J Intell Manuf (2011) 22:279–288
DOI 10.1007/s10845-009-0287-5

An artificial neural network based heuristic for flow shop scheduling problems

T. Radha Ramanan · R. Sridharan · Kulkarni Sarang Shashikant · A. Noorul Haq

Received: 7 June 2008 / Accepted: 24 June 2009 / Published online: 12 July 2009

Abstract The objective of this paper is to find a sequence of jobs in the flow shop to minimize makespan. A feed forward back propagation neural network is used to solve the problem. The network is trained with the optimal sequences of completely enumerated five, six and seven jobs, ten machine problem and this trained network is then used to solve the problem with greater number of jobs. The sequence obtained using artificial neural network (ANN) is given as the initial sequence to a heuristic proposed by Suliman and also to genetic algorithm (GA) as one of the sequences of the population for further improvement. The approaches are referred as ANN-Suliman heuristic and ANN-GA heuristic respectively. Makespan of the sequences obtained by these heuristics are compared with the makespan of the sequences obtained using the heuristic proposed by Nawaz, Enscore and Ham (NEH) and Suliman Heuristic initialized with Campbell, Dudek and Smith (CDS) heuristic called as CDS-Suliman approach. It is found that the ANN-GA and ANN-Suliman heuristic approaches perform better than NEH and CDS-Suliman heuristics for the problems considered.

Keywords Flow shop scheduling · Makespan · Artificial neural network · Feed forward back propagation

Introduction

Scheduling is the allocation of resources (e.g. machines) to tasks (e.g. jobs) in order to ensure the completion these tasks in a reasonable amount of time. In a flow shop, the technological constraints demand that the jobs pass between the machines in the same order; i.e. If J_1 must be processed on machine M_k before machine M_i then the same is true for all jobs. A permutation schedule is one on which each machine processes the jobs in the same order; i.e. if on machine M_1 job J_i is processed before J_k, then the same is true for all machines (French 1982). The objective is to find a sequence of jobs that minimizes the makespan i.e. time required to complete all the jobs. This problem is found to be NP-hard (Garey et al. 1976).

Except for a minority of problems for which constructive algorithms of polynomial complexity are described, most scheduling problems have been proved to be NP-hard which means the computational requirements for obtaining an optimal solution grow exponentially as the problem size increases. Exact approaches are therefore limited to implicit enumeration algorithms tailored to the problem (branch and bound, dynamic programming) or to formulation in mixed-integer linear programming (MILP) and subsequent solution of the model, which also resorts to implicit enumeration. In either situation, the algorithms display exponential complexity, which means exact methods can only be applied with success to problems of small size (French 1982). For larger scheduling problems, solution approaches described in the literature make use of heuristics, meta-heuristics (like Tabu search, simulated annealing or genetic algorithms) and artificial intelligence (namely constraint programming and neural networks; Lee and Shaw 2000).

The objective of this work is to find a permutation of jobs that minimizes the makespan i.e. time required to complete all the jobs. Complete enumeration, integer programming and branch and bound techniques can be used to find the optimal sequences for small size problems but they are not efficient for large size problems. In view of the combinatorial complexity and time constraints, most of the large problems can be solved only by heuristic methods (Lee and Shaw 2000).

T. R. Ramanan (✉) · R. Sridharan · K. S. Shashikant · A. N. Haq
Department of Mechanical Engineering, National Institute of Technology Calicut, NIT Campus P.O, Calicut 673601, Kerala, India
e-mail: radha_ramanan@nitc.ac.in

This work investigates an approach termed as ANN-Suliman that uses the Artificial Neural Network (ANN) to generate an initial solution to the flow shop scheduling problem. The solution generated by the ANN is then improved using an improvement heuristic proposed by Suliman (2000). The solution is also improved using genetic algorithm.

The solution thus obtained is compared with the results obtained using the Suliman Heuristic initialized with Campbell, Dudek and Smith (CDS) heuristic (Campbell and Smith 1970). The solution is also compared with Nawaz, Enscore and Ham (NEH) (Nawaz et al. 1983) heuristic, as NEH is reported in the literature to give betters results (Taillard 1990).

The present study explores the further improvement in the solution quality, if the improvement phase is provided with better initial solutions. This approach can be considered as the significant contribution of this study. Towards this objective, the result obtained using ANN is initialized to Suliman's heuristic for further improvement and the results obtained using CDS is also initialized to Suliman's heuristic. The solution obtained using ANN is initialized to GA for further improvement. The problem size considered for ANN, testing of ANN results with benchmark problems and improvement of the ANN results with GA can be considered as unique contributions to the literature.

The rest of the paper is organized as follows. The next section presents brief review of the relevant literature. It is followed by the section on the "Problem description". We then describe the use of neural network to get an initial solution. We also explain the architecture used for the problem under study in the same section. The discussion is then followed by the section "Improvement of initial solution using Suliman's heuristic and genetic algorithm". Finally results and conclusion are presented.

Literature review

A survey of literature shows that most of the heuristics for flowshop scheduling aim at minimizing makespan (MS). Johnson's algorithm (Johnson 1954) is the earliest known optimal solution algorithm for n jobs two machines flow shop scheduling problems. Palmer (1965) proposed a slope order index to sequence the jobs on the machines based on the processing time. The heuristic proposed by Campbell, Dudek and Smith (CDS) (Campbell and Smith 1970) is basically an extension of Johnson's algorithm. It builds $m - 1$ schedules by clustering the m original machines into two virtual machines and solving the generated two machine problem by repeatedly using Johnson's rule.

Gupta (1971) suggested another heuristic which is similar to Palmer's heuristic. He defined his slope index in a different manner taking into account some interesting facts about optimality of Johnson's rule for the three machine problems. Dannenbring (1977) developed a procedure called Rapid Access (RA). It attempts to combine the advantages of Palmers slope index and the CDS methods. NEH (Nawaz et al., 1983) heuristic algorithm is based on the assumption that a job with high total processing time on all the machines should be given higher priority than job with low total processing time. The NEH algorithm does not transform the original m-machine problem in to one artificial two-machine problem. It builds the final sequence in a constructive way, adding a new job at each step and finding the best partial solution.

Ho and Chang (1991) developed an improvement heuristic for solving the flow shop scheduling problem. The method minimizes gaps between successive operations in solutions generated by other heuristic. In other words, it minimizes machine idle time and job waiting time resulting from an initial solution. The heuristic produced was compared with five well-known heuristics, and it showed better performance with respect to makespan, mean flow time, and mean utilization.

Rajendran (1995) in his improvement heuristic takes the first seed from CDS algorithm. A heuristic preference relation is proposed and is used as the basis to restrict the search for possible improvement in the multiple objectives. Koulamas (1998) reported a new two phase heuristic, Heuristic Flowshop scheduling with C_{max} objective (HFC). In the first phase, the HFC heuristic makes extensive use of Johnson's algorithm. The second phase improves the resulting schedule from the first phase by allowing job passing between machines, i.e. by allowing non-permutation schedules.

Suliman (2000) developed a two phase improvement heuristic. In the first phase, a schedule is generated with the CDS heuristic method. In the second phase, the schedule generated is improved with a job pair exchange mechanism with a directionality constraint. The solutions obtained are compared with the solutions obtained by the NEH sequencing method. The comparison shows similar performance by the two methods.

Genetic algorithms (GAs) have been widely used to optimize the performance criteria. Holland (1975) first introduced GAs as a highly robust search algorithm. GAs may be viewed as evolutionary or population based algorithms. Reeves (1995) in his genetic algorithm uses a one point order crossover and shift mutation. Reeves also chose to seed the initial population with a good sequence among randomly generated ones. This good sequence is obtained with the NEH heuristic. Chen et al. (1995) generated the initial population with the CDS and RA heuristics and also from simple job exchanges of some of the individuals. Only the crossover operator is applied (no mutation), and the crossover used is the Partially Mapped Crossover or PMX. Murata et al. (1996) use the two-point crossover operator and a shift mutation along with an elitist strategy to obtain good solu-

tions for the PFSP. The authors implemented two hybrid versions of the genetic algorithm; genetic simulated annealing and genetic local search. Sridhar and Rajendran (1996) and Venkata Ranga et al. (1996) have also used GA to optimize the performance criteria.

Wang et al. (2003) have proposed a hybrid approach of ordinal optimization and genetic algorithm called as order based genetic algorithm (OGA) for flow shop scheduling problems. Problems were solved from the Matttfeld and Vaessens OR Library (http://mscmga.ms.ic.ac.uk). The simulated results show that the OGA gives better results than GA, NEH and Blind search methods. They have also tested the various parameters of OGA and have provided statistical results.

Arroyo and Armentano (2005) have proposed a genetic local search for multi-objective flowshop scheduling problems. The two sets of objectives of the work were as follows: optimization of makespan and maximum tardiness; makespan and total tardiness. The concept of pareto dominance is used to assign fitness to the solutions and in the local search procedure. Their results are compared with the results of two other genetic local search methods (proposed by Ishibuchi and Murata and Jaskiewicz) for bigger size problems.

Simulated annealing is a stochastic optimization technique derived from statistical mechanics for finding globally optimal solutions to combinatorial problems. It was originally developed as a simulation model for a physical annealing process of condensed matter (Metropolis et al. 1953). Kirkpatrick et al. (1983) were the first to point out the relevance of the simulation techniques from statistical physics to combinatorial optimization problems. Much research has been done on the application of simulated annealing to sequencing (Osman and Potts 1989; Ogbu and Smith 1990)

Nearchou (2004) presents a hybrid simulated annealing (SA) for the permutation flow shop scheduling problems. The proposed approach combines the characteristic of a canonical SA procedure together with the procedure borrowed from the field of genetic algorithms (GA). The proposed hybrid algorithm is compared with two other simulated annealing approaches and a genetic algorithm approach. The approach performs near optimal schedules in a shorter computation time.

Wang and Zheng (2003a) propose an effective hybrid heuristic approach. Simple genetic algorithm (SGA) is initialized with the results of NEH, multicrossover operators are applied and finally mutation is replaced with metropolis sample of simulated annealing. The results of the hybrid approach have been shown superior to the results of NEH, SGA and NEH + SGA

Many researchers have used also Artificial Neural Network (ANN) approach for modeling and analysis of scheduling problems. Sabuncuoglu and Gurgun (1996) in their paper used a variation of Hopfield network to obtain inhibitory connections so that feasibility could be achieved. In addition to the feasibility, the network searches for a minimum energy level corresponding to the value of cost functions (i.e. makespan or mean tardiness). The paper has discussed the ANN approach in detail using the single machine mean tardiness scheduling problem and the makespan of job shop problem.

El-bouri et al. (2000) presented the methodology of sequencing jobs on a single machine using neural network. They have used the backpropagation neural network to solve the problem. Akyol (2004) used the artificial neural networks to model six different heuristic algorithms applied to n jobs m machines flow shop scheduling problem with the objective of minimizing makespan. Lee and Shaw (2000) have presented a hybrid algorithm using neural network and genetic algorithm to solve the flow shop scheduling problem. The decision environment in which the work is based is a flow shop for printed circuit board. The problem sizes considered are also small ranging from three machines to seven machines only.

El-bouri et al. (2005) presented the neural network approach to enhance local search in the permutation flowshop. They have used total flow time as performance criterion. Tang et al. (2005) propose a neural network method for the dynamic scheduling of hybrid flow shop with dynamic job arrivals. Simulation results show that the method performs well for various scheduling criteria. In order to obtain training examples for the neural network, the performance of some dispatching rules is studied through simulation that has demonstrated effectiveness in the previous related research. The results are then transformed into training examples. The training process is optimized by the delta-bar-delta method that can speed up training convergence.

Wang and Zheng (2003b) proposed a modified evolutionary programming (MEP) in order to avoid premature convergence and to balance the exploration and exploitation abilities of simple evolutionary programming (SEP). The bench mark problems are taken from OR Library of Matttfeld and Vaessens. Their study finds that MEP gives better results than SEP and NEH methods.

Temiz and Erol (2004) propose a fuzzy branch and bound algorithm approach for the flow shop scheduling problem. The branch and bound algorithm of Ignall and Schrage is modified and rewritten for three-machine flow shop problems with fuzzy processing times. Fuzzy numbers is used to determine the minimum completion time. Proposed algorithm gets a scheduling result with the membership function for the final completion time.

In the present paper, the neural network architecture proposed by El-bouri et al. (2005) has been used. The network is trained for small size problems and then this trained network is used for obtaining sequence for large size problems. This sequence is given as a starting sequence to the improvement heuristic given by Suliman to get final solution. Problems are taken from benchmark problems given in literature (Taillard 1993).

Problem description

The flow shop problem can be formulated as follows. Each of n jobs from the job set $i = \{1, 2, \ldots, n\}$, for $n > 1$, has to be processed on m machines 1,2,….,m in the order given by the indexing of the machines. Thus job j, $j \in J$, consists of sequence of m operations; each of them corresponding to the processing of job j on machine i during an uninterrupted processing time $p_{ij} \geq 0$. It is assumed that a zero processing time on a machine corresponds to a job performed by that machine in an infinitesimal time. Machine j, $j = 1, 2, \ldots, m$, can execute at most one job at a time, and it is assumed that each machine processes the job in the same order (Nowicki and Smutnicki 1996).

The objective is to find a sequence for the processing of the jobs on the machines so that the total completion time or makespan of the schedule ($C_{\max}$) is minimized. The processing times needed for the jobs on the machines are denoted as p_{ij}, where $i = 1,\ldots,n$ and $j = 1, \ldots, m$; these times are fixed, known in advance and non-negative. There are several assumptions that are made regarding this problem:

- Each job i can be processed at most on one machine j at the same time.
- Each machine m can process only one job i at a time.
- No preemption is allowed, i.e. the processing of a job i on a machine j cannot be interrupted.
- The set-up times of the jobs on machines are included in the processing times.
- The machines are continuously available.
- In-process inventory is allowed. If the next machine on the sequence needed by a job is not available, the job can wait and join the queue at that machine.

The flow shop considered in the present work consists of ten machines and capable of handling a specified number of jobs only.

Initial solution using ANN

A neural network is a collection of highly interconnected processing units (called neurons or nodes) that can be trained to learn relationships between input and output patterns. One of the most frequently used neural network architecture is back-propagation neural network (BPN). This architecture consists of one input layer, one output layer and at least one hidden layer of neurons. The BPN is trained by presenting patterns of data at the input layer together with the corresponding desired output patterns at the output layer. The weights placed on connections between the nodes in the network's processing layers are updated by using an algorithm called back-propagation (BP) algorithm. The BP algorithm minimizes the error between network's output and desired output. A series of input-output pairs are presented to the neural network to train the network. When mean square error between desired output and actual output is minimized, the neural network is deemed trained. A trained neural network is expected to produce an output, based on the relationship it has learnt, whenever a new pattern (one not seen in the training set) is introduced at the input layer.

The proposed neural network has two hidden layers containing 30 and 20 neurons respectively. This particular design, in which the size of the input layer depends on the number of machines, means that separate and individually trained networks are needed for flowshops having different numbers of machines. The number of hidden layers and number of neurons in that layer are determined empirically by trying different combinations. The values for the input patterns must be between zero and one. Hence, the input data must be normalized. The minimum and maximum processing times in the flowshops considered in this study are 1 and 99 minutes. Therefore, the divisor of 100 in Eq. (1) guarantees that processing times in the range (1, 99) are covered in the normalization.

Each job is represented by a vector that contains a number of elements that describe the processing time of the job individually and relative to the machine workloads and other jobs to be scheduled. The size of this vector is three times the number of machines (3 m). The input layer of the proposed neural network has 3 m nodes allocated as follows:

- The first m nodes contain a job's processing times on each of the m machines.
- The middle m nodes contain the average processing times on each of the m machines.
- The last m nodes contain the standard deviation of the processing times on each of the m machines.

Here, the network is trained for the relationships between the jobs in a given batch. To get this relationship, the standard deviation and averages of the jobs on each machine is used. The averages and standard deviation will guarantee that the same jobs in different batches will be distinguished clearly for easier training of the ANN. For example, Job 1 in the batch 1,2,3,4,5 will be different from Job 1 in the batch 1,2,3,4,6. The first m values for the vector representing Job 1 will always be the same but, the next ten values in the vector will vary according to the batch of jobs in which it is present.

The output layer has only one node, regardless of m. The output node assumes values between 0.1 and 0.9. Problems are taken from the benchmark problems in the literature with 20 jobs, 30 jobs and 50 jobs.

The first step is to train the neural network. For this purpose, five, six and seven jobs problems are generated from the given problem set of 20 jobs or 50 jobs. These problems are solved by complete enumeration to get the optimal solution. In case of multiple optima, the sequence having lesser total flow time is selected as an optimal one. Each five job problem will give five input–output pairs called exemplars, while each six jobs problem will give six exemplars; similarly seven jobs problem will give seven exemplars. In all, 10,005 exemplars are generated by this process. In these exemplars, half of the population i.e. 5,005 exemplars represent the seven jobs problem while 3,000 are from six jobs problem and remaining 2,000 exemplars are from five jobs problem.

The values assumed by the nodes of the input layer when a job $i$is presented to the network, are computed by using following equations:

$$\text{Node}\, q = \begin{cases} \dfrac{p_{i,q}}{100} & q = 1, \ldots, m \\ \dfrac{\overline{p}_{(q-m)}}{100} & q = m+1, \ldots, 2m \\ \sqrt{\dfrac{X_{(q-2m)} - n\overline{p}^2_{(q-2m)}}{(n-1) \times 10^4}} & q = 2m+1, \ldots, 3m \end{cases} \quad (1)$$

where

$$\overline{p}_{(k)} = \frac{1}{n}\sum_{i=1}^{n} p_{i,k} \quad \text{where}\, k = q - m \quad (2)$$

and

$$X_{(l)} = \sum_{i=1}^{n} p^2_{i,k} \quad \text{where}\, l = q - 2m \quad (3)$$

The target output (O_i) for the ith job in the optimal sequence is determined by

$$o_i = 0.1 + \frac{0.8(i-1)}{(n-1)} \quad (4)$$

Equation (4) distributes the targets for the three jobs in equal interval between 0.1 for the first job to 0.9 for the last job. The network is trained with these 10005 exemplars by using *Neural Network Toolbox of Matlab 7.0.* After sufficient training, the network is presented with the 20 and 50 jobs problem and the sequence is obtained by arranging the jobs in ascending values of neural network's output

The first step in applying a trained network to produce a sequence for a new problem is to use Eq. (1) to create n input patterns representing the n jobs. These patterns are applied one at a time, to the input layer of the trained network. An output value is generated in response to each input pattern. The outputs obtained after all input patterns have been processed and then sorted in increasing order, and the job associated with each of these outputs is noted. The resulting sequence of these jobs represents the neural network's recommended sequence.

Table 1 Processing times for the five jobs

Jobs	Processing times on machines									
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
1	46	61	3	51	37	79	83	22	27	24
2	52	87	1	24	16	93	87	29	92	47
3	79	51	58	21	42	68	38	99	75	39
4	45	25	85	57	47	75	38	25	94	66
5	97	73	33	69	94	37	86	98	18	41

To demonstrate how the neural network is employed to construct a sequence from a set of jobs, the example of 5 jobs 10 machines problem from the data set is considered. Suppose jobs 1, 2, 3, 4 and 5 from the example under study are to be sequenced. Their processing times on each machine are shown in Table 1.

Each of the five jobs is first expressed as a thirty element vector (vector consisting of thirty rows) of input nodes by using Eqs. (1)–(3). The results of this step are displayed in Table 2. Next, these vectors are introduced one at a time, at the input layer of the neural network that has been trained for the 10-machine flowshop. The corresponding output from the network in each instance is given in the last row of Table 2. By sorting the six jobs in non-decreasing order of their neural outputs, the neural sequence is obtained, which is shown in Table 3.

Improvement of initial solution using Suliman's heuristic and genetic algorithm

Second phase of Suliman's heuristic

Suliman (2000) developed a two phase improvement heuristic. In the first phase, a schedule is generated with the CDS heuristic method. In the second phase, the schedule generated is improved with a job pair exchange mechanism with a directionality constraint. In the present study, for the first phase a schedule is generated using ANN above and given for improvement to the second phase, which is improved using a pair exchange mechanism. In order to improve the performance of the pair exchange mechanism, a directionality constraint is proposed by which the search load may be reduced significantly compared with that required for the exhaustive pair exchange mechanism. It is assumed that if movement of a job in a specific sequence results into an improved sequence measure, then such improvement will be achieved by moving that job only in one direction while its movement in the other direction will deteriorate the measure. Thus, a job is moved one step in one direction, and if there is an improvement, the movement continues in that direction until no further

Table 2 Input layer vectors and outputs for the five job ten machine problem

Input elements	Jobs				
	1	2	3	4	5
1	0.4600	0.5200	0.7900	0.4500	0.9700
2	0.6100	0.8700	0.5100	0.2500	0.7300
3	0.0300	0.0100	0.5800	0.8500	0.3300
4	0.5100	0.2400	0.2100	0.5700	0.6900
5	0.3700	0.1600	0.4200	0.4700	0.9400
6	0.7900	0.9300	0.6800	0.7500	0.3700
7	0.8300	0.8700	0.3800	0.3800	0.8600
8	0.2200	0.2900	0.9900	0.2500	0.9800
9	0.2700	0.9200	0.7500	0.9400	0.1800
10	0.2400	0.4700	0.3900	0.6600	0.4100
11	0.6380	0.6380	0.6380	0.6380	0.6380
12	0.5940	0.5940	0.5940	0.5940	0.5940
13	0.3600	0.3600	0.3600	0.3600	0.3600
14	0.4440	0.4440	0.4440	0.4440	0.4440
15	0.4720	0.4720	0.4720	0.4720	0.4720
16	0.7040	0.7040	0.7040	0.7040	0.7040
17	0.6640	0.6640	0.6640	0.6640	0.6640
18	0.5460	0.5460	0.5460	0.5460	0.5460
19	0.6120	0.6120	0.6120	0.6120	0.6120
20	0.4340	0.4340	0.4340	0.4340	0.4340
21	0.2315	0.2315	0.2315	0.2315	0.2315
22	0.2347	0.2347	0.2347	0.2347	0.2347
23	0.3608	0.3608	0.3608	0.3608	0.3608
24	0.2104	0.2104	0.2104	0.2104	0.2104
25	0.2870	0.2870	0.2870	0.2870	0.2870
26	0.2078	0.2078	0.2078	0.2078	0.2078
27	0.2597	0.2597	0.2597	0.2597	0.2597
28	0.4015	0.4015	0.4015	0.4015	0.4015
29	0.3623	0.3623	0.3623	0.3623	0.3623
30	0.1521	0.1521	0.1521	0.1521	0.1521
Output	0.99997	0.41032	0.33002	0.10090	0.99331

improvement is achieved. Otherwise, the job is moved in the other direction. If there is still no improvement, the job is left at its initial position in the sequence and another job is considered. An ordered search is adopted in which all possible adjacent job interchanges are considered in a sequential manner provided that the directionality constraint is maintained.

Generating initial population for GA

GA deals with a population of solutions and not with a single solution, and ANN gives only one sequence as solution. Hence, there is a need to initialize GA with an initial set of population from the obtained ANN solution. In the method called ANN-GA, the population is initialized with one sequence obtained through ANN and the remaining N - 1 sequences are generated randomly, where N is the population size.

Table 3 Sequence obtained after sorting the neural output

Sorted output	Sequence
0.10090	4
0.33002	3
0.41032	2
0.99331	5
0.99997	1

Genetic algorithm

Genetic algorithm (GA) may be described as a mechanism that mimics the genetic structure of the species. The main difference with other neighbourhood search techniques is that GA deals with a population of solutions rather than with a single solution. In GA, the main operators used to generate and explore the neighbourhood of the population and select a new generation are *selection, crossover and mutation.*

In this study GA with roulette wheel selection procedure, partially matched crossover operator and shift mutation operator is used. The premature convergence problem is one that has been noted by several researchers in applying GA. In this paper, the following method is used to come out of the premature convergence. If for a specified number of generations, there is no improvement of the solution then, the GA again restarts with the available best solution.

Fitness function

Genetic algorithms mimic the 'survival of the fittest' principle of nature to make a search process. Therefore, genetic algorithms are naturally suitable for solving maximization problems. Minimization problems are usually transformed into maximization problems by some suitable transformation. In general, a fitness function $F(x)$ is first derived from the objective function $f(x)$and used in successive genetic operations. For maximization problems, the fitness function can be considered to be same as the objective function or $F(x) = f(x)$. For minimization problems, the fitness function is an equivalent maximization problem chosen such that the optimum point remains unchanged. A number of such transformations are possible. The following fitness functions are often used:

$$F(x) = 1/(1 + f(x)) \quad \text{or} \quad F(x) = 1/f(x) \quad \text{or} \quad F(x) = R - f(x)$$

where R is a sufficiently large number.

These transformations do not alter the location of the minimum, but converts a minimization problem into an equivalent maximization problem. The fitness function value of a string is known as string's *fitness*.

In the present study, the objective function is minimization of makespan (M) and hence the following function is used to convert minimization problem into maximization problem:

Let, $M = f(x)$

$F(x) = R - M$, where, R is a sufficiently large number, in this case it is taken as summation of processing times of all the jobs on all machines.

Crossover operation

The crossover adopted here is partially matched crossover (PMX). In PMX, two strings are aligned, and two crossing sites are picked uniformly at random, along the strings. These two points define a matching section that is used to effect a cross through position-by-position exchange operations. To see this, consider two strings:

$$\begin{array}{l} \text{A} = 9\ 8\ 4 \mid 5\ 6\ 7 \mid 1\ 3\ 2\ 10 \\ \text{B} = 8\ 7\ 1 \mid 2\ 3\ 10 \mid 9\ 5\ 4\ 6 \end{array}$$

PMX proceeds by position wise exchanges. First, mapping string B to string A, the 2 and 5, the 3 and the 6, and the 10 and the 7 exchange places. Similarly mapping string A to string B, the 5 and the 2, the 6 and the 3, and the 7 and the 10 exchange places. Following PMX we are left with two offsprings, A′ and B′ where each of these offsprings contains ordering information partially determined by each of its parents such that repetition of a gene is avoided:

$$\begin{array}{l} \text{A}' = 9\ 8\ \ 4 \mid 2\ 3\ 10 \mid 1\ 6\ 5\ 7 \\ \text{B}' = 8\ 10\ 1 \mid 5\ 6\ 7 \mid 9\ 2\ 4\ 3 \end{array}$$

Mutation

In the present work, shift mutation is adopted. In this type of mutation, two random points are chosen along the chromosome, and instead of swapping the positions of genes at these positions, the first gene is replaced to the other position and entire genes before or after that position is shifted. This can be explained with the help of the following example. Consider a string A = (a b c d e f g h) is to be mutated, and the two random numbers are 3 and 7. Then, the gene at third position will take the seventh position and all the genes from fourth to seventh in the previous chromosome are shifted to left to give new chromosome A′. Thus the new chromosome obtained will be A′ = (a b d e f g c h). Mutation is also done probabilistically, with a small probability of mutation.

Steps of genetic algorithm

After fine tuning the probabilities of crossover and mutation, the values of parameters are set as follows:

Population size = 60.
Probability of crossover = 0.8
Probability of mutation = 0.2
Maximum number of generations = 50
Number of generation to restart the GA if solution is not improved = 10
Number of iterations =30

The steps of the genetic algorithm are as follows:

(i) Initialize P_c=0.8, P_m=0.2, count_gen=1, count = 0, best = ann_ms, current = ann_ms, iteration = 1.
(ii) Generate Initial Population.
(iii) If Iteration $\leq$ 30 goto step iv else goto step xv.
(iv) If count_gen > 50 or count > 10 goto step xiii else go to step v.
(v) Calculate the fitness value of individuals.
(vi) Selection.
(vii) Do Crossover and Mutation to get new population.
(viii) count_gen = count_gen +1.
(ix) If new_solution $\geq$ current, count = count+1; goto step xii else count = 0 go to step x.
(x) current = new_solution.
(xi) If current $\leq$ best, best = current.
(xii) Go to step iv.
(xiii) Iteration = Iteration+1.
(xiv) Go to step iii.
(xv) Best_sequence = sequence corresponding to best. Stop.

where, ann_ms = makespan for the sequence given by ANN.
current = current best makespan in the generation.
best = best makespan found so far.
P_c = probability of crossover.
P_m = probability of mutation.

Results and discussion

In the first phase of Suliman heuristic, a sequence is generated by using neural network, called as ANN-Suliman approach. A sequence is also generated by using CDS heuristic, called as CDS-Suliman heuristic. These two sequences are given to the phase II of the Suliman heuristic to obtain the final sequences. Makespan of both the approaches are compared. Five instances each of 20, 30 and 50 jobs taken from benchmark problems are solved and results are compared. Also,

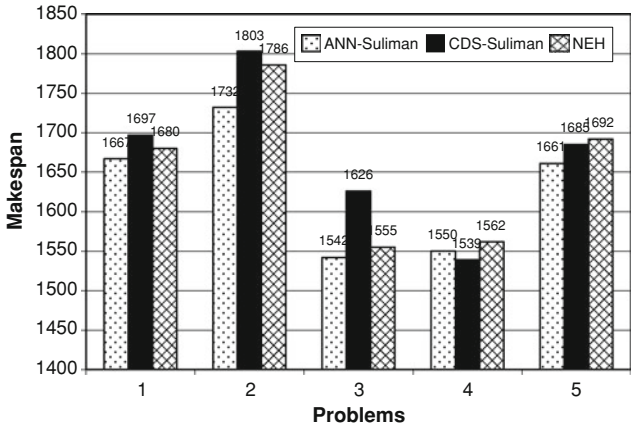

Fig. 1 Comparison of makespan for 20 jobs problem

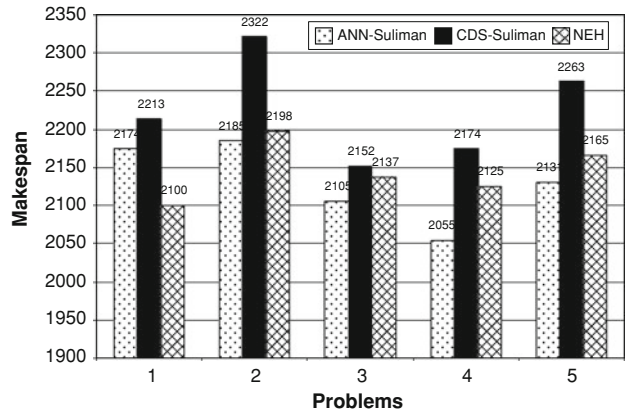

Fig. 2 Comparison of makespan of 30 jobs problem

the makespan values thus obtained are compared with that of NEH. The results are shown in Figs. 1, 2 and 3.

Figure 1 shows the comparison of makespan for 20 jobs 10 machines problem set. In four of the five instances, ANN-Suliman heuristic is found to provide better results, and in one instance, CDS-Suliman heuristic provides better results. In all the instances ANN-Suliman heuristic has given better results than NEH heuristic.

Figure 2 compares the results of the performance measure under consideration for 30 jobs 10 machines problem set. It shows that ANN-Suliman heuristic has performed better in all five instances when compared with CDS-Suliman heuristic. It is also found that ANN-Suliman outperforms NEH heuristic in four of the five instances.

Figure 3 compares the results of the performance measure under consideration for 50 jobs 10 machines and shows that NN-Suliman has performed better in all five instances when compared with CDS-Suliman. In one instance the result of NN-Suliman is same as that of NEH. NEH outperforms the other two heuristics in four instances.

For the twenty job problems and thirty job problems, as shown in Figs. 1 and 2, ANN-Suliman approach gives better

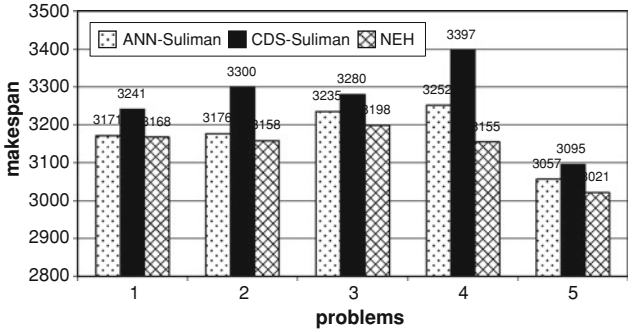

Fig. 3 Comparison of makespan for 50 jobs problem

solutions than CDS-Suliman and NEH heuristic in most of the instances. This is because, in the initial sequence which is given by ANN, the jobs are placed as close as possible with respect to their positions in optimal sequences. Hence, the second phase of Suliman heuristic is able to improve the solution with the constrained improvement mechanism.

But as job sizes are increased (i.e. in this case 50 jobs), more training of neurons are required because of NP hard nature of the problem to improve the solution quality. The directionality constraint imposed by Suliman heuristic also does not help to improve the solution further.

The objective of the present work was to study the impact of better solution quality that is initialized to the second phase of the Suliman's heuristic and compare the results initialized with CDS heuristic. All the results are compared with those obtained using NEH heuristic and NEH heuristic. Table 4 show the quality of initial solutions of ANN and CDS that is initialized to the second phase of Suliman, called as ANN-Suliman and ANN-GA for 20 jobs, 30 jobs and 50 jobs problem set respectively.

Due to the NP hardness nature of the problem and directionality constraints of the Suliman's heuristic, the results of ANN-Suliman are found not better than NEH heuristic, though the results are better than ANN-CDS heuristic. Hence the authors initialized the results of ANN to GA to obtain an improved performance measure to one small size problem(say 20 jobs and one bigger size problem (say 50 jobs). The results of the study are given in Table 5.

It can be observed from the Table 5 that ANN-GA performs better than all the heuristics, viz. CDS-Suliman, ANN-Suliman and also NEH. Genetic algorithm with its inherent characteristics of jumping out of the local optimal is able to obtain better results.

Conclusion

In this paper, ANN based improvement heuristic is proposed for flow shop scheduling problem. It uses genetic algorithm and Suliman heuristic for improvement. From the results, it is inferred that for smaller problems with size up to 30 jobs,

Table 4 Makespan of ANN, CDS (initial solutions) and NEH

No.	Initial (first phase) solutions		Second phase solutions		NEH
	ANN	CDS	ANN-Suliman	CDS-Suliman	
50 Jobs					
1	3,288	3,464	3,171	3,241	3,168
2	3,374	3,392	3,176	3,300	3,158
3	3,408	3,436	3,235	3,280	3,198
4	3,439	3,597	3,252	3,397	3,155
5	3,167	3,231	3,077	3,095	3,075
30 Jobs					
1	2,328	2,269	2,174	2,213	2,100
2	2,347	2,383	2,185	2,322	2,198
3	2,214	2,253	2,105	2,152	2,137
4	2,175	2,343	2,055	2,174	2,125
5	2,237	2,341	2,131	2,263	2,165
20 Jobs					
1	1,770	1,774	1,667	1,697	1,680
2	1,817	1,879	1,732	1,803	1,786
3	1,644	1,658	1,542	1,626	1,555
4	1,665	1,610	1,550	1,539	1,562
5	1,830	1,796	1,661	1,685	1,692

Table 5 Results of CDS-Suliman, ANN-Suliman, ANN-GA and NEH

Makespan				
Taillard's seed no.	CDS-Suliman	ANN-Suliman	ANN-GA	NEH
Set-I problems (20 jobs, 10 machines)				
1	1,697	1,667	1,624	1,680
2	1,803	1,732	1,691	1,786
3	1,620	1,542	1,525	1,555
4	1,539	1,550	1,500	1,562
5	1,685	1,661	1,642	1,692
Set-II problems (50 jobs, 10 machines)				
1	3,241	3,171	3,175	3,168
2	3,300	3,176	3,030	3,150
3	3,280	3,235	3,010	3,198
4	3,397	3,252	3,126	3,155
5	3,095	3,077	3,062	3,075

ANN-Suliman heuristic approach provides better results than NEH. Also, Suliman heuristic starting from sequence generated by neural network gives better results than that starting from sequence given by CDS, in most of the problems. But for the problem size 50, NEH is found to give better solution than ANN-Suliman heuristic, because of directionality constraint imposed by the Suliman Heuristic. Hence, to improve the quality of solution further, the result of ANN is initialized to GA. ANN-GA approach has performed better than all the other approaches. The present study reveals that there is considerable scope to use other methods of improving solution such as Simulated Annealing, Tabu Search etc. Also, combination of 5, 6 and 7 jobs is used in the present study in training exemplars with the 2:3:5 proportions. Different proportions can also be tried to improve the initial solution quality of ANN.

Acknowledgments The authors are thankful to the reviewers whose comments and suggestions have been valuable to improve the earlier version of the paper.

References

Akyol, D. (2004). Application of neural networks to heuristic scheduling algorithms. *Computers & Industrial Engineering, 46*, 679–696.

Arroyo, J. E. C., & Armentano, V. A. (2005). Genetic local search for multiobjective flowshop scheduling problems. *European Journal of Operational Research, 167*(3), 717–738.

Campbell, H. R., & Smith, D. M. (1970). A heuristic algorithm for n-jobs m-machines sequencing problem. *Management Science, 16*, 630–637.

Chen, C.-L., Vempati, V. S., & Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research, 80*, 389–396.

Dannenbring, D. (1977). An evolution of flowshop scheduling heuristics. *Management Science, 23*, 174–1182.

El-bouri, A., Subramaniam, B., & Popplewell, N. (2000). Sequencing jobs on a single machine: A neural network approach. *European Journal of Operational Research, 126*, 474–490.

El-bouri, A., Subramaniam, B., & Popplewell, N. (2005). A neural network to enhance local search in the permutation flowshop. *Computers & Industrial Engineering, 49*, 182–196.

French, S. (1982). *Sequencing and scheduling—an introduction to the mathematics of the job-shop*. Chichester, UK. West Sussex, England: Ellis Horwood.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129.

Gupta, J. (1971). A functional heuristic algorithm for the flow shop scheduling problem. *Operational Research Quarterly, 22*, 27–39.

Ho, J. C., & Chang, Y.-L. (1991). A new heuristic for the *n*-jobs, *m*-machine flowshop scheduling problem. *European Journal of Operational Research, 52*, 194–202.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigian Press.

Johnson, S. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*(1), 61–68.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Koulamas, C. (1998). A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research Society, 105*, 66–71.

Lee, Shaw, M. J. (2000). A neural-net approach to real time flow shop sequencing. *Computers & Industrial Engineering, 38*, 125–147.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics, 21*, 1087–1092.

Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering, 30*(4), 1061–1071.

Nawaz, M., Enscore, J., & Ham, I. (1983). A Heuristic algorithm for the mmachine, n-job flow shop sequencing problem. OMEGA. *International Journal of Management Science, 11*(1), 91–95.

Nearchou, A. C. (2004). Flow shop sequencing using hybrid simulated annealing. *International Journal of Intelligent Manufacturing, 15*, 317–328.

Neppalli, V. R., Chen, C.-L., & Gupta , J. N. D. (1996). Genetic algorithms for the two stage bicriteria flow shop problem. *European Journal of Operational Research, 95*(2), 356–373.

Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research, 91*(10), 160–175.

Ogbu, F. A., & Smith, D. K. (1990). The application Of the simulated annealing algorithm to the solution of the N/M/Cmax flow shop problem. *Computers &Operations Research, 17*, 243–253.

Osman, I. H., & Potts, C. N. (1989). Simulated annealing for permutation flow shop scheduling. *Omega, 17*(6), 551–557.

Palmer, D. (1965). Sequencing jobs through a multi-stage. Process in the minimum total time—a quick method of obtaining a near optimum. *Operations Research, 16*, 45–61.

Rajendran, C. (1995). Theory and methodology heuristics for scheduling in flow shop with multiple objectives. *European Journal of Operational Research, 82*, 540–555.

Reeves, C. (1995). A genetic algorithm for flow shop sequencing. *Computers & Operations Research, 22*, 5–13.

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

Sridhar, J., & Rajendran, C. (1996). Scheduling in flowshop and cellular manufacturing systems with multiple objectives—a genetic algorithmic approach. *Production Planning and Control, 7*, 374–382.

Suliman, S. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics, 64*, 143–152.

Taillard, E. (1990). Some efficient heuristic methods for flow shop sequencing problem. *European Journal of Operational Research, 47*, 65–74.

Taillard, E. (1993). Benchmark for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Tang, L., Liu, W., & Liu, J. (2005). A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. *Journal of Intelligent Manufacturing, 16*, 361–370.

Temiz, I., & Erol, S. (2004). Fuzzy branch and bound algorithm for flow shop scheduling. *International Journal of Intelligent Manufacturing, 15*, 449–454.

Wang, L., & Zheng, D. Z. (2003). A modified evolutionary programming for flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 22*(7/8), 522–527.

Wang, L., & Zheng , D. Z. (2003). An effective hybrid heuristic for flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 21*(1), 38–44.

Wang, L., Zhang, L., & Zheng, D. Z. (2003). A class of order-based genetic algorithm for flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 22*(11/12), 828–835.