first create a schedule S and then assign jobs to various machines at both stages. As in Moore's[12] algorithm for the single-machine case, the assignment is done on a first-come-first-serve basis except that a decision rule T is used to reject (make it tardy) a job whenever the first tardy job is encountered. This process is continued until all jobs are assigned. The basic steps of such a heuristic algorithm are as follows:

Algorithm *ST*

Input: a_i, b_i, d_i for $i \in N = \{1, 2, \ldots, n\}$.

Step 1. Using an appropriate sequencing rule, determine a list schedule $S = (s(1), s(2), \ldots, s(n))$. Set $M_1 = \cdots = M_m = 0$. Let $\sigma_1 = \cdots = \sigma_m = \{\phi\}$, $A = 0$, $i = 1$, and $e = 0$. Enter step 2.

Step 2. Select machine h so that $M_h = \min_{1 \le y \le m} M_y$. If $\max\{A + a_{s(i)}; M_h\} + b_{s(i)} \le d_{s(i)}$, schedule job $s(i)$ on machine h, represented by the right concatenation of job $s(i)$ as $\sigma_h = (\sigma_h, s(i))$, and set $C_{1,s(i)} = A + a_{s(i)}$, $M_h = \max\{C_{1,s(i)}; M_h\} + b_{s(i)}$, $A = C_{1,s(i)}$ and $e = e + 1$; otherwise, use decision rule T to find the job to be made tardy and, if needed, update A, σ_y, and M_y for $y = 1, \ldots, m$. Enter step 3.

Step 3. If $i < n$, then set $i = i + 1$, and return to Step 2. Otherwise, STOP as the schedule σ with jobs in σ_h assigned to machine h is the heuristic solution of the problem with $n - e$ tardy jobs.

1.2.1 Creating the List Schedule S

For finding schedule S, two types of lists are suggested by Gupta and Tunc[7] . In the first list type (called algorithm D), a set K of jobs are arranged in the earliest-due-date (EDD) order while in the second list type (called Algorithm D'), jobs are arranged in the *earliest-modified-due-date* order (EMDD) where the modified due date for each job i, $d'_i = d_i - b_i$.

In the application of algorithm D (or D') above, there is a choice of the set of jobs K. In order to solve the $F2|P(1,m)|\sum U_i$ problem, we can let $K = N$, the set of all jobs. However, we can reduce the number of jobs in set K by ignoring all jobs that were found tardy, while minimizing the total number of tardy jobs for a single machine problem,

where the processing time of job $i \in N$ is $p_i = a_i$ and its due date is $d'_i = d_i - b_i$. Such a reduction in set K is justified since these tardy jobs are likely to be tardy in an optimal schedule. Thus, we can let $K = N'$ where N' is the set of jobs completed on or before their modified due dates in the application of Moore's[12] algorithm to solve the above single-machine problem.

1.2.2 Various Types of Decision Rule T

We now describe ways a list (schedule S) of jobs can be assigned to various machines at both stages. Our discussion also includes decision rules to identify jobs. The first procedure (called algorithm L) assigns a job to the earliest available machine at the second stage and counts the total number of tardy jobs. However, the effectiveness of algorithm L can be improved by deleting the first tardy job from the list since doing so will free up machine time that can be used to complete the subsequent jobs earlier. Such a modified list scheduling algorithm is termed algorithm L'.

The effectiveness of the above list-processing algorithms can be improved by using additional information about the scheduled jobs. The first algorithm (called algorithm M to denote *minimum makespan scheduling*) works as follows: whenever a first tardy job is found, we find a job which, if removed from the list, will make the first tardy job early and will minimize the makespan of all the remaining jobs. If such a job is found, we make that job tardy, otherwise the first job to be completed after its due date is tardy.

The computational effort required to solve the problem by algorithm M can be decreased by reducing the need to find completion times of a large number of partial sequences. This is done by using the concept of a *critical path* and *critical job* yielding the makespan value. In the two-stage-hybrid-flowshop problem, we implement this idea (called algorithm C representing *critical path scheduling*) by identifying the critical path shifts from stage 1 to machine h at stage 2 that defines the makespan. Then, as in Moore's algorithm, a job with maximum processing time on the critical path is made tardy so long as it makes the first tardy job early. If the first tardy job is not early, then that job is made tardy.

The computational effort to solve a given problem can be decreased further by creating an artificial single-machine problem where the processing time of each job $i \in N$ is given by: $p_i = a_i + b_i$. The job-assignment procedure (called algorithm P to denote

largest processing-time scheduling) is described as follows: whenever the first tardy job is encountered among all the assigned jobs (including the first tardy job), a job with largest processing time (p_i) is made tardy.

Each heuristic method employs an appropriate selection of algorithms from N, N', D, D', L, L', M, C, and P. In heuristic NDC, for example, Algorithm D is applied first to obtain a sequence π containing all jobs in set N, which is input into Algorithm C to obtain a heuristic solution. Since each possible combination of list creation algorithms N, N', D, and D' is used with each of the five heuristics L, L', M, C, and P to assign jobs to machines, we get a total of twenty heuristic algorithms.

The computational complexity of each of the proposed heuristic algorithms is $O(n^2)$ since the initial list is prepared in $O(n \log n)$ time and the generation of the final schedule takes $O(n^2)$ computational time.

2 Selecting an Algorithm

This section shows that the problem of selecting a scheduling algorithm can be viewed as a prediction problem and suggests the use of neural networks for solving it. Subsequently, the specific neural-network system used in our computational work is described.

2.1 Heuristic Selection Problem

In order to use neural networks for selecting a scheduling algorithm, assume that a total of r heuristic algorithms, represented by a set R, are under consideration. Let $n_t(j)$ represent the total number of tardy jobs of the schedule obtained by using algorithm $j \in R$. Define the following:

$$\alpha = \max_{j \in R}\{n_t(j)\} \tag{3}$$

and for each $j \in R$, define

$$\beta_j = n_t(j)/\alpha \tag{4}$$

Clearly, the smaller the value of β_j calculated by using (3) and (4), the smaller the total number of tardy jobs of a schedule obtained by heuristic j. Therefore, we can base the selection of a heuristic algorithm on the predicted values of the β vector and define the heuristic selection problem as follows:

- *Given the processing times and due dates of a scheduling problem with n jobs, and m machines at the second stage, we wish to predict the values of β_j for $j \in R$ and select that algorithm j to solve a specific scheduling problem for which the predicted β_j value is minimum.*

Gupta and Tunc[8] used the above definition of the heuristic selection problem and designed a neural-network approach to solve it. However, their approach required the design and training of an individual neural network for each problem size since the input to their network included the processing times of jobs at both stages. For realistic application of neural networks to practical scheduling problems, this type of approach is rather difficult to use. Therefore, we need to find some critical problem characteristics to represent a scheduling problem.

In analyzing various heuristic algorithms using ANOVA, Gupta and Tunc[7] found that the number of jobs, number of machines at the second stage, the job processing times at both stages, and the job due dates affect a heuristic's effectiveness in finding an optimal schedule. These findings show that factors based on the above problem characteristics may be sufficient to predict the best heuristic algorithm to be used to solve a specific problem instance. For these reasons, we propose the use of the average and standard deviation of the processing times at each stage, the number of jobs, the number of machines at the second stage, and the average and standard deviation of the due dates as the critical characteristics of a scheduling problem. Once the heuristic algorithm is selected, actual problem data are used to find an optimal or near-optimal schedule for the given problem instance. Thus, in this paper, we define the heuristic selection problem as follows:

- *Given the averages and standard deviations of the processing times at both stages and the average and standard deviations of the due dates of all jobs of a scheduling problem with n jobs, and m machines at the second stage, we wish to predict the values of β_j for $j \in R$ and select that algorithm j to solve a specific scheduling problem for which the predicted β_j value is minimum.*

This selection may not be perfect and we may not always pick the best heuristic algorithm. However, in the absence of any other polynomially bounded mechanism to predict the optimal number of tardy jobs, this would appear to be the best option. Thus, for the

$F2|P(1,m)|\sum U_i$ problem, the actual values of $\beta = (\beta_1, \ldots, \beta_r)$ can be used to deduce which algorithm results in the lowest number of tardy jobs and hence is the most appropriate heuristic to solve a given instance of the $F2|P(1,m)|\sum U_i$ problem. If we could predict this vector prior to the solution of the problem, we could then select the best algorithm to solve this problem. In this paper, we propose the use of neural networks to do this.

2.2 Neural Network Configuration

The neural-network approach to solve combinatorial optimization problems, including scheduling problems is reviewed by Smith[16]. Given the large size of the corresponding combinatorial optimization problem and the limited types of the objective functions that can be optimized using neural networks, a direct solution of our scheduling problem by neural networks is not practical.

The artificial neural network (NN) approach to selecting a scheduling algorithm transforms a scheduling-algorithm-selection problem to one of predicting the rankings of a given number of heuristic algorithms. For this purpose, we developed two neural-network systems described below.

2.2.1 One-Tier Neural Network

The input layer of this neural network (see Figure 1a), included as inputs the number of jobs (n), the number of machines (m) at the second stage, the averages and standard deviations of the processing times for stages 1 (a, σ_a) and 2 (b, σ_b), and of the due dates (d, σ_d). It has been shown in previous NN research[5,9] that as few as one hidden layer is sufficient to approximate any unknown function to any desired degree of accuracy, as long as there are sufficiently many hidden nodes. The NN configuration uses three layers. The choice of the number of nodes in the hidden layer is made through experimentation. In our preliminary computational experiments, we found that the use of four, eight, and ten nodes in the hidden layer performed worse than did six nodes. In our computational experiments, therefore, we used six nodes in the hidden layer, as this NN architecture provided the best overall results.

To clarify the inputs used in the one-tier network, a sample observation corresponding to a ten-job problem with five machines at the second stage is shown in Table I. The

values were normalized by dividing each input by the largest value in the respective category giving a range of (0, 1) for all categories.

Table I. Normalized Input Values for an Observation

No. of jobs	No. of machines	Stage 1		Stage 2		Due Date	
		Mean	Stdev	Mean	Stdev	Mean	Stdev
0.1	0.5	0.64	0.80	0.79	0.64	0.03	0.03

The *desired* or *target output vector* for a problem instance consists of the β_j values obtained by solving that problem by each of the 20 heuristic algorithms. Based on these solutions for each problem instance, the algorithms were ranked by their relative number of tardy jobs given by (3) and (4), with smaller value of tardy jobs recognized as being superior. Thus, rankings of these β_j values represent the rankings of the corresponding algorithms.

The output layer of the one-tier network consists of 20 nodes, where node j represents the β_j value (number of tardy jobs) for algorithm j given by (3) and (4). For the output node j of the first tier, let r_j be its integer predicted rank of the β_j, such that $r_j \leq r_{j+1}$ implies that $\beta_j \leq \beta_{j+1}$ and $1 \leq j \leq 20$. For the input observation in Table I, the NN output and predicted rankings are shown in Table II.

Table II. Output from One-tier NN

Node j	1	2	3	4	5	6	7	8	9	10
β_j	0.091	0.084	0.104	0.103	0.107	0.048	0.042	0.030	0.105	0.098
r_j	7	6	10	9	14	4	3	1	12	8
Node j	11	12	13	14	15	16	17	18	19	20
β_j	0.080	0.128	0.317	0.156	0.780	0.039	0.104	0.243	0.106	0.785
r_j	5	15	18	16	19	2	11	17	13	20

—> Insert Figures 1a and 1b about here <—

2.2.2 Two-Tier Neural Network

Empirical results from the one-tier neural network, although very promising, showed that while the neural network may not perform very well in terms of picking the best algorithm, it performed extremely well when asked to pick the top five algorithms. To

improve the NN's performance in selecting the best heuristic algorithm, we developed a two-tier neural network where the second-tier network is a three-layer neural network with 20 input nodes, six hidden nodes and 20 output nodes (see Figure 1b).

The input vector to the second-tier neural network is $\delta = (\delta_1, \delta_2, \ldots, \delta_R)$ where $\delta_j = 1$ if $r_j \leq 5$ and 0 otherwise, and r_j is the predicted integer rank of algorithm j obtained from the one-tier neural network. For the sample observation of Table I, the one-tier network ranked algorithms 6, 7, 8, 11, and 16 to be the best five algorithms (see Table II). Therefore, as shown in Table III, the input vector δ for these heuristics has value 1 while other values are zero.

Table III. Input and Output from the Two-Tier NN

Node j	1	2	3	4	5	6	7	8	9	10
δ_j	0	0	0	0	0	1	1	1	0	0
γ_j	0.311	0.289	0.350	0.260	0.174	0.746	0.562	0.803	0.409	0.356
Rank	15	16	13	17	18	4	6	2	9	12
Node j	11	12	13	14	15	16	17	18	19	20
δ_j	1	0	0	0	0	1	0	0	0	0
γ_j	0.481	0.399	0.357	0.349	0.110	0.966	0.664	0.768	0.455	0.142
Rank	7	10	11	14	20	1	5	3	8	19

Given this information, the two-tier network attempted to map the relationship between the five best algorithms selected with the one-tier network to the algorithms that were actually the best for a given problem. This is very different from the one-tier network, which simply tried to rank all the algorithms. With the five best solutions from the one-tier network, the two-tier network could still choose an algorithm that was not included in the original five best algorithms. Also, since the input and output data were transformed, much of the noise inherent in the one-tier network was eliminated allowing the two-tier network to focus on the relationship between the five candidate algorithms and the actual best algorithm.

For each problem instance, the *target output vector* for the two-tier network was generated by assigning a 1 for each algorithm that actually found the minimum number of tardy jobs, and 0 for all other algorithms. For example, if only algorithms 2 and 5 found zero tardy jobs for a problem instance, then its target output vector for the two-tier network would be $(0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$. Thus, the value of

the output node $j \leq R$ of the second tier shows the predicted ranked value of heuristic j, denoted by γ_j. The larger the value of γ_j, the better is heuristic j for solving a given problem. Therefore, the highest value of γ_j in the output vector of the two-tier neural network identifies the best heuristic algorithm for the given problem. Thus, for the second tier NN, we define the heuristic selection problem as follows:

- *Given the binary ranking of each heuristic $j \in R$ as δ_j as the input, we wish to predict the values of γ_j for $j \in R$ and select that algorithm j to solve a specific scheduling problem for which the predicted γ_j value is maximum.*

We define the integer ranking of heuristic j as p_j such that $1 \leq p_j \leq R$ where $\gamma_j \geq \gamma_{j+1}$ implies that $p_j \geq p_{j+1}$. As shown in Table III, for the sample observation in Table I, the best algorithm according to the two-tier network will be algorithm 16. However, had we used the one-tier network, we would have selected heuristic 8 as the best algorithm.

The two-tier NN proposed above is different than a simple NN with two hidden layers in at least two ways: first, it changes the input vector for the second tier, which is not possible in a NN with two hidden layers; and second, it changes the selection criteria from minimization in the first tier to maximization in the second tier. Thus, in effect, we have created a *multi-tier neural network architecture* with decision criteria being used between various tiers of the NN.

The neural network has to be trained before it can be used for solving problems. In this training, we identify the optimal weights applied to each input-output link of the networks shown in Figures 1a and 1b. The majority of NN research has relied upon a gradient algorithm, typically a variation of backpropagation[14] to search for optimum weights in the NN. When applied to the training of artificial neural networks, backpropagation often results in inconsistent and unpredictable performance. Since the genetic algorithm has been shown to outperform backpropagation in the training of NNs[6,15], we use the genetic algorithm to train our NNs.

2.3 The Genetic Algorithm

A formal description of the algorithm is provided by Dorsey and Mayer[3] and briefly shown in Figure 2. In general, the algorithm begins by randomly selecting a population of possible solutions. Each potential solution is a set of weights for the NN. This

population is the first generation from which the genetic algorithm will begin its search for an optimal solution. For several difficult non-linear functions, Dorsey and Mayer[3] experimented with different population sizes varying from 10 through 50 and found that a population size of 20 was sufficient for solving these problems. Although larger population sizes would allow for a wider search over the weight space, it was found that the increased computational effort was unnecessary for solving the problem. Thus, our genetic algorithm used a population size of 20. In our genetic-algorithm-trained NN, 20 sets of weights are evaluated in each generation. The genetic algorithm searches the weight space from one set of weights to another set, searching in many directions simultaneously, unlike more traditional search methods. This enhances the probability of finding the global optimum.

—> Insert Figure 2 about here <—

For a candidate solution (member of the current population) s, let t_{xj} and z_{xj}^s be the *target* (found by using algorithm j) and *predicted* (from NN) output values for output node $j = 1, \ldots, R$ and problem instance $x = 1, \ldots, X$. Then, the sum of squared errors for the candidate solution s, $SSE(s)$ is computed as follows:

$$SSE(s) = \sum_{x=1}^{X} \sum_{j=1}^{R} (t_{xj} - z_{xj}^s)^2 \tag{5}$$

In the genetic algorithm for training the NN, we seek a candidate solution that minimizes the sum of squared errors given by (5). Therefore, the sum of squared errors for each member of the current population (candidate solution) is computed using (5) and is used as its *fitness function*. A probability is assigned to each solution based on the value of the fitness function. For example, using the sum of the squared errors, the solutions that result in the smallest sum of squared errors are assigned the highest probabilities. This completes the first generation. The second generation begins by selecting a new population from the former, based on their assigned probabilities. Twenty solutions are chosen with replacement so that good solutions are likely to be well represented in the new population and poor solutions are unlikely to be drawn. This process is known as *reproduction*. The algorithm generally parallels the process of natural selection; hence its

name. As in the saying *survival of the fittest*, the most favorable traits in optimizing the objective function will reproduce and thrive in future generations, while weaker traits die out.

This new population of solutions (all of which existed in the prior generation) is next randomly grouped into pairs of solutions, and a subset of the weights from each solution are switched with its paired solution (crossover). This creates two new solutions for each pair, each with some parameters (weights) from their respective parents. Finally, each solution has a small probability that any of its weights may be replaced with a value uniformly selected from the parameter space (mutation). This resulting set of solutions becomes the new population or next generation, and the process repeats. This process continues until the initial population evolves to a generation that will best solve the optimization problem, ideally the global solution.

3 Experimental Results

This section describes the computational tests used to evaluate the effectiveness and efficiency of the neural-network approach to select the best heuristic to solve a particular problem instance.

3.1 Test Problems

Two classes of test problems were generated. In each class, second-stage processing times are random integers generated from the uniform distribution $(1, 100)$. Problem "hardness"is likely to depend on whether there is a balance between average workloads on the first-stage machines and the total workload at the second stage. Thus, two data sets were generated. The first-stage processing times for each data set are random integers generated from the following uniform distributions:

- First data set (called the *non-dominant data set*): $(1, 100)$.
- Second data set (called the *dominant data set*): $(1, 100/m)$.

Problem "hardness"is also likely to depend on the range of the due dates. Therefore, following Gupta and Tunc[7], due dates were generated by using an estimated value of the makespan, E, and two parameters $\lambda = \{0.2, 0.4, 0.6, 0.8\}$ and $\omega = \{0.4, 0.6, 0.8, 1.0\}$ with

$\omega > \lambda$. Thus, the due dates are random integers generated from a uniform distribution in the range $(E\lambda, E\omega)$ where the estimated makespan E is given by

$$E = \Big\{ \sum_{i=1}^{n} a_i + (\sum_{i=1}^{n} b_i/m) + (n-1)\max\Big[\sum_{i=1}^{n} a_i; \sum_{i=1}^{n} b_i/m\Big]\Big\}/n. \tag{6}$$

Further, in order to ensure that all n jobs in any specific problem were not tardy, the due date of at least one job i, $d_i \geq a_i + b_i$. Depending upon the values of the two parameters λ and ω, in relation to the estimated makespan E, due dates may be quite tight or loose. Thus, the ten classes of due dates generated by using (6) and various values of λ and ω would contain problems of varying difficulty in finding an optimal schedule.

The number of jobs, n varied from ten through 100 in increments of ten jobs. The number of identical parallel machines at the second stage was $m = 2, 3$, or 5. For each value of n and each combination of λ and ω, ten problems were generated giving a total of 100 problems of each size (number of jobs and number of machines at the second stage) for each problem class.

3.2 Training the Neural Networks

The constructive heuristic algorithms described in section 2 were applied to solve each test problem. For each problem and each heuristic algorithm, the number of machines, number of jobs, the average and standard deviation of processing times for stage-1 and stage-2, the average and standard deviations of the due dates, and the total number of tardy jobs were normalized to the interval (0,1) by dividing them by their respective maximum value among all 6,000 problems.

The normalized parameters and the outputs for all odd-numbered problems for each problem size, each data set, and each due-date class comprised the training data, a total of 3,000 problems. The NNs were trained using the genetic algorithm with the suggested parameter settings from Dorsey, Johnson, and Mayer[4] described in Figure 2. The training of the one-tier and two-tier networks consisted of 100 generations.

3.3 Effectiveness of the NN Approach

After training the NNs, the normalized parameters and the outputs for the even-numbered problems for each problem size, each data set, and each due-date class (a total of 3,000 problems) were used to evaluate the effectiveness of the neural-network approach to select a scheduling heuristic. The effectiveness of the proposed NN approach was evaluated by finding neural network's ability to (1) predict the best heuristic algorithm to use for a given problem instance, and (2) reduce the average percentage of tardy jobs produced.

3.3.1 Effectiveness in Predicting the Best Heuristic

For each problem size and data set, the number of times the NN accurately predicted the correct heuristic algorithm to be used was calculated, separately for the *in-sample* (or training data) and *out-of-sample* (or the test observations).

Table IV. Percentage of Accurate Prediction by One-Tier Network

Problem Data Set	No. of Job	In-sample			Out-of-sample		
		2 Mach.	3 Mach.	5 Mach.	2 Mach.	3 Mach.	5 Mach.
Non-dominant	10	66	66	86	60	74	92
	20	82	94	96	76	94	98
	30	92	96	98	92	94	98
	40	96	100	100	98	96	100
	50	92	100	92	92	100	100
	60	88	94	98	96	94	100
	70	98	96	96	94	100	98
	80	94	88	96	98	96	94
	90	94	94	92	92	94	98
	100	94	90	80	96	92	92
		In-sample Average = 91.60			Out-sample Average = 93.27		
Dominant	10	58	70	98	64	90	90
	20	66	70	84	72	60	66
	30	68	60	58	70	52	60
	40	80	72	48	82	70	40
	50	98	86	94	96	88	94
	60	94	98	92	100	98	92
	70	94	94	98	96	96	92
	80	96	96	100	100	100	96
	90	96	88	100	98	94	96
	100	98	100	100	94	98	96
		In-sample Average = 85.13			Out-sample Average = 84.67		
Overall		In-sample Average = 88.36			Out-sample Average = 88.97		

For the one-tier NN, Table IV depicts the percentage of problems for which the NN correctly predicted the best heuristic algorithm. These results show that the proposed one-tier network accurately predicts the best heuristic algorithm to be used to solve

a specific problem in 89% of the cases. However, its ability to predict accurately the best heuristic algorithm to be used to solve a specific problem in the dominant data set decreases to 85% of the cases.

For the two-tier NN, Table V depicts the percentage of problems for which the NN correctly predicted the best heuristic algorithm. These results show that the effectiveness of the proposed two-tier network to predict correctly the best heuristic algorithm increases to 94% from 89% for the one-tier NNs. For problems in the dominant data set, the two-tier NNs correctly predicted the best heuristic in 90% of the cases, as compared to 85% for the one-tier NNs. Tables IV and V also show similar performance between the in-sample and out-of-sample results, indicating good generalization of the learning. This is perhaps due to the choice of the genetic algorithm for determining the neural network weights, since the genetic algorithm does not suffer from the learning-memorization tendencies of the backpropagation algorithm[15].

Table V. Percentage of Accurate Prediction by Two-Tier Network

Problem Data Set	No. of Job	In-sample			Out-of-sample		
		2 Mach.	3 Mach.	5 Mach.	2 Mach.	3 Mach.	5 Mach.
Non-dominant	10	96	94	94	90	96	82
	20	96	98	100	96	98	96
	30	96	96	96	94	100	100
	40	96	100	100	100	100	100
	50	96	98	96	96	100	100
	60	96	96	98	100	98	100
	70	98	100	98	98	100	100
	80	98	98	100	100	96	100
	90	92	98	96	96	100	100
	100	100	98	100	100	100	100
		In-sample Average = 97.27			Out-sample Average = 97.87		
Dominant	10	76	72	74	70	68	66
	20	92	72	62	92	76	54
	30	88	90	64	92	92	64
	40	90	98	92	96	98	98
	50	98	96	98	96	98	100
	60	94	98	96	100	98	96
	70	94	94	98	96	96	94
	80	96	96	100	100	100	96
	90	96	88	100	98	94	96
	100	98	100	100	94	98	94
		In-sample Average = 90.33			Out-sample Average = 90.33		
Overall		In-sample Average = 93.80			Out-sample Average = 94.10		

The results in Table V show that the proposed two-tier neural-network approach was successful in predicting the best heuristic algorithm for well over 94% of the problem instances. Further, the ability of the neural-network approach to predict the best heuristic

algorithm increases with an increase in the number of jobs and identical parallel machines at the second stage.

The effectiveness of the neural-network approach in selecting a heuristic algorithm for the dominant-data-set problems is a little less than that for the non-dominant-data-set problems. This could be a result of the rather large differences between the averages and standard deviations of the processing times at the two stages for problems in the dominant data set. This effect could also be the result of insufficient training given to the NNs as the training sessions used for problems in the dominant and non-dominant data sets were identical.

A quick analysis of additional detailed results not reported in Tables IV and V showed that about 18% of the test problems (mostly in the dominant data set) required only one of the twenty algorithms to obtain the best solution. Among the test problems for which only one of the twenty algorithms produced the best solution, the neural-network approach found the best heuristic algorithm 92% of the time. Considering that limited problem characteristics were used as inputs to the proposed neural network, these computational results show that the proposed neural-network approach is quite effective in predicting the best heuristic algorithm to solve the two-stage-hybrid-flowshop-scheduling problem.

3.3.2 Effectiveness in Minimizing Percent Tardy Jobs

A heuristic may be very good at minimizing the tardy jobs most of the time. However, when it is not the best heuristic, the percentage of tardy jobs it produces could be very bad. Therefore, we also evaluated the effectiveness of the proposed neural-network approach in reducing the average number of tardy jobs for any problem. For this purpose, the average and standard deviation of the percentage of tardy jobs in a problem by each of the 20 algorithms were calculated. In addition, the average and standard deviation of the percentage of tardy jobs in a problem obtained by using the algorithm predicted to be best by the neural-network approach were calculated.

From the results depicted in Table VI, it follows that there is no single heuristic that performs best on both measures of performance used. Compared to an algorithm that found the best solution most of the time (algorithm 18), the results in Table VI show that the neural-network approach reduces the average percentage of tardy jobs by 4.57%.

This percentage increases to 7.23% for problems in the dominant data set. The average percentage of tardy jobs produced by using the neural-network approach is only 0.32% higher than the best possible average percentage for all 20 heuristics. Interestingly, for problems in the dominant data set, using the proposed neural-network approach results in the lowest average percentage of tardy jobs.

3.3.3 Effectiveness in Selecting Non-dominant Algorithms

When considering the primary goal of the one- and two-tier networks as that of selecting the best heuristic algorithm to be used to solve a given problem, algorithm 18 in Table VI is a dominant heuristic. Training and using the proposed two-tier neural network by excluding algorithm 18 showed that the proposed two-tier network correctly predicted the best heuristic algorithm for 82% of the problem instances without any changes in the average percentage of tardy jobs reported in Table VI.

Table VI. Comparative Performance of Heuristics and Neural-Network Approach

Heuristic	Heuristic		Overall		Non- dominant		Dominant	
#	Name	% Best	% Tardy	Std	% Tardy	Std	% Tardy	Std
1	NDP	39.40	20.65	16.54	16.24	13.53	25.05	18.03
2	NDC	38.43	21.05	16.79	16.26	13.54	25.84	18.30
3	NDM	40.43	20.61	16.63	16.23	13.51	24.99	18.23
4	NDL'	37.83	21.13	16.80	16.28	13.56	25.98	18.26
5	NDL	35.60	26.83	21.54	16.55	13.94	37.12	22.85
6	$ND'P$	69.73	19.09	16.01	15.30	13.01	22.88	17.75
7	$ND'C$	65.00	20.28	17.11	15.31	13.01	25.26	19.16
8	$ND'M$	71.53	18.93	15.92	15.29	12.99	22.56	17.66
9	$ND'L'$	60.47	21.46	17.90	15.33	13.05	27.58	19.90
10	$ND'L$	58.62	27.20	23.57	15.43	13.27	38.97	25.66
11	$N'DP$	28.23	21.65	17.40	17.63	14.94	25.68	18.70
12	$N'DC$	18.38	23.47	17.90	20.26	16.51	26.69	18.65
13	$N'DM$	51.28	21.11	20.19	16.06	14.39	26.16	23.61
14	$N'DL'$	16.88	24.07	18.24	20.86	16.98	27.28	18.87
15	$N'DL$	4.38	44.06	28.35	38.83	27.75	49.29	27.99
16	$N'D'P$	48.60	19.88	16.51	16.86	14.35	22.90	17.92
17	$N'D'C$	30.37	22.74	17.99	18.86	15.49	26.61	19.43
18	$N'D'M$	93.53	19.90	20.99	15.54	14.08	24.26	25.39
19	$N'D'L'$	24.82	24.33	18.86	19.11	15.74	29.54	20.22
20	$N'D'L$	6.98	46.08	30.03	38.79	28.13	53.37	30.11
NN results		94.10	18.99	18.38	15.48	13.73	22.51	21.50
Best possible for all heuristics			18.93	15.92	15.29	12.99	22.56	17.66
NN % improv over heuristic # 18			4.57	12.43	0.40	2.46	7.23	15.32

In order to test further the effectiveness of our proposed neural networks, we successively deleted all those heuristic algorithms that could be considered dominant. Similar to the results in sections 3.3.1 and 3.3.2, Tables VII and VIII report the results obtained for the reduced model consisting of eight remaining (non-dominant) heuristic algorithms. While the effectiveness of the neural-network approach decreases somewhat, the proposed two-tier neural network did predict the best heuristic algorithm in 73.45% of the cases. If heuristic algorithm 11 (best of the eight heuristics) was used, the best solution will be generated only 55% of the time. Further, compared to algorithm 11, the use of the proposed neural-network approach reduced the average percentage of tardy jobs by 2.76%. These results show that the proposed neural-network approach in selecting the best heuristic algorithm is quite effective even when there is no dominant heuristic.

Table VII. Percentage of Accurate Prediction by Two-Tier Network for the Reduced Model

Problem Data Set	No. of Job	In-sample			Out-of-sample		
		2 Mach.	3 Mach.	5 Mach.	2 Mach.	3 Mach.	5 Mach.
Non-dominant	10	66	78	82	66	72	68
	20	88	78	82	62	82	76
	30	84	88	84	68	82	84
	40	84	94	92	90	86	94
	50	88	94	96	80	88	86
	60	88	88	92	90	94	94
	70	98	92	98	86	90	94
	80	82	88	94	82	92	88
	90	90	90	94	84	90	90
	100	82	84	94	86	92	86
		In-sample Average = 87.73			Out-sample Average = 84.07		
Dominant	10	78	50	56	62	48	56
	20	62	52	32	54	48	22
	30	74	50	40	70	52	40
	40	74	58	36	74	62	42
	50	82	64	90	68	70	90
	60	84	58	46	78	58	46
	70	80	60	34	68	68	50
	80	82	62	38	78	78	52
	90	80	78	42	84	66	44
	100	86	72	44	84	60	44
		In-sample Average = 61.47			Out-sample Average = 60.53		
Overall		In-sample Average = 74.60			Out-sample Average = 72.30		

3.4 Efficiency of the NN Approach

As stated earlier, the computational complexity of each heuristic algorithm is $O(n^2)$ where n is the total number of jobs. After training, the proposed neural-network approach can identify the best heuristic algorithm to be used to solve a given problem instance in

constant time. Therefore, the computational time required to find the best heuristic algorithm by the proposed neural-network approach and then solve a problem instance using this selected heuristic algorithm will be about 1/20 of that required to solve a problem instance using all 20 heuristics.

Table VIII. Comparative Performance of Heuristics and Neural Network-Approach For the Reduced model

Heuristic #	Heuristic Name	% Best	Overall		Non- dominant		Dominant	
			% Tardy	Std	% Tardy	Std	% Tardy	Std
5	NDL	47.95	26.83	21.54	16.55	13.94	37.12	22.85
11	$N'DP$	55.17	21.65	17.40	17.63	14.94	25.68	18.70
12	$N'DC$	30.62	23.47	17.90	20.26	16.51	26.69	18.65
14	$N'DL'$	25.48	24.07	18.24	20.86	16.98	27.28	18.87
15	$N'DL$	4.55	44.06	28.35	38.83	27.75	49.29	27.99
17	$N'D'C$	45.02	22.74	17.99	18.86	15.49	26.61	19.43
19	$N'D'L'$	31.85	24.33	18.86	19.11	15.74	29.54	20.22
20	$N'D'L$	7.17	46.08	30.03	38.79	28.13	53.37	30.11
NN results		73.45	21.06	17.10	16.46	13.90	25.65	18.65
Best possible for all heuristics			21.06	17.10	16.46	13.90	25.65	18.65
NN % improv over heuristic # 11			2.76	1.74	6.60	6.97	0.12	0.08

To test the above assertion empirically, the efficiency of the neural-network approach in selecting the best heuristic algorithm was determined by the CPU time required to select and use the best heuristic algorithm using the two-tier neural network as opposed to using all 20 heuristic algorithms to solve that specific problem instance. The initial training of the NNs required a total of 5,785 seconds on an IBM-compatible 200 MHz PC running under Windows 98.

Figure 3 depicts the CPU time required to solve various problem instances using all 20 heuristics (20 Alg.), the CPU time required to select an algorithm through the proposed two-tier neural-network approach (NN), and the CPU time required to select and use the best heuristic algorithm using the two-tier neural-network approach (NN+Alg.).

—> Insert Figure 3 about here <—

As expected, the CPU time required to select the best heuristic algorithm using the proposed two-tier neural network is independent of the problem size. Further, the difference between the CPU time required to select and use the best heuristic algorithm

using the two-tier neural network and the CPU time required for using all 20 heuristic algorithms increases quadratically with the increase in the number of jobs. Based on these computational time requirements, we conclude that the proposed two-tier neural network is quite efficient in selecting the best heuristic for the scheduling problem.

4 Conclusions

This paper has considered the problem of selecting an appropriate heuristic algorithm for solving a scheduling problem. Using the example of a two-stage hybrid flowshop, where the first stage consists of only one machine and stage two consists of multiple identical machines, the heuristic-selection-problem is shown to be a prediction problem that can be effectively solved using a neural-network system. Computational results show that the proposed neural-network approach correctly predicted the best heuristic algorithm to solve a given problem in more than 94% of the problem instances. Further, the effectiveness of the proposed approach was enhanced with the increase in the number of jobs and identical parallel machines at the second stage. In addition, the effectiveness of the proposed neural-network approach to find the best heuristic algorithm did not significantly decrease, even when only one of the twenty algorithms produced the best solution.

The architecture of our proposed neural networks is independent of the size of the problem and has constant input-output lengths. The input to the proposed neural networks consisted only of the averages and standard deviations of the processing times and of the due dates along with the number of jobs and the number of machines. Given these apparent limitations of the input requirements of the neural network, the results are quite encouraging. Thus, with a given set of problem characteristics, a neural-network approach can be used to select a scheduling heuristic for large-size problems. Hence, the use of a neural-network system is a workable approach for the selection of an appropriate heuristic algorithm for NP-hard optimization problems found in industry.

Several fruitful directions for future research can be identified. First, the exact configuration of the neural-network system that promises the best results in all cases should be investigated. This may involve the determination of the exact number of nodes in the hidden layer. Second, further research in the specification of the problem parameters

in the neural-network approach may improve the effectiveness and applicability of the neural-network approach to the heuristic-algorithm-selection problem. Third, extension of the approach outlined here to general job-shop scheduling problems will be both interesting and useful. Fourth, development of a neural-network approach for multi-criteria scheduling problems provides an interesting and useful area for future research. Finally, development of multi-tier neural networks for the selection and prediction problems would enlarge the applicability domain of the neural-network approaches to industrial problems.

Acknowledgements: The authors express deep appreciation to Dr. Kate A. Smith of Monash University, Australia, the Associate Editor and three anonymous referees for their constructive comments and suggestions that improved the presentation of this paper.

References

1. D. ALPSAN, , M. TOWNSEY, O. OLDAMAR, A. C. TSOI, and D. N. GHISTA, 1995. Efficiency of Modified Backpropagation and Optimization Methods on a Real-World Medical Problem, *Neural Networks 8*, 945-962.

2. L. I. BURKE and J. P. IGNIZIO, 1992. Neural Networks and Operations Research: An Overview, *Computers and Operations Research 19*, 179-189.

3. R. E. DORSEY and W. J. MAYER, 1995. Genetic Algorithms for Estimation Problems with Multiple Optima, Non-Differentiability, and Other Irregular Features, *Journal of Business and Economic Statistics 13*, 53-66.

4. R. E. DORSEY, J. D. JOHNSON and W. J. MAYER, 1992. The Genetic Adaptive Neural Network Training (GANNT) for Generic Feedforward Artificial Neural Systems, *Working Paper*, School of Business Administration, University of Mississippi, University, MS.

5. K.-I. FUNAHASHI, 1989. On the Approximate Realization of Continuous Mappings by Neural Networks, *Neural Networks 2*, 183-192.

6. J. N. D. GUPTA and R. S. SEXTON, 1999. Comparing Backpropagation With a Genetic Algorithm for Neural Network Training, *OMEGA 27*, 679-684.

7. J. N. D. GUPTA and E.A. TUNC, 1998. Minimizing Tardy Jobs in a Two-stage Hybrid Flowshop, *International Journal of Production Research 36*, 2397-2417.

8. J. N. D. GUPTA and E. A. TUNC, 1997. Neural Networks Approach to Select Scheduling Heuristics for a Two-Stage Hybrid Flowshop, *International Journal of Management and Systems 13*, 283-298.

9. K. HORNIK, M. STINCHCOMBE, AND H. WHITE, 1989. Multilayer Feedforward Networks are Universal Approximators, *Neural Networks 2*, 359-366.

10. E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN and D. B. SHMOYS, 1993. Sequencing and Scheduling: Algorithms and Complexity, in *Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin (eds.), North Holland, Amsterdam, the Netherlands, 445-522.

11. J. K. LENSTRA, A. H. G., RINNOOY KAN, and P. BRUCKER, 1977. Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics 1*, 343-362.

12. J. M. MOORE, 1968. An n Job, One-Machine Sequencing Algorithm for Minimizing the Number of Late Jobs, *Management Science 15*, 102-109.

13. K. E. NYGARD, P. JUELL and N. KABADA, 1990. Neural Networks for Selecting Vehicle Routing Heuristics, *ORSA Journal on Computing 4*, 485-493.

14. D. E. RUMELHART, G. G. HINTON and R. J. WILLIAMS, 1986. Learning Internal Representations by Error Propagation, in *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, D. E. Rumelhart (ed.), MIT Press, Cambridge, MA, 318-362.

15. R. S. SEXTON, R. E. DORSEY, and J. D. JOHNSON, 1998. Toward Global Optimization of Neural Networks, *Decision Support Systems 22*, 171-185.

16. K. A. SMITH, 1999. Neural Network for Combinatorial Optimization: A Review of More Than a Decade of Research, *INFORMS Journal on Computing 11*, 15-34.

17. D. TUZUN, M. A. MAGENT, and L. I. BURKE, 1997. Selection of Vehicle Routing Heuristics Using Heuristic Networks, *International Transactions on Operations Research 4*, 211-221.