J Intell Manuf (2008) 19:191–201
DOI 10.1007/s10845-008-0073-9

A neural network job-shop scheduler

Gary R. Weckman · Chandrasekhar V. Ganduri · David A. Koonce

Received: 1 May 2006 / Accepted: 1 July 2007 / Published online: 20 January 2008
© Springer Science+Business Media, LLC 2008

Abstract This paper focuses on the development of a neural network (NN) scheduler for scheduling job-shops. In this hybrid intelligent system, genetic algorithms (GA) are used to generate optimal schedules to a known benchmark problem. In each optimal solution, every individually scheduled operation of a job is treated as a decision which contains knowledge. Each decision is modeled as a function of a set of job characteristics (e.g., processing time), which are divided into classes using domain knowledge from common dispatching rules (e.g., shortest processing time). A NN is used to capture the predictive knowledge regarding the assignment of operation's position in a sequence. The trained NN could successfully replicate the performance of the GA on the benchmark problem. The developed NN scheduler was then tested against the GA, Attribute-Oriented Induction data mining methodology and common dispatching rules on a test set of randomly generated problems. The better performance of the NN scheduler on the test problem set compared to other methods proves the feasibility of NN-based scheduling. The scalability of the NN scheduler on larger problem sizes was also found to be satisfactory in replicating the performance of the GA.

G. R. Weckman (✉)
Department of Industrial and Systems Engineering,
Ohio University, 280, Stocker Center, Athens, OH 45701, USA
e-mail: weckman@bobcat.ent.ohiou.edu

C. V. Ganduri
Department of Industrial and Systems Engineering,
Ohio University, 279, Stocker Center, Athens, OH 45701, USA
e-mail: ganduri@bobcat.ent.ohiou.edu

D. A. Koonce
Department of Industrial and Systems Engineering,
Ohio University, 283, Stocker Center, Athens, OH 45701, USA
e-mail: koonce@ohio.edu

Keywords Artificial neural networks · Scheduling · Job-shop · Machine learning · Genetic algorithms

Introduction

Scheduling involves the sequencing of activities under time and resource constraints to meet a given objective. It is a complex decision making activity because of conflicting goals, limited resources and the difficulty in accurately modeling real world scenarios. In a manufacturing context, scheduling activities are mapped to operations, and resources to machines. The purpose of a scheduler is to determine the starting time for each operation to achieve the desired performance measures, while satisfying the capacity and technological constraints. In today's highly competitive manufacturing environment, there is a definite need for a robust and flexible approach capable of generating good solutions within an acceptable timeframe.

Scheduling theory is concerned with the formulation and study of various scheduling models and development of associated solution techniques. Some widely researched classical models are: the single machine, parallel machine, flow-shop and the job-shop scheduling models (Baker 1974). Of these, the deterministic job-shop scheduling model has attracted the most attention for two key reasons. First, the generic formulation of the model makes it applicable to non-manufacturing scheduling domains. Second, the problem's sheer intractability has inspired researchers to develop a broad spectrum of strategies, ranging from simple heuristics to adaptive search strategies based on conceptual frameworks borrowed from biology, genetics and evolution.

The deterministic job-shop scheduling problem (JSSP) consists of a finite set of jobs to be processed on a finite set of machines. The difficulty in JSSP lies in the number

of possible schedules. In theory, for an n × m JSSP, the cardinality of a set of possible schedules is $(n!)^m$. Though the set of feasible schedules in which precedence constraints have not been violated is a subset of this set, it is still large enough to discourage complete enumeration for even moderately sized problems. The computation time for algorithms searching the possible solution space to identify an optimal schedule increases exponentially with problem size. Hence, the JSSP belongs to a set of problems classified as NP-Hard problems. French (1982) predicts that it is not possible for algorithms to tackle such problems in polynomial time.

The search-based approach to the JSSP is based on the exploration of the feasible solution space to identify an optimal solution. Adaptive search algorithms like Genetic Algorithms (GAs), Tabu Search and Simulated Annealing have been applied to this domain with success and in many cases are capable of providing optimal or near optimal solutions. Heuristics offer a knowledge-based alternative to the problem. Many dispatching rules are abstractions formulated from expert knowledge of the problem. Elementary priority dispatch rules such as Shortest Processing Time (SPT), First Come First Served (FCFS), Earliest Due Date (EDD), etc. have proven useful in simulation studies of the job shop environment (Blackstone et al. 1982). Ease of application, rapidity of computation and flexibility to changing shop floor conditions are the key reasons for heuristic-based approaches to be widespread in many industrial settings. Their main weakness, however, is that different heuristics cater to different problems and no single heuristic dominates the rest across all scenarios.

A key shortcoming of adaptive search methods for scheduling problems is the lack of insight offered by them into their decision making process. The stochastic nature of search in these algorithms, while often yielding good solutions does not help explain the process by which they are obtained or the properties attributable to the provided solutions. An interesting line of investigation would be to cast the scheduling problem as a learning task with the goal of capturing the properties of known good solutions. In such a formulation, the optimal solutions generated by efficient optimizers provide the desired learning material. In these optimized sequences, each individual operation is treated as a decision which captures some problem-specific knowledge. Hence, these solutions contain valuable information such as the relationship between an operation's attributes and its position in the sequence (solution). An exploration of these sequences by machine learning techniques would capture predictive knowledge regarding the assignment of operation's position in a sequence based on its attributes.

While this approach learns knowledge contained in schedules produced by an optimization method, there is no reason that knowledge contained in schedules produced by a human scheduler could not serve as a training set. In this way, training a neural network scheduler can serve as a mechanism for capturing the knowledge contained in historical scheduling decisions and be an invaluable aid in industrial scheduling.

Background

Artificial Intelligence (AI) aims at constructing artifacts (machines, programs) that have the capability to learn, adapt and exhibit human-like intelligence. Hence, learning algorithms are important for practical applications of AI. The field of machine learning is the study of methods for programming computers to learn (Dietterich 1996). Many important algorithms have been developed and successfully applied to diverse learning tasks such as speech recognition, game playing, medical diagnosis, financial forecasting and industrial control (Mitchell 1997). This research uses Artificial Neural Networks (ANNs) as a machine learning tool to study the scheduling process. An ANN is a data-driven modeling tool that is able to capture and represent complex and non-linear input/output relationships. ANNs are being recognized as a powerful and general technique for machine learning because of their non-linear modeling abilities and robustness in handling the noise-ridden data (Príncipe et al. 2000). Neural networks are used in many important applications, such as function approximation, pattern recognition and classification, memory recall, prediction, optimization and noise-filtering. They are used in many commercial products such as modems, image-processing and recognition systems, speech recognition software, data mining, knowledge acquisition systems and medical instrumentation, etc. (Widrow et al. 1994).

A neural network is composed of several layers of processing elements or nodes. These nodes are linked by connections, with each connection having an associated weight. The weight of a connection is a measure of its strength and its sign is indicative of the excitation or inhibition potential. A neural network captures task-relevant knowledge as part of its training regimen. This knowledge is encoded in the network in: the architecture or topology of the network, the transfer functions used for nonlinear mapping and a set of network parameters (weights and biases). There have been several applications of neural networks in scheduling. Cheung (1994) provides a comprehensive survey of the main neural network architectures used in scheduling. These are: searching networks (Hopfield net), probabilistic networks (Boltzmann machine), error-correcting networks (multilayer perceptron), competing networks and self-organizing networks. Jain and Meeran (1998) also provide an investigation and review of the application of neural networks in JSSP.

Foo and Takefuji (1988) were the first to employ neural networks for the JSSP. They formulate the scheduling problem as an integer linear programming problem and use a

modified Hopfield network to model the problem. The energy function for the network is a linear function and is the sum of starting times of the jobs. In a later work, Foo et al. (1995) investigate the scaling properties of the modified Hopfield network for scheduling problems. Some of the drawbacks of these networks include lack of convergence, convergence to local minima and the number of hardware processing elements. Yang and Wang (2000) proposed an adaptive neural network for the generalized JSSP where the precedence and resource constraints of the problem are mapped to network architecture. The network consists of linear-segmented activation functions. The network generates feasible solutions, which are further improved by heuristics to obtain non-delay solutions.

The other prominent NN systems used in scheduling are the error-correcting networks, which adapt the network parameters (weights and biases) based on the propagation of error between the desired and computed output of the network. A major class in such systems is the multi-layer perceptron (MLP) network, where supervised learning takes place by the back propagation algorithm. Jain and Meeran (1996) propose a modified MLP model, where the neural network performs the task of optimization and outputs the desired sequence. A novel input–output representation scheme is used to encode the JSSP for NN training. Although the method has been able to handle large problem sizes (30×10) compared to other approaches, the generalization capability of the model is limited to approximately 20% deviation from the training sample.

In contrast to the above approach, many applications of the error-correcting networks to JSSP utilize the NN as a component of a hybrid scheduling system. Rabelo and Alptekin (1989) use the NN to rank and determine coefficients of priority rules. An expert system utilizes these coefficients to generate schedules. Dagli and Sittisathanchai (1995) use a GA for optimization and the NN performs multiobjective schedule evaluation. The network maps a set of scheduling criteria to appropriate values provided by experienced schedulers. Yu and Liang (2001) present a hybrid approach for JSSP in which GAs are used for optimization of job sequences and a NN performs optimization of operation start times. This approach has been successfully tested on a large number of simulation cases and practical applications. Fonseca and Navaresse (2002) investigate the use of multi-layer perceptron networks for simulation of the job-shop environment. This work compares the performance of NN in estimating the manufacturing lead time to traditional simulation approaches. In an approach very similar to the NN learning mechanism, Agarwal et al. (2006) proposed an adaptive learning approach for the flow shop scheduling problem. In their approach, the heuristics are guided through a problem search space based on weighted processing time. The weights are adaptive with two parameters, learning rate and reinforcement rate used to influence the extent and memory of search. The authors report a satisfactory performance on several sets of benchmark problems drawn from literature.

Methodology

The goal of the current research is to develop a NN scheduler for the JSSP. To accomplish this, the learning task for the NN needs to be identified. The central premise of the research is that optimal solutions to a scheduling problem have common features which can be implicitly captured by a machine learning tool like NNs. A second premise is that the learned function (i.e., neural network) can be utilized to generate good solutions to new problems. To generate the learning or knowledge base, GAs were chosen for producing optimal solutions as they have proven to be successful in empirical scheduling research.

Scheduling can be understood as a decision making process, where the decision is the selection of the next operation to add to the partial schedule from among a set of competing operations with the objective of minimizing a chosen performance measure. A complete solution (schedule) is thus the consequence of repeated decisions to select the best operation. So, in an optimal solution, every individually scheduled operation of a job is treated as a decision which contains knowledge. Each decision is modeled as a function of a set of job characteristics (e.g., processing time, machine load), which are divided into classes using domain knowledge from common dispatching rules (e.g., shortest processing time). The goal of the NN is then to capture the predictive knowledge regarding the assignment of operation's position in a sequence.

A schematic illustration of the methodology is depicted in Fig. 1. This first task is to generate optimal solutions to a benchmark scheduling problem using GAs. The second task is to model the scheduling function as a machine learning problem by defining a classification scheme, which maps the optimal sequences to data patterns. The third task is to develop a NN model with suitable architecture. The NN is trained on the classification data patterns. The final task focuses on the development of a scheduler which combines a well known active scheduling algorithm with the NN to obtain schedules for any new problems. The different tasks are explained in detail in the following subsections.

Generating GA solutions

The knowledge base for the learning task is constructed from the optimal solutions to a job shop problem generated by a GA. A well-known 6×6 problem instance, ft06 devised by Fisher and Thompson (1963) has been chosen as the benchmark problem. This instance has six jobs, each with six

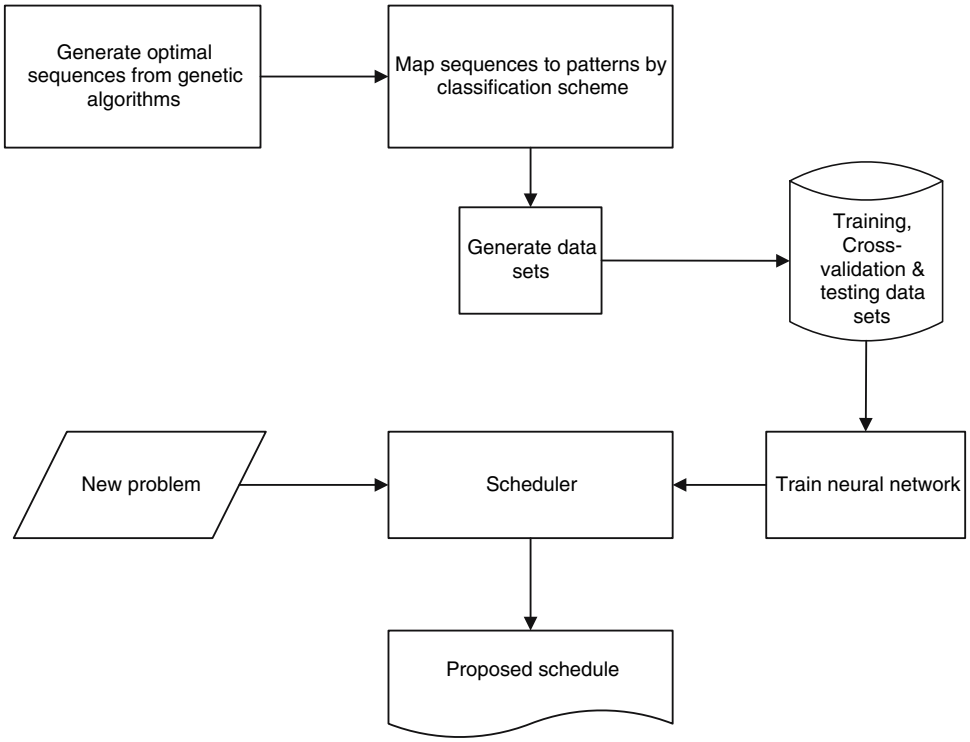

Fig. 1 Schematic illustration of NN-based scheduler

Table 1 The ft06 instance devised by Fisher and Thompson (1963)

Job	Operation					
	1	2	3	4	5	6
1	3,1	1,3	2,6	4,7	6,3	5,6
2	2,8	3,5	5,10	6,10	1,10	4,4
3	3,5	4,4	6,8	1,9	2,1	5,7
4	2,5	1,5	3,5	4,3	5,8	6,9
5	3,9	2,3	5,5	6,4	1,3	4,1
6	2,3	4,3	6,9	1,10	5,4	3,1

operations to be scheduled on six machines and has a known optimum makespan of 55 units. The data for the instance is shown in Table 1 using the following structure: machine, processing time.

A distributed GA implemented in Java developed by Shah and Koonce (2004) was utilized in this research for obtaining solutions to the benchmark problem. A solution generated by the GA is a sequence of 36 operations (6 jobs × 6 machines), like the following: {1, 3, 2, 4, 6, 2, 3, 4, 3, 6, 6, 2, 5, 5, 3, 5, 1, 1, 6, 4, 4, 4, 1, 2, 5, 3, 2, 3, 6, 1, 5, 2, 1, 6, 4, 5}. Each number in the sequence is representative of the job number and the current operation number. The repetition of a job number in the sequence indicates the next operation of that job. The representation of the GA solution is shown in Fig. 2.

On the benchmark instance shown in Table 1, the GA was run 2,000 times. Each run produced a solution and the optimal makespan of 55 units was achieved 1,147 times. The next

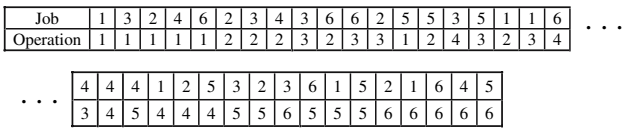

Fig. 2 Representation of the GA solution

step was to transform these 1,147 solutions (chromosomes) into a data structure suitable for the classification task.

Setting up the classification problem

The solutions obtained by the GA contain valuable information relevant to the scheduling process. The learning task was to predict the position of an operation in the GA sequence (chromosome), based on its features or attributes. A sequence contains information about the ordering of operations on each machine. A schedule specifies both the sequence and starting times of operations. The reason for utilizing sequences produced by the GA instead of schedules for the learning task is twofold:

(1) In a GA, chromosomes represent solutions to the optimization problem. In constructing a GA for scheduling, the decision to represent chromosomes either as sequences or schedules is a design decision. In the former case, a decoder is specified to map the sequence to a schedule. The genetic operators needed to manipulate

sequences are simpler than those needed to manipulate schedules. The efficiency of the GA for a tough combinatorial optimization problem like job shop scheduling is an important issue and hence the sequences were utilized to represent chromosomes.

(2) There exists an N:1 mapping between sequences and schedules. This implies that an operation in different positions in two sequences might still occupy the same position in the decoded schedule. As prediction of the position of the operation is the goal of the learning task, sequences were utilized instead of schedules to prevent any loss of information relevant to the learning task.

Based on a study of operation attributes commonly used in priority dispatch rules, the following attributes have been identified as input features: operation, process time, remaining time and machine load. These input features have been clustered into different classes using the concept hierarchy for job shop problems developed by Koonce and Tsai (2000).

Each job in the benchmark problem has six operations that must be processed in a given sequence. The Operation feature identifies the sequence number of the operation ranging between 1 and 6. This feature has been clustered into four classes as: {1} First, {2, 3} Middle, {4, 5} Later, and {6} Last. The ProcessTime feature represents the processing time for the operation. The RemainingTime feature denotes the sum of processing times for the remaining operations of that job and provides a measure of the work remaining to be done for completion of the job assuming no resource conflicts. For the benchmark ft06 instance, the processing times ranged from 1 to 10 units, while the remaining times ranged from 0 to 39 units. Based on the data, three classes (clusters) for these features were identified with the ranges being split into three equal intervals and classified as Short, Medium and Long. The MachineLoad feature determines the machine loading and was clustered into two classes: Light and Heavy. This feature represents the capacity or utilization of machines in units of time and helps in differentiating between possible bottleneck and non-bottleneck machines. These input features thus signify features of the operation that affect its placement in a schedule.

The target concept to be learned is the priority or position in the sequence. Since an operation can be positioned in any one of the 36 locations available in the GA sequence, it may be difficult to discover an exact relationship between the input features and the position. However, if the problem was modified to predict a range of locations for the operation, the learning task becomes simpler. The target feature priority, thus determines the range of positions in the sequence where the operation can be inserted. The possible range of positions have been split into 6 classes and assigned class labels as shown in Table 2. The classification problem with the input and target features is illustrated in Fig. 3.

Table 2 Assignment of class labels to the target feature

Range of positions	Priority
1–6	Zero
7–12	One
13–18	Two
19–24	Three
25–30	Four
31–36	Five

Development of a NN model

There are two aspects related to the development of a NN model. The first is the selection of a suitable architecture, training algorithm, learning constants and termination criteria for building the model. The second relates to the determination of sizes and choice of data patterns for the training, cross-validation and testing data sets. Considerable experimentation was necessary to achieve a good network model of the data. A commercially available software, NeuroSolutions developed by NeuroDimensions Incorporated was used for development and testing of the NN model.

The type of NN chosen for the classification task was a MLP. An MLP consists of groups of nodes arranged in layers. Each node in a layer is connected to all nodes in the next layer by links which have weights associated with them. The input layer contains nodes that represent the input features of the classification problem. A real-valued feature is represented by a single node, whereas a discrete feature with n distinct values is represented by n input nodes. The classification problem under consideration has four discrete input features with four, three, three and two distinct values. Thus, the input layer in the MLP has a total of 12 $(= 4+3+3+2)$ nodes, each representative of an input feature value. The hidden layers map the input to the output layer and the number of nodes in the hidden layers is empirically chosen to obtain a good model. The output layer represents the decision of the classifier. Each output node representing a possible target class and hence there are six output nodes in the MLP. A winner-take-all heuristic is used to determine the class membership when multiple nodes are present in the output layer. The class of the output node with maximum activation or output is the class computed by the network.

A training algorithm is used to determine the values of network parameters which best map a given set of input patterns to desired outputs. The most commonly used training algorithm is the back-propagation algorithm, was first discussed by Rumelhart et al. (1986). The term back-propagation refers to the direction of propagation of error. The goal of the training regimen is to adjust the weights and biases of the network to minimize a chosen cost function. Though several cost functions are available, the function appropriate for classification problems is the cross-entropy function, which is a

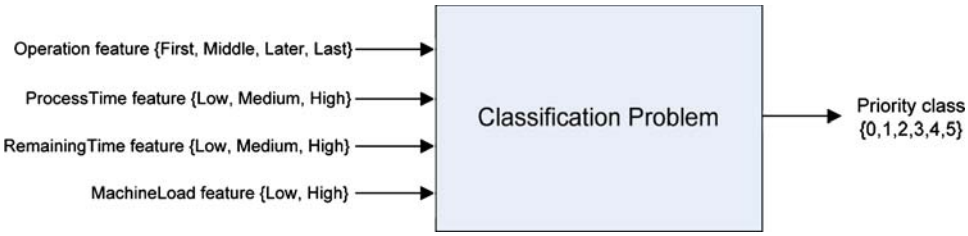

Fig. 3 Input features and target classes for the classification problem

measure of relative entropy between different classes. This function needs to be minimized and the back-propagation algorithm uses a form a gradient descent to update weights. A learning rate or step size parameter scales the correction term in revising the weights during the training process and hence governs the rate of learning. In addition, a momentum parameter was also used to speed up the training process. This parameter further scales the correction based on the memory of the size of previous increment to the weight. In building the NN model, we employed a variant of back-propagation algorithm with momentum learning as specified by Príncipe et al. (2000) and implemented in NeuroSolutions.

The network is initialized with random weights and the training algorithm modifies the weights according to the above discussed procedure. Since random initial conditions (weights) were specified, the network was trained multiple times (runs) before the best model was chosen. Further, in each run, the entire training data set is presented to the network multiple times and each view is known as an epoch. The number of epochs/run and the number of runs are user-specified inputs to the training algorithm.

The 1,147 optimal schedules obtained by the GA represent a total of 41,292 scheduled operations (1,147 schedules × 36 operations/schedule). Assignment of input features and target classes was done for each operation according to the classification scheme described in the previous subsection. Sample data for the classification task is shown in Table 3. The collection of input features-target class pairs used to train the NN is called the training set. The testing set contains patterns not used for the training purpose and is used to evaluate the performance of the network. Even with testing, the model might over fit the training data, causing degradation in performance when tested on new patterns. Cross-validation is a useful technique which avoids this problem of over generalization by periodically testing the model performance on the cross-validation data set during the training phase. The "best" model is the one with least cross-validation error. This classification data set was split into training, cross validation and testing data sets with 70%, 15% and 15% memberships.

Different network architectures were experimented with to determine the best multi-layered perceptron (MLP) classifier. As the final model, a two hidden layered MLP with a 12-12-10-6 architecture and hyperbolic tangent transfer functions at the hidden layers was chosen (see Fig. 4), as it had

Table 3 Sample data for the classification task

Pattern-ID	Operation	Process time	Remaining time	Machine load	Priority
1	First	Short	Long	Light	0
2	Middle	Medium	Long	Light	1
...	...	...	...	...	...
...	...	...	...	...	...
41,291	Later	Medium	Short	Heavy	4
41,292	Last	Short	Short	Light	5

the best classification accuracy for the testing data set. The user-specified training parameters (i.e., learning parameters and the termination criteria) for the 12-12-10-6 MLP classifier are given in Table 4. The quality of the solutions obtained by this classifier is discussed in the results section.

Development of the NN scheduler

A scheduler was developed which is based on the output of the NN. Given an input scheduling problem of size $m \times n$, the scheduler first maps each of the mn operations to an input pattern according to the classification scheme described previously. These mn input patterns are then presented to the trained NN, which assigns a priority index to each of them. For example, to schedule the 6×6 benchmark instance (ft06), the 36 operations are transformed into 36 input patterns and a priority index is assigned to them by the NN.

The well-known Giffler–Thomson (GT) algorithm (Giffler and Thompson 1960), which generates active schedules, was then used for schedule generation. An active schedule can be described as a schedule in which no operation can be started earlier without delaying any other operation. The set of active schedules is guaranteed to contain an optimum solution for any regular performance measure like makespan (French 1982). The GT algorithm iteratively chooses an operation to schedule from among the set of available operations based on the priority index. The operations are first ranked based on the priority index assigned to each operation by the NN. The operation with the lowest rank is scheduled on the identified machine. This process continues till all the mn operations of the problem have been scheduled. The next section discusses the performance of the NN classifier and scheduler on a test set of problems. The GT-algorithm was implemented in Java programming language as a module of

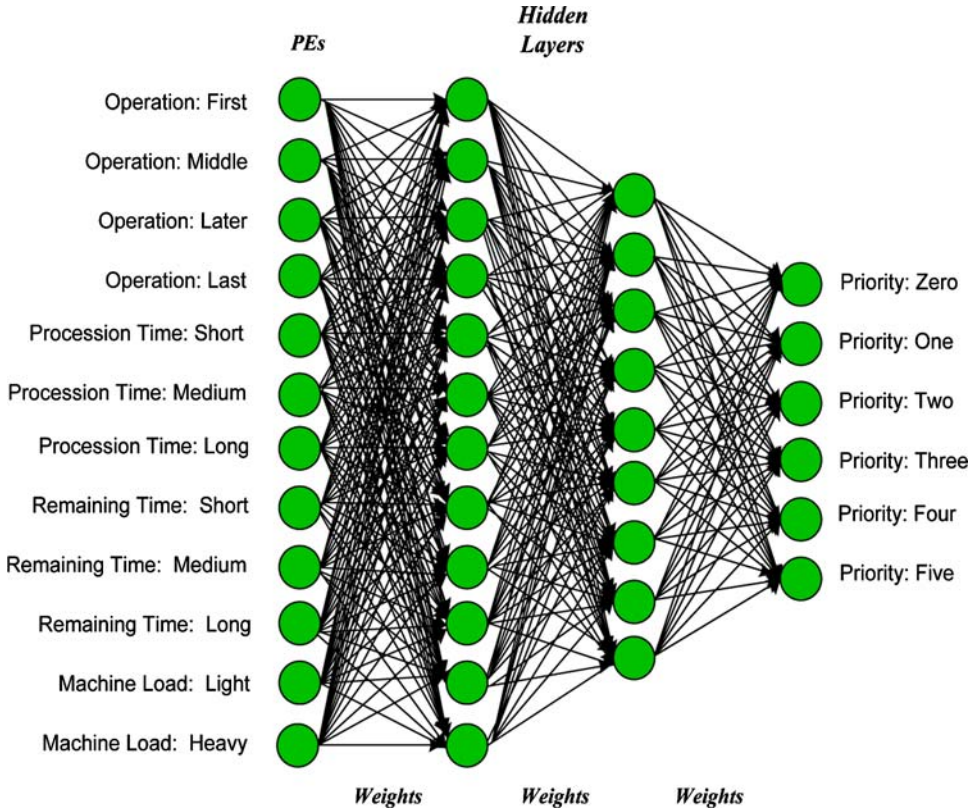

Fig. 4 Multi-layered perceptron (12-12-10-6)

Table 4 Training parameters for the 12-12-10-6 MLP classifier

Network parameters	Value
Step size	0.01
Momentum factor	0.7
Number of runs	10
Number of epochs/run	10,000
Number of epochs without improvement in CV error	500

the IMPlanner system (Sormaz 2003) which is the integrative system for intelligent manufacturing planning and scheduling. The implementation is generic and could also generate schedules with other priority dispatching rules like the shortest processing time, etc.

Results and discussion

Performance of the 12-12-10-6 MLP classifier

A confusion matrix, shown in Table 5 was used to evaluate the performance of the 12-12-10-6 MLP classifier. A confusion matrix is a table where the desired classification (GA solutions) and the output of the classifier are compared on the testing data set.

In the confusion matrix, the diagonal element entries represent the number of testing set instances classified correctly, as set by the GA, by the 12-12-10-6 MLP classifier. The classification accuracy for each class was calculated by dividing the number of correct classifications by the total number of instances in that class. It can be seen from table that often the deviation from the correction classification is only by one class. This deviation can be attributed to the presence of considerable noise in the classification data set. The two main sources of noise were:

1. *GA assignments*: The GA assigned different priorities to the same operation in different schedules, since multiple sequences may produce the same optimal makespan. This led to some ambiguity in the classification data set with the same training patterns having different target features. This considerably increased the complexity of the learning task.
2. *Encoding of the classification problem*: The chromosome sequences were mapped according to the classification scheme into training patterns. While this generalization reduced the dimensionality of the data which was desirable for comprehensibility, it represented a loss of information and a source of noise for the classification task.

Schedule generation and comparison

A comparative summary of the makespans of schedules generated by the GA, NN, Shortest Processing Time (SPT) and other priority dispatching rules for the ft06 instance is provided in Table 6. The performance of the Attribute-Oriented Induction (AOI) rule set is reported from the work

Table 5 Confusion matrix of the 12-12-10-6 MLP classifier

Output/desired	Priority (Zero)	Priority (One)	Priority (Two)	Priority (Three)	Priority (Four)	Priority (Five)
Priority (Zero)	811	81	3	0	0	0
Priority (One)	346	846	244	5	0	0
Priority (Two)	200	428	789	176	7	0
Priority (Three)	0	27	412	1101	506	39
Priority (Four)	0	0	0	56	314	106
Priority (Five)	0	0	1	46	514	1221
Classification accuracy (%)	59.76	61.22	54.45	79.55	23.41	89.38
Total accuracy (%)	61.38					

Table 6 Makespans of schedules for ft06

Scheduler	Makespan	Deviation (%)
Genetic algorithms (GA)	55	0
Neural network (NN)	59	7.27
Attribute-oriented induction (AOI)	67	21.81
Shortest processing time (SPT)	83	50.90
Most work remaining (MWKR)	67	21.81
Shortest remaining processing time (SRMPT)	84	52.72
Smallest ratio of processing time to total work (SPT-TWORK)	71	29.09

of Koonce and Tsai (2000). AOI is a rule induction method using concept hierarchies to aggregate membership of tuples that produced a set of rules which assign a priority to an operation based on either a mean or mode placement. Makespans for the schedules built by using dispatching rules (other than SPT) are from an empirical study conducted by Käschel et al. (1999). Only the GA was able to achieve an optimal makespan of 55 units. The NN, developed in the current work could achieve a makespan of 59 units, a deviation of 4 time units (7.27%) from the optimum makespan. The deviations of other methods ranged from 12 to 29 units (21.8–52.7%). The performance of the NN is considerably better than the performance of other methods in scheduling the benchmark 6×6 problem.

To assess the generalization capabilities of the NN, a test problem set consisting of 10 randomly generated 6×6 problem scenarios was constructed. The motivation in using a test set of 6×6 problems was to keep the scheduling complexity similar to the benchmark ft06 problem, while altering the processing times and precedence constraints of operations. Schedules based on different approaches were built using the GT algorithm described previously, with the priorities for the operations being assigned from the base algorithm (NN, AOI-Mean, AOI-Mode, and SPT). Table 7 shows the performance of these various schedulers on the test cases. The GA solutions are considered the benchmark solutions for comparing other schedulers. The entries indicated in bold in the Table 7 represent the best solutions obtained for the test instance under consideration. The NN scheduler provides better makespans (closest to the GA makespans) for more problem instances than other methods on the test problem set. Also, the NN scheduler performed better than the SPT heuristic in nine of the ten cases. From an inspection of the average makespan values and percentage deviations provided in this table, it is evident that the NN scheduler comes closest to the GA in scheduling the test problems. This demonstrates the generalization capabilities of the NN scheduler as the learning material comes from only from optimal solutions to the benchmark ft06 problem instance.

Analysis of Variance (ANOVA) was used for comparing the performance of alternate schedulers. The experiment was designed as a Randomized Complete Block Design to account for the variability arising from different job-shop problem scenarios in the test set. The assumptions made for this experiment are that the observations are independent and normally distributed with the same variance for each treatment (scheduler). An Anderson-Darling test verified the normality assumption. Bartlett's test was used to validate the assumption of homogeneity of variances. The null hypothesis for this experiment, H_0, (stating that all the treatment means are equal) was tested at 5% significance. The null hypothesis was rejected, concluding that there exists a significant difference in the treatment means.

Duncan's multiple range test was then used to identify the pairs of treatments (schedulers), which had a significant difference in means. The treatment means were sorted in an ascending order. The test statistic is the least significant studentized range, r_p (subscript p denotes the number of treatment means) and depends on the number of means and the degrees of freedom. For p values between 2 and 6, the values of r_p were obtained from the least significant studentized range table. Duncan's critical value, R_p for these means was

Table 7 Makespans obtained by various schedulers on the test set

Scenario-name	GA	NN	AOI-mode	AOI-mean	SPT
ft06-R1	46	50	**49**	53	54
ft06-R2	53	**56**	**56**	58	64
ft06-R3	60	**61**	62	67	71
ft06-R4	48	**56**	60	55	63
ft06-R5	55	**61**	**61**	63	66
ft06-R6	54	**58**	61	59	67
ft06-R7	51	55	**53**	**53**	60
ft06-R8	67	74	76	75	**71**
ft06-R9	54	**57**	59	68	59
ft06-R10	59	**65**	70	70	86
Average	54.7	59.3	60.7	62.1	66.1
Deviation (%)		8.41	10.97	13.53	20.84

The bold values indicates the best solutions obtained for the test instance under consideration

Table 8 Different groups of schedulers

Scheduler	Group
Genetic algorithm (GA)	A
Neural network (NN)	B
Attribute oriented induction (AOI-mean)	B
Attribute oriented induction (AOI-mode)	B
Shortest processing time (SPT)	C

computed. The difference between a pair of means drawn from the set of ordered treatment means was compared with the critical value, R_p. This comparison was carried out for all 15 possible combinations of treatment pairs. The determination of significant difference in means allowed the treatments (schedulers) to be combined into groups as shown in Table 8.

The three groups identified by the Duncan's test correspond to different scheduling approaches for the job-shop problem. The optimization method (GA) provided the best makespan. The machine learning methods (NN, AOI-Mean & AOI-Mode) constituted the second group (B). While not statistically significant, within this group, the NN-based approach provided the best results on the test problems, having the lowest average makespan. Also, the AOI-based methods produce rules which assign priorities to operations. These rules were not sufficient to schedule any randomly generated 6×6 scenario. For some operations, the input features of the operation did not match the rule antecedent and in such cases, an average priority index of 2.5 was assigned to such operations. In contrast to the rule-based approach, the trained NN could successfully describe any randomly generated 6×6 scenario. The performance of all the members in the second group was better than the SPT heuristic.

The NN scheduler has been shown to have good generalization capabilities based on its performance on the test set of 10 random generated 6×6 problems. Another dimension for evaluating the performance of the NN scheduler is its scalability on larger problem sizes. For this purpose, five well-known problem instances from scheduling literature with sizes ranging from 100 to 400 operations were selected. These problems are ft10 (Fisher & Thompson, 1963), la24 and la36 (Lawrence 1984), abz7 (Adams et al. 1988) and yn1 (Yamada and Nakano 1992). The problems were selected from different sources to provide a gradation in the problem size. The makespan achieved by the NN scheduler is compared to the makespan of the best known solution, GA and active schedulers based on the SPT heuristic and random assignment heuristic (see Table 9). The last heuristic selects an operation from among the set of schedulable operations randomly. The performance of different schedulers is graphically illustrated in Fig. 5.

An analysis of the results indicates a consistent pattern in the performance of schedulers. The GA provides the best solution among different schedulers. The NN scheduler outperforms other schedulers (SPT & Random) on all problems. The deviation in the average makespan provided by the NN scheduler is 13.82% from the average makespan provided by the GA, while those of the SPT and random assignment heuristic are 41.56% and 65.43%, respectively. The deviation of NN scheduler from the average GA makespan increases by 5.41 percentage points on the larger problem data set from 8.41% on the test set of 6×6 problems. This compares favorably to the increase in deviation of the SPT scheduler by 20.42 percentage points between the two data sets. This is impressive because the NN is trained on the solutions obtained from a single 6×6 benchmark problem. To be able to provide good solutions with minimum computational effort on considerably larger problems with different sequencing constraints and processing times validates the learning of the NN. It shows that the NN was able to capture certain key problem size-invariant properties of good solutions as part of its training regimen.

An inspection of the results in Table 9 also reveals a consistent divergence in the GA's performance when compared to the best known solutions. This is because the GA used in this research had a simple evolutionary scheme with standard

Table 9 Comparison of makespan performance of different schedulers

Problem name	Size	Best known solution	GA	NN	SPT	Random
ft10 (10 × 10)	100	930	1056	1154	1429	1748
la24 (20 × 10)	200	935	1204	1564	1874	2014
la36 (15 × 15)	225	1268	1538	1721	2289	2377
abz7 (20 × 15)	300	665	828	940	1030	1400
yn1 (20 × 20)	400	886	1139	1183	1539	1998
Average		936.8	1153	1312.4	1632.2	1907.4
Deviation from GA (%)		–	–	13.82	41.56	65.43

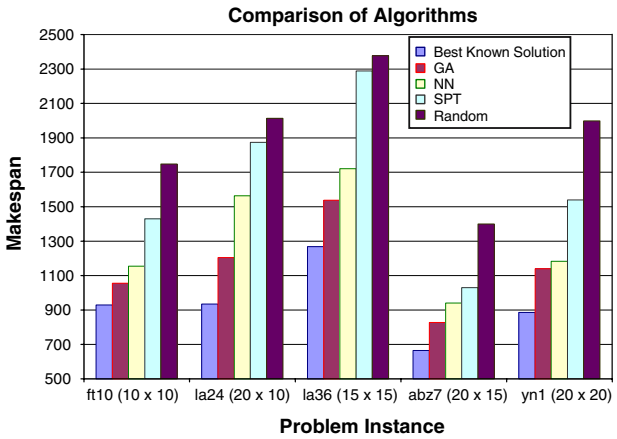

Fig. 5 Illustration of performance of different schedulers

genetic operators and constant parameters. Typically, the GA parameters have to be tuned with considerable experimentation. For larger job shop problems, GA's have been shown to achieve very good solutions with more sophisticated genetic operators and adaptive parameters. In one such study (Yamada and Nakano 1997), the authors report that the by incorporating local search into the GA scheme, near-optimal or optimal solutions were obtained on a similar set of benchmark problems to those utilized in this study.

As the primary objective of this research is to develop and demonstrate the potential of NNs to learn from good solutions, we have employed a simple GA which provided optimal solutions for the 6 × 6 problem. The source of the good solutions for the learning task is not important and is indeed one of the strengths of our approach. In a real world setting, solutions considered as good solutions by experts can be used to train the NN thus providing a computational model based on expert knowledge. Alternately, more effective optimizers could be used to generate good solutions for training the NN. The NN scheduler developed in this research is generic and could be effectively combined with any optimizer to generate good solutions with low computational effort and time.

Though the NN scheduler could not match the performance of the GA, it is computationally less intensive than the GA and offers a more comprehensible scheduling approach. It also provides an attractive alternative to simple heuristics like SPT as it effectively utilizes more problem-specific knowledge in making scheduling decisions with similar computational effort.

For an arbitrary problem, development of an efficient optimizer involves significant design effort and an understanding of the problem domain. The learning framework using NNs presented in this research is generic and could be easily applied whenever there are known good solutions to problems, irrespective of their source. A thorough understanding of the problem domain is not essential for successful application of the methodology. Thus, the learning framework is also particularly suited to problem domains in which the current understanding is limited.

Conclusions

This paper presents a novel knowledge-based approach to solving the job shop scheduling problem by utilizing the various constituents of the machine learning paradigm. The ability of a GA to provide multiple optimal solutions was exploited to generate a knowledge base of good solutions. A NN was successfully trained on this knowledge base. The developed NN scheduler can be utilized to schedule any job shop scenarios of comparable size to the benchmark problem. Also, this methodology can be employed for learning from any set of schedules, regardless of their origin.

A test problem set consisting of 10 randomly generated 6 × 6 scenarios was used to evaluate the generalization capabilities of the NN scheduler. A comparative evaluation of the NN scheduler with other schedulers developed from a different machine learning methodology and SPT heuristic was also undertaken. Among these schedulers, the NN scheduler developed in the current work had the closest average makespan to that of the GA. The NN scheduler also performed satisfactorily on a test set of larger problem sizes, which demonstrates the success of NNs in learning key properties to identify good solutions.

In summary, this research was able to successfully develop a NN scheduler, which provides a close approximation to the performance of a GA scheduler for job shop scheduling problems.

References

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

Agarwal, A., Colak, S., & Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research, 169*(3), 801–815.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Blackstone, J. H., Jr., Phillips, D. T., & Hogg, G. L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research, 20*(1), 27.

Cheung, J. Y. (1994). Scheduling. In C. H. Dagli (Ed.), *Artificial neural networks for intelligent manufacturing* (1st ed., p. 469). London, New York: Chapman & Hall.

Dagli, C. H., & Sittisathanchai, S. (1995). Genetic neuro-scheduler: A new approach for job shop scheduling. *International Journal of Production Economics, 41*(1–3), 135–145.

Dietterich, T. (1996). Machine learning. *ACM Computing Surveys (CSUR), 28*(4es), 3-es.

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth & G. L. Thompson (Eds.), *Industrial scheduling* (pp. 225–251). Englewood Cliffs, New Jersy: Prentice Hall.

Fonseca, D. J., & Navaresse, D. (2002). Artificial neural networks for job shop simulation. *Advanced Engineering Informatics, 16*(4), 241–246.

Foo, Y. S., & Takefuji, T. (1988). Integer linear programming neural networks for job-shop scheduling. *IEEE International Conference on Neural Networks*, San Diego, California, July 24–27 (Vol. 2, pp. 341–348).

Foo, S. Y., Takefuji, Y., & Szu, H. (1995). Scaling properties of neural networks for job-shop scheduling. *Neurocomputing, 8*(1), 79–91.

French, S. (1982). *Sequencing and scheduling: An introduction to the mathematics of the job-shop*. Chichester, West Sussex: E. Horwood.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research, 8*(4), 487–503.

Jain, A. S., & Meeran, S. (1996). *Scheduling a job-shop using a modified back-error propagation neural network*. Paper presented at the Proceedings of the IMS'96 First Symposium on Intelligent Manufacturing Systems Conference, Adapazari, Turkey, May 30–31 (pp. 462–474).

Jain, A. S., & Meeran, S. (1998). Job-shop scheduling using neural networks. *International Journal of Production Research, 36*(5), 1249–1272.

Käschel, J., Teich, T., Köbernik, G., & Meier, B. (1999). Algorithms for the job shop scheduling problem: A comparison of different methods. In *European Symposium on Intelligent Techniques*, Greece (pp. 3–4).

Koonce, D. A., & Tsai, S.-C. (2000). Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. *Computers and Industrial Engineering, 38*(3), 361–374.

Lawrence, S. (1984). *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)*. Pittsburgh, PA: Graduate School of Industrial Administration, Carnegie-Mellon University.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Príncipe, J. C., Euliano, N. R., & Lefebvre, W. C. (2000). *Neural and adaptive systems: Fundamentals through simulations*. New York: Wiley.

Rabelo, L. C., & Alptekin, S. (1989). Using hybrid neural networks/expert systems for intelligent scheduling in flexible manufacturing systems. In *IJCNN: International Joint Conference on Neural Networks*, June 18–22, Washington (Vol. 2, 608 pp).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Shah, N., & Koonce, D. A. (2004). Using distributed genetic algorithms for solving job shop scheduling problems. In *Proceedings of the IIE 2004 Annual Conference*, Houston, TX.

Sormaz, D. N. (2003). Application of space search tools in intelligent manufacturing planning. In *Industrial Engineering Research Conference*, May 18–20, Portland, OR.

Turban, E., & Aronson, J. E. (2001). *Decision support systems and intelligent systems* (6th ed.). Upper Saddle River, NJ: Prentice Hall.

Widrow, B., Rumelhart, D. E., & Lehr, M. A. (1994). Neural networks: Applications in industry, business and science. *Communications of the ACM, 37*(3), 93–105.

Yamada, T., & Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop instances. In R. Manner & B. Manderick (Eds.), *Parallel instance solving from nature 2* (pp. 281–290). Amsterdam: North-Holland.

Yamada, T., & Nakano, R. (1997). Job-shop scheduling. In A. M. S. Zalzala & P. J. Fleming (Eds.), *Genetic algorithms in engineering systems* (pp. 134–160). The Institution of Electrical Engineers.

Yang, S., & Wang, D. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks, 11*(2), 474–486.

Yu, H., & Liang, W. (2001). Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling. *Computers and Industrial Engineering, 39*(3–4), 337–356.